ERAD 2018

XVIII Escola Regional de Alto Desempenho

4 a 6 de Abril de 2018

INF - UFRGS - Porto Alegre - RS

Organização

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

UNIVERSIDADE FEDERAL DE PELOTAS
RS-BRASIL

FURG

PUCRS

Universidade Federal de Santa Maria
SEDES SAPIENTIAE
1960

Realização

SBC
Sociedade Brasileira de Computação

CRAD

ERAD/RS 2018

18ª Escola Regional de Alto Desempenho
do Estado do Rio Grande do Sul

ERAD/RS 2018

4 a 6 de abril de 2018
Porto Alegre, RS, Brasil

ANAIS

Editora
Sociedade Brasileira de Computação – SBC

Edição
Lucas Mello Schnorr (UFRGS)
Mauricio Aronne Pillon (UDESC)

Execução
Universidade Federal do Rio Grande do Sul – UFRGS

Patrocínio Governamental
FAPERGS

Patrocínio Empresarial
TechDec, SDC, Laniaq

Realização
Sociedade Brasileira de Computação – SBC

Capa: Lucas Mello Schnorr
Supervisão Gráfica: Lucas Mello Schnorr

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Escola Regional de Alto Desempenho do estado do Rio Grande do Sul (18.: 4-6 abril 2018: Porto Alegre, RS)
Anais / Edição: Lucas Mello Schnorr, Mauricio Aronne Pillon. – Porto Alegre, RS, Brasil: SBC/UFRGS, 2018.
290 f.: il.

ISSN 2177-0085

Conhecido também como ERAD/RS 2018

1. Processamento de Alto Desempenho. 2. Arquiteturas de Computadores. 3. Processamento Paralelo e Distribuído. I. ERAD/RS (18.: 4-6 abril 2018: Porto Alegre, RS). II. UFRGS. III. Mello Schnorr, Lucas. IV. Aronne Pillon, Mauricio. V. Título.

APRESENTAÇÃO

Bem-vindos à décima oitava edição da Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul, a ERAD/RS 2018.

A ERAD/RS é um evento anual, promovido pela Sociedade Brasileira de Computação, por meio da Comissão Regional de Alto Desempenho do Rio Grande do Sul (CRAD-RS), desde 2001. O objetivo deste encontro, de caráter essencialmente regional, abrangindo a região sul do Brasil, é de qualificar profissionais da região nas áreas que compõem o Processamento de Alto Desempenho (PAD) e de prover um fórum regular onde se possa tanto apresentar os avanços recentes nessas áreas quanto discutir as formas de ensino de processamento de alto desempenho nas universidades do sul do Brasil. Para potencializar seus resultados, este evento é itinerante, sendo abrigado, a cada ano, por uma instituição diferente. Neste ano coube à Universidade Federal do Rio Grande do Sul (UFRGS) gerenciar a execução deste encontro, que será realizado em Porto Alegre – RS, com o apoio da Universidade do Estado de Santa Catarina (UDESC), da Universidade Federal de Pelotas (UFPel), da Universidade Federal do Rio Grande (FURG), da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), da Sociedade Educacional Três de Maio (SETREM), e da Universidade Federal de Santa Maria (UFSM).

O Estado do Rio Grande do Sul é, há muitos anos, referência no país na área de Processamento de Alto Desempenho. A ERAD/RS reflete este fato e, mais ainda, reforça esta posição de destaque quando se preocupa em formar novos pesquisadores e em manter atualizados os pesquisadores que criaram as bases nos estados do sul. Portanto, é com satisfação que nos envolvemos neste já tradicional evento, nos tornando parte dessa história e auxiliando o desenvolvimento científico do País. E é nesse momento que reconhecemos a presença de instituições que acompanham a evolução desta história colaborando de forma efetiva para sua realização. Neste ano, a ERAD/RS conta com fomento da FAPERGS e com o patrocínio da TechDec, SDC e Laniaq.

A programação da ERAD/RS 2018 contará com cinco palestras, sete minicursos, a primeira maratona de programação paralela da sequência desta escola, os fóruns de iniciação científica e pós-graduação com apresentação de cerca de 80 trabalhos, além do tradicional painel onde se discute de maneira provocativa as tendências da área de alto desempenho. Neste ano, a ERAD/RS é organizada em até três tracks paralelas. O Fórum de Iniciação Científica deste ano é coordenado pelos professores Alexandre Caríssimi (UFRGS) e Odorico Mendizabal (FURG). O Fórum de Pós-Graduação é coordenado pelos professores Gerson Geraldo H. Cavalheiro (UFPEL) e Dalvan Griebler (PUCRS/SETREM). A maratona de programação paralela é coordenada pelo professor João Vicente Ferreira Lima (UFSM). Neste volume encontram-se o texto de um minicurso,

o resumo dos demais, bem como os trabalhos selecionados para os Fórum de Iniciação Científica e para o Fórum de Pós-Graduação.

Agradecemos, em especial, a todos os autores que submeteram trabalhos às sessões técnicas e aos convidados diversos que aceitaram nosso convite e vêm honrar nossa Escola com suas presenças. Por fim, agradecemos a prontidão com que diversos colegas do corpo multi-institucional deste evento, que tomaram para si diversos encargos e os conduziram a termo com pleno sucesso.

Obrigado pela presença de todos e aproveitem a ERAD/RS 2018.

Lucas, Mauricio
Coordenadores da ERAD/RS 2018
Porto Alegre, 4 de abril de 2018

ERAD/RS 2018

18ª Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul

Comitê Organizador

Coordenação Geral

Lucas Mello Schnorr (UFRGS)

Mauricio Aronne Pillon (UDESC)

Coordenação do Fórum de Pós-Graduação

Gerson Geraldo H. Cavalheiro (UFPEL)

Dalvan Griebler (PUCRS/SETREM)

Coordenação do Fórum de Iniciação Científica

Alexandre Caríssimi (UFRGS)

Odorico Mendizabal (FURG)

Coordenação da Maratona de Programação Paralela

João Vicente Ferreira Lima (UFSM)

Comitê de Programa do Fórum de Pós-Graduação

Adenauer Yamin (UCPEL/UFPEL)
Andriele Busatto do Carmo (PUCRS)
Afonso Sales (PUCRS)
Alexandre Carissimi (UFRGS)
Andre Du Bois (UFPEL)
Andre Martinotto (UCS)
Andrea Charão (UFSM)
Carlos Holbig (UPF)
Charles Miers (UDESC)
Claudio Geyer (UFRGS)
Claudio Schepke (UNIPAMPA)
Cristiano Costa (UNISINOS)
Dalvan Griebler (PUCRS)
Douglas Macedo (UFSC)
Edson L. Padoin (UNIJUI)
Eduardo Henrique Molina da Cruz (UFRGS)
Francieli Zanon Boito (UFRGS)
Fabio Diniz Rossi (IFFar)
Gerson Geraldo H. Cavalheiro (UFPEL)
Guilherme Galante (UNIOESTE)
Guilherme Koslovski (UDESC)
João V. F. Lima (UFSM)
Laércio L. Pilla (UFSC)
Lucas Mello Schnorr (UFRGS)
Luiz Gustavo Leão Fernandes (PUCRS)
Marcelo Neves (PUCRS)
Marcelo Rebonatto (UPF)
Mauricio Aronne Pillon (UDESC)
Marcia Pasin (UFSM)
Márcio Castro (UFSC)
Mario Dantas (UFSC)
Mauricio Pilla (UFPEL)
Odorico Machado Mendizabal (FURG)
Patricia K. V. Mangan (UNILASALLE)
Patricia Plentz (UFSC)
Philippe O. A. Navaux (UFRGS)
Rodrigo Righi (UNISINOS)
Sandro Camargo (UNIPAMPA)
Tiago Ferreto (PUCRS)

Comitê de Programa do Fórum de Iniciação Científica

Adenauer Yamin (UCPEL/UFPEL)
Afonso Sales (PUCRS)
Alexandre Carissimi (UFRGS)
Andre Du Bois (UFPEL)
Andre Martinotto (UCS)
Andrea Charão (UFSM)
Antônio Carlos S. Beck (UFRGS)
Arthur Lorenzon (UFRGS)
Benhur Stein (UFSM)
Carlos Holbig (UPF)
Carlos Rolim (UFRGS)
Cesar Zeferino (Univali)
Charles Miers (UDESC)
Claudio Schepke (UNIPAMPA)
Cristiano Costa (UNISINOS)
Daniel S. Marcon (UNISINOS)
Dalvan Griebler (PUCRS/SETREM)
Edson Luiz Padoin (UNIJUI)
Eduardo Cruz (UFRGS)
Eduardo Roloff (UFRGS)
Emilio Francesquini (UNICAMP)
Fábio D. Rossi (IF Farroupilha)
Fauzi Shubeita (SETREM)
Francieli Z. Boito (UFSC)
Gerson Geraldo H. Cavalheiro (UFPEL)
Guilherme Galante (UNIOESTE)
Guilherme Koslovski (UDESC)
Henrique C. Freitas (PUC Minas)
Jerônimo Ramos (UFPEL)
João V. F. Lima (UFSM)
Julio César S. dos Anjos (UFRGS)
Laércio L. Pilla (UFSC)
Leonardo Pinho (UNIPAMPA)
Lucas Mello Schnorr (UFRGS)
Luiz Gustavo L. Fernandes (PUCRS)
Marcelo Rebonatto (UPF)
Marcia Pasin (UFSM)
Marcio Castro (UFSC)
Marco Alves (UFPR)
Matheus S. Serpa (UFRGS)
Matthias Diener (UFRGS)
Mauricio A. Pillon (UDESC)
Odorico M. Mendizabal (FURG)
Patricia P. Barcelos (UFSM)
Patricia K. V. Mangan (UNILASALLE)
Patricia Plentz (UFSC)
Pedro B. Marcos (FURG)
Raffael Schemmer (IFRS)
Renata Reiser (UFPEL)
Rodrigo Righi (UNISINOS)
Samuel Feitosa (UFSM)
Sandro Camargo (UNIPAMPA)
Tiago Ferreto (PUCRS)
Valderi Leithardt (UNIVALI)

Comitê Organizador Local

Claudio Geyer (UFRGS)
Lucas Mello Schnorr (UFRGS)
Philippe O. A. Navaux (UFRGS)

Equipe de voluntários

Breno Zanchetta (UFRGS)
Gabriel Job Antunes Grabher (UFRGS)
Jobe Diego Dylbas dos Santos (UFRGS)
Kassiano Jose Matteussi (UFRGS)
Matheus S. Serpa (UFRGS)
Lucas Leandro Nesi (UFRGS)
Pablo Pavan (UFRGS)
Vinícius Garcia Pinto (UFRGS)

SBC

Sociedade Brasileira de Computação

CRAD/RS

Comissão Regional de Alto Desempenho do Rio Grande do Sul

Membro Honorário

Prof. Tiaraju Asmuz Diverio (UFRGS)

Comissão Executiva

Prof. Andre Dubois, UFPEL
Profa. Andrea Charão, UFSM
Prof. Claudio Schepke, UNIPAMPA
Prof. Lucas Mello Schnorr, UFRGS (coordenador)
Prof. Rodrigo Righi, UNISINOS

Comissão Deliberativa

Instituição	Representantes
CESUP RS	Denise Ewald, Magali Longhi
Embrapa C.Temperado	Nelsi Warken
Feevale	Edvar B. Araújo
FURG	Odorico Machado Mendizabal
IFSUL	João Ladislau B. Lopes, Rodrigo Santos de Souza
PUCRS	Luiz Fernandes, Tiago Ferreto
SENAC	Carlos Vinicius Rasch Alves. Angelo Gonçalves Luz
SETREM	Fauzi Shubeita, Dalvan Griebler
UCPEL	Adenauer Yamin, Cláudio Machado Diniz
UCS	André Luis Martinotto, Ricardo V. Dorneles
UFPEL	André Du Bois, Gerson G. H. Cavalheiro
UFRGS	Alexandre da Silva Carissimi, Philippe O. A. Navaux
UFSM	Marcia Pasin, João V. F. Lima
ULBRA	Roland Teodorowitsch
UNICRUZ	Alessandro Copetti
UNIFRA	Ana Paula Canal
UNIJUI	Edson Padoin
UNIPAMPA	Claudio Schepke, Leonardo Bidese de Pinho
UNIRITTER	Atila Vasconcelos, Mozart Siqueira
UNISC	Daniela Saccol Peranconi
UNISINOS	Cristiano André da Costa, Rodrigo R. Righi
UPF	Carlos A. Hölbig, Marcelo T. Rebonatto
URI - Santo Ângelo	Carlos Oberdan Rolim

SUMÁRIO

MINICURSOS

1
Arquitetura e Programacao de GPUs em CUDA

Esteban Clua[1]

Resumo:

Neste mini-curso será apresentado uma visão geral das arquiteturas mais recentes das GPUs NVIDIA, dando-se especial atenção a Volta e Pascal. Em seguida será apresentada a linguagem CUDA, destacando aspectos referentes a estas novas arquiteturas, bem como caracteristicas do CUDA 9. Serão feitos pequenos exercicios, especialmente para que os alunos possam aprender gerar pequenos projetos para GPUs.

[1]Esteban Clua é professor da Universidade Federal Fluminense e coordenador geral do UFF Medialab, jovem cientista do nosso Estado pela FAPERJ em 2009 e 2013. Pesquisador CNPq PQ2. Possui graduação em Computação pela Universidade de São Paulo, mestrado e doutorado em Informática pela PUC-Rio. Sua área de atuação está especialmente focada na área de Video Games, Realidade Virtual, GPUs e visualização. É um dos fundadores do SBGames (Simpósio Brasileiro de Games e Entretenimento Digital), tendo sido presidente da Comissão Especial de Jogos da SBC entre 2010 e 2014. Atualmente é o representante para o Brasil do Technical Committee de Entretenimento Digital da International Federation of Information Processing (IFIP) e membro honorário do conselho diretivo da Abragames (Associação Brasileira de Desenvolvimento de Games). Em 2015 foi nomeado NVIDIA Fellow (único na America Latina). Em 2007 recebeu o prêmio da ABRAGAMES como o maior contribuidor da academia para a indústria de jogos digitais no Brasil. Esteban é membro do comitê de programa das mais importantes conferencias na área de entretenimento digital. Atualmente é coordenador do Centro de Excelência da NVIDIA no Brasil, que funciona no instituto de computação da Universidade Federal Fluminense. Esteban é do conselho de inovação da Secretaria da Cultura do Estado do Rio de Janeiro, membro da comissão permanente do Rio Criativo, Membro do Fórum permanente de Inovação e Tecnologia da Assembléia Legislativa do Rio de Janeiro e membro do conselho da Agencia de Inovação da UFF.

2

Deep Learning: Image Classification with DIGITS

João Paulo Peçanha Navarro[1]

Resumo:

Deep learning is giving machines near human levels of visual recognition capabilities and disrupting many applications by replacing hand-coded software with predictive models learned directly from data. This lab introduces the machine learning workflow and provides hands-on experience with using deep neural networks (DNN) to solve a real-world image classification problem. You will walk through the process of data preparation, model definition, model training and troubleshooting, validation testing and strategies for improving model performance. You will also see the benefits of GPU acceleration in the model training process. On completion of this lab you will have the knowledge to use NVIDIA DIGITS to train a DNN on your own image classification dataset.

[1]João Paulo Peçanha Navarro é Cientista da Computação graduado pela Universidade Federal de Juiz de Fora (UFJF) e mestre em Modelagem Computacional pela mesma instituição. Por 4 anos foi membro do Grupo de Computação Gráfica da UFJF, se envolvendo em pesquisas nas áreas de Computação Visual, Simulação Física e Computação de Alto Desempenho. Trabalhou como Analista de Sistemas na empresa de consultoria Accenture, principalmente em projetos voltados para área financeira e de risco de mercado. Sua sólida experiência como engenheiro de software foi construída durante anos desenvolvendo sistemas na PUC-Rio, com foco em aplicações de Machine Learning para a indústria de Óleo & Gás. É autor de publicações em conferências internacionais nas áreas de Machine Learning e Computação Gráfica, suas especialidades. Atualmente é Arquiteto de Soluções na NVIDIA, responsável pela expansão e democratização de técnicas de Deep Learning no Brasil.

3

Intel Modern Code: Introdução à Programação Vetorial e Paralela para o Processador Intel Xeon Phi Knights Landing

Matheus da Silva Serpa[1],
Vinícius Garcia Pinto[2],
Philippe Olivier Alexandre Navaux[3]

Resumo:

Tradicionalmente o aumento de desempenho das aplicações se dava de forma transparente aos programadores devido ao aumento do paralelismo a nível de instruções e aumento de frequência dos processadores. Entretanto, este modelo não se sustenta mais. Atualmente para se ganhar desempenho nas arquiteturas modernas, é necessário conhecimentos sobre programação paralela e programação vetorial. Ambos paradigmas são tratados de forma superficial em cursos de computação, sendo que muitas vezes nem são abordados. Neste contexto, este tutorial objetiva propiciar um maior entendimento sobre os paradigmas de programação paralela e vetorial, de forma que os participantes aprendam a otimizar adequadamente suas aplicações para arquiteturas modernas. Como plataforma de desenvolvimento, serão utilizados processadores Intel Xeon Phi Knights Landing.

[1]Matheus da Silva Serpa é atualmente Mestrando no Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande do Sul (UFRGS). É bacharel em Ciência da Computação pela Universidade Federal do Pampa (UNIPAMPA), tendo recebido o Prêmio Destaques UNIPAMPA (2015) e o Prêmio Aluno Destaque da SBC (2016). Suas principais áreas de pesquisa são Arquitetura de Computadores e Computação de Alto Desempenho.

[2]Vinícius Garcia Pinto se graduou em Ciência da Computação na Universidade Federal de Santa Maria (UFSM) e concluiu seu mestrado na Universidade Federal do Rio Grande do Sul (UFRGS). Atualmente é doutorando na UFRGS em co-tutela com a Universidade Grenoble Alpes - França. Sua linha de pesquisa envolve programação paralela, otimização de aplicações paralelas para arquiteturas CPU/GPU e análise de desempenho.

[3]Philippe Olivier Alexandre Navaux é Professor Titular na Universidade Federal do Rio Grande do Sul (UFRGS) desde 1973. Se graduou em Engenharia Elétrica na UFRGS em 1970. Recebeu seu mestrado pela UFRGS em 1973 e seu doutorado pela Instituto de Tecnologia de Grenoble (INPG), na França, em 1979. Ele é o coordenador do Grupo de Processamento Paralelo e Distribuído (GPPD) na UFRGS e consultor de várias entidades de fomento nacionais e internacionais como DoE (US), ANR (FR), CNPq (BR), CAPES (BR), entre outras.

4

Introdução à programação paralela em Fortran usando OpenMP e MPI

Henrique Gavioli Flores[1],
Alex Lima de Mello[2],
Marcelo Trindade Rebonatto[3]

Resumo:

Fortran é uma linguagem utilizada em projetos de cunho científico/matemático, tanto por ser competente neste tipo de tarefa como por ser compatível com versões anteriores de 'legacy software'. Este tipo de problema é ideal para ser resolvido com o auxílio de paralelismo. Visando isso, torna-se importante o domínio do funcionamento dessas tecnologias, buscando uma melhora no desempenho em problemas complexos. Serão abordadas as bibliotecas OpenMP e MPI, descrevendo sua forma de funcionamento e uso com Fortran.

[1]Henrique Gavioli Flores é formado em Ciência da Computação pela Universidade de Passo Fundo (UPF) - 2011/1 - 2014/2, atualmente analista de Sistemas da Divisão de Tecnologia da Informação da UPF desde 2013/2 e mestrando do Programa de Pós-Graduação em Computação Aplicada da UPF desde 2016/1

[2]Alex Lima de Mello é formado em Ciência da Computação pela Universidade de Passo Fundo (UPF), e atualmente mestrando do Programa de Pós-Graduação em Computação Aplicada da UPF.

[3]Marcelo Trindade Rebonatto é formado em Ciência da Computação pela Universidade de Passo Fundo (UPF), mestre pela Universidade Federal do Rio Grande do Sul (UFRGS) junto ao GPPD, e doutor pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) em 2014. É professor na Universidade de Passo Fundo desde 1996. Membro do Programa de Pós-graduação em Computação Aplicada (PPGCA) da UPF desde 2014.

4.1. Introdução

A linguagem de programação FORmula TRANslation (FORTRAN), desenvolvida na IBM entre os anos de 1954 e 1957, é considerada uma das primeiras linguagens de programação de alto nível (HLL: High Level Language). O objetivo era produzir uma linguagem de fácil interpretação mas, ao mesmo tempo, com uma eficiência muito próxima à linguagem *assembly*. Fortran não inventou o conceito de HLL, mas foi uma das primeiras linguagens a usá-lo, e entre elas, a que obteve o maior sucesso.

Com o passar do tempo, os computadores foram evoluindo, assim como as linguagens de programação. Muito dessa evolução foi pressionada pela demanda crescente de aplicações com elevadas capacidades computacionais, ampliando os domínios onde os computadores podem auxiliar a sociedade, em especial nas aplicações científicas. Nesse contexto, o processamento paralelo surge como opção real para proporcionar um aumento na capacidade computacional, proporcionado uma efetiva redução no tempo de processamento e, em alguns casos, a ampliação das opções exploradas. Assim, surge a necessidade da programação paralela de computadores.

A programação paralela explícita, onde o programador declara de forma clara e sem ambiguidades as instruções para realizar e controlar o paralelismo pode ser realizada de diversas formas. Atualmente, uma expressiva parcela das aplicações paralelas utilizam bibliotecas para a exploração do paralelismo, como por exemplo OpenMP e MPI. Outros recursos estão sendo agregados, como o uso de diretivas de compilação, dividindo o trabalho entre o programador e o compilador, como por exemplo OpenMP e OpenACC[4].

Message Passing Interface (MPI) é um padrão para comunicação de dados na computação paralela. É possível obter comunicação de dados entre processos sendo apenas de um processo para outro ou comunicação coletiva entre os processos. O objetivo do MPI é a portabilidade, alto desempenho e escalabilidade das aplicações [14].

Open Multiple-Processing (OpenMP) é uma interface de programação com objetivo de prover multiprocessamento em uma aplicação. Esse recurso permite que uma aplicação executando em um único fluxo se divida em vários, estes realizando o processamento designado e se unam novamente um único fluxo [8].

Este capítulo tem por objetivo introduzir a programação paralela usando as bibliotecas OpenMP e MPI com a linguagem de programação Fortran. Ele é destinado as pessoas que conheçem Fortran e desejam se inserir no ambiente de programação paralela e também aos profissionais da área de computação que desejam conhecer um pouco mais dessa instigante linguagem.

4.2. Fortran

Fortran é uma linguagem de programação proposta por John W. Backus para a IBM no ano de 1957, com intuito de desenvolver uma alternativa mais prática para linguagem Assembly, utilizada para programar o computador IBM 704. Os desenvolvedores que trabalharam no projeto da linguagem foram Richard Goldberg, Sheldon F. Best, Harlan Herrick, Peter Sheridan, Roy Nutt, Robert Nelson, Irving Ziller, Lois Haibt e David Sayre. A linguagem foi adotada por cientistas para a escrita de programas com uma base

[4]No caso de OpenACC, a exploração do paralelismo utiliza recursos das Unidades de Processamento Gráfico (GPU) e extrapola os objetivos desse texto.

matemática relativamente complexa, influenciando os desenvolvedores de compiladores para o desenvolvimento de ferramentas de compilação que pudessem otimizar cada vez mais os códigos [11].

Por ser uma linguagem que sobrevive por 60 anos, ela passou por diversas atualizações e versões. Em alguns casos, as atualizações foram pequenos ajustes, porém em outros as mudanças foram profundas. Um breve histórico das versões do Fortran é descrito a seguir [3, 9].

- Fortran I (1954-1957): O compilador Fortran I manteve o recorde de traduzir um código, por mais de 20 anos;
- Fortran II (1958): Capacidade de compilar módulos de programas, não executáveis, para serem "link editados" com outros programas;
- Fortran III (1958): Não saiu do laboratório;
- Fortran IV (1966) ou Fortran66 (1966): Implementação dos comandos COMMON e EQUIVALENCE. Foi o primeiro compilador oficialmente padronizado;
- Fortran77 (1977): Foi padronizado utilizando o conceito de programação estruturada;
- Fortran90 (1990): Atualização do Fortran77 que levou 12 anos para ser efetuada. Vários recursos do Fortran90 se aproximam aos existentes na linguagem C (alocação dinâmica de memória, apontadores e orientação ao objeto). Marca o final da codificação em cartão perfurado;
- HPF (1990): High Performance Fortran para executar em ambientes com memória distribuída. Extensão ao Fortran90;
- Fortran95 (1996): ajustes para se aproximar ao HPF;
- Fortran 2003 (2003): especificação que incorpora, por exemplo, o controle de exceções e programação orientada a objetos.
- Fortran 2008 (2010) pequenos melhoramentos, incorpora correções ao Fortran 2003 e novas funcionalidades.

Diversos fatores podem ser apontados para o uso em larga escala da linguagem Fortran. Atualmente existem diversos projetos de sistemas computacionais em uso que são escritos nessa linguagem. Como exemplo, pode-se citar o modelo de previsão numérica do tempo Eta [20], um modelo de previsão regional que auxilia pesquisas em diversas áreas, como agrícola e energética.

Apesar de ser uma linguagem de programação consideravelmente antiga, Fortran é largamente utilizada até hoje por pesquisadores que necessitam realizar grandes operações matemáticas e de grande complexidade [10].

4.2.1. OpenMP

OpenMP é uma API de programação paralela com memória compartilhada proposta na Supercomputing Conference de 1997, com o objetivo de propor uma alternativa para unificar os diversos modelos de programação paralela que estavam surgindo [2]. Foi montado o OpenMP Architecture Review Board(ARB) para melhor especificar, gerenciar, promover e oferecer suporte ao até então informal padrão OpenMP[13]. No mesmo ano é lançada a versão 1.0 das especificação OpenMP para Fortran e, no ano seguinte (1998) surgem as primeiras aplicações híbridas com OpenMP e MPI [19].

A unificação das especificações para C/C++ e Fortran é iniciada em 2002 e concluída em 2005, com o término da versão 2.5, ainda em 2005 ocorreu o primeiro International Workshop on OpenMP (IWOMP). Atualmente OpenMP se encontra na versão 4.5 com vários novos recursos e conceitos, e a versão 5.0 ja está sendo construída.

OpenMP trabalha utilizando o conceito fork/join, que permite que uma aplicação executando em um único fluxo (thread master), se divida (fork) em várias threads que realizam o processamento de informações simultaneamente e se unam (join) novamente um único fluxo, como demonstra a Figura 4.1

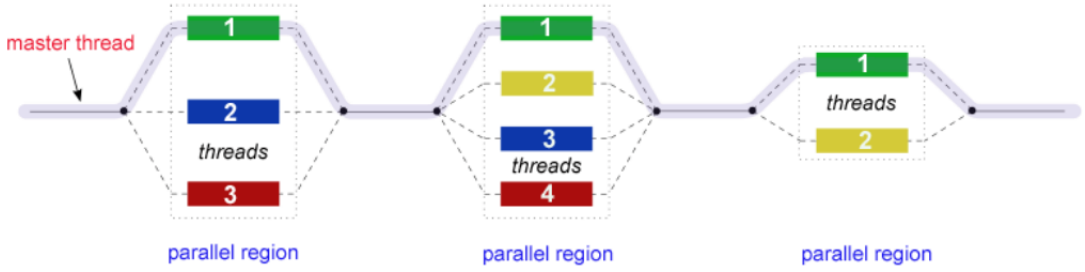


Figura 4.1: Exemplo de fluxo fork/join.

A tecnologia OpenMP tem como vantagens sua simplicidade, uma vez que a decomposição das operações é realizada de forma automática por meio das diretivas. De modo geral, não é preciso grandes modificações nos códigos fonte para realizar a integração das diretivas.

OpenMP possui alguns contras em sua utilização, sendo eles, o fato que aplicações só são executadas de forma eficiente em multiprocessadores com memória compartilhada, a escalabilidade das aplicações é limitada pela arquitetura da memória e esta carece de mecanismos refinados para controle de mapeamento thread-processador.

4.2.2. MPI

O padrão Message Passing Interface (MPI) surgiu a partir do problema da falta de interoperabilidade de programas paralelos, acentuado no final da década de 1980. Com o objetivo de criar padrões para troca de mensagens(Message Passing) foi realizado um workshop no Centro de Pesquisa em Computação Paralela de Williamsburg, em abril de 1992. Um grupo de trabalho foi criado para dar continuidade ao processo de padronização e em novembro de 1992 foi criado o MPI Forum, reunindo aproximadamente 175 pessoas de 40 organizações. O grupo de trabalho do MPI continuou a se reunir e buscando uma base mais formal ao padrão e produziu uma proposta para o padrão MPI1. Foi decidido colocar o processo de padronização adotando a organização e procedimentos do High Performance Fortran Forum (HPF). O *draft* do MPI1 foi revisado e finalizado em fevereiro de 1993, então o padrão MPI foi apresentado na conferência Supercomputing 93, sendo a partir dela a versão oficial do MPI de 5 de maio de 1994.

Desde a criação o MPI passou por correções e melhoramentos, como a gerada a partir do Forum MPI do Supercomputing 94, que possibilitou a criação do MPI-2 em 1995, sendo apresentada seu *draft* no SuperComputing 96, aprovado e lançado em 1997. As melhorias não param, sendo a versão 3.0 (2012), uma extensão do MPI 2.2 de 2009, com esclarecimentos e novas rotinas [7].

O MPI rapidamente se tornou reconhecido como padrão de fato e de direito para a programação paralela e de alto desempenho em ambientes de memória distribuída. Alguns fatores contribuíram para esse sucesso: **a)** mais de uma implementação de boa qualidade disponível gratuitamente; **b)** grupos de comunicação sólidos, eficientes e determinísticos; **c)** buffers de mensagens gerenciados de forma ágil; **d)** uso numa ampla variedade de arquiteturas paralelas (incluindo multiprocessadores) eficientemente; **e)** fortemente especificado; **f)** portável [1].

O funcionamento do MPI se dá por uma aplicação constituída de um ou mais processos que se comunicam através de troca de mensagens. É possível que cada processo execute funções diferentes ao mesmo tempo, se existir mais de uma unidade processadora (core) disponível. Normalmente, os desenvolvedores não utilizam mais processos que o número de núcleos disponível no processador, pois isto reduziria o ganho de desempenho, já que os processos ficariam realizando troca de contexto para processar [6].

As rotinas MPI podem ser utilizadas para criar paralelismo nas linguagens Fortran, C e C++. Desta forma, a programação em Fortran e o uso das chamadas MPI se tornam relativamente simples de ser utilizadas, bastando apenas a inclusão da biblioteca e uso de compiladores específicos.

4.2.3. Compiladores Fortran com OpenMP e MPI

Fortran sempre foi uma linguagem voltada a aplicações científicas. Seu uso em aplicações de alto desempeno (paralelas) foi apenas uma consequência. A exploração do paralelismo em aplicações, usando MPI e OpenMP vem se difundindo cada dia mais. Acrescentando o uso de GPUs com CUDA (ou OpenACC) tem-se as ferramentas de programação paralela mais usadas nos dias de hoje. Atualmente, há diversas opções de compiladores Fortran que podem ser utilizados em conjunto de OpenMP ou MPI. A lista a seguir mostra alguns dos principais compiladores Fortran que podem ser usados, com suporte a MPI e OpenMP.

- gfortran (GNU)
- pgfortran (PGI)
- ifort (Intel)
- XL Fortran (IBM)
- Cray (Cray)

Alguns dos compiladores listados acima não oferecem compatibilidade completa com MPI e OpenMP em todas as suas versões, mas possuem pelo menos uma com suporte completo. O uso de OpenMP vem, por padrão habilitado no compilador, sendo necessário que se informe uma *flag* para habilitar o uso de OpenMP. No caso do MPI, é necessário que uma implementação do MPI (MPICH ou OpenMPI, por exemplo) esteja disponível e, em alguns casos, se necessita escolher o compilador a usar com o MPI. Este curso vai se concentrar no uso de OpenMP e MPI com a linguagem Fortran, deixando a parte de instalação e configuração para os manuais de referência. Os exemplos aqui passados serão baseados em ambiente com processador x86, Sistema Operacional (SO) Linux de 64bits, gfortran e OpenMPI.

4.3. Programação com OpenMP

O OpenMP utiliza o recurso de *pragma*, reconhecidos apenas por compiladores compatíveis. Com os *pragma*, os programadores indicam ao compilador as opções de paralelismo e de controle, e o compilador fica responsável por "embutir" o código necessário para a realização dessas tarefas. Além dos *pragma*, um conjunto de funções auxiliam o programador a desenvolver seu código.

Em Fortran os *pragma* OpenMP são identificados por "!$", e servem para definir o início de uma diretiva OpenMP, neste trabalho serão mostrados algumas diretivas consideradas básicas, explicando seu funcionamento.

Para compilar um programa fazendo uso de OpenMP, é necessário utilizar a flag identificando o uso da API, que varia de acordo com o compilador utilizado. Para o compilador gfortran, a flag que indica o uso de OMP é "-fopenmp", exemplificado na Figura 4.2.

```
gfortran {-fopenmp} {fonte.f90} [-o binário] [parâmetros]   (a)

gfortran -fopenmp primo.f90 -o primo                        (b)

gfortran -fopenmp raiz.f90 -o raiz -lm                      (c)

                          Legenda
        {  } Obrigatório        [   ]Opcional
```

Figura 4.2: Exemplo de compilação utlilizando OpenMP com o compilador GFortran..

Tem-se na Figura 4.2a o esquema geral de compilação de um programa utilizando OpenMP, os únicos argumentos obrigatórios são a flag que indica o uso de OpenMP e o nome do código fonte. O uso de '-o' é opcional e serve para alterar o nome do arquivo de saída. A Figura 4.2b mostra um exemplo de uso comum do compilador com a flag OpenMP de compilação ativada. Como apresentado na figura 4.2c, pode-se incluir outros parâmetros caso desejado. A execução é igual a qualquer outro programa.

4.3.1. Funções básicas do OpenMP

OpenMP possui quatro funções básicas que são utilizadas para controle de threads, são elas:

- OMP_GET_NUM_THREADS(): Retorna o número de threads utilizadas em uma seção paralela;
- OMP_GET_MAX_THREADS(): Retorna o número de threads disponível;
- OMP_SET_NUM_THREADS(nthreads): Define o número de threads a ser usado pelas seções paralelas,
- OMP_GET_THREAD_NUM(): Retorna o número identificador(ID) da thread. as IDs variam de 0(thread master) até OMP_GET_NUM_THREADS() -1.

A Figura 4.3 ilustra o funcionamento das funções descritas acima. É importante notar que as funções OMP_GET_THREAD_NUM(), OMP_GET_NUM_THREADS() e OMP_GET_MAX_THREADS() precisam ser declaradas como EXTERNAL (linha 4).

```
1.  PROGRAM  basicfunc
2.    IMPLICIT NONE
3.    INTEGER :: nthreads, myid, maxthreads
4.    INTEGER, EXTERNAL :: OMP_GET_THREAD_NUM, &
           OMP_GET_NUM_THREADS, OMP_GET_MAX_THREADS

5.     maxthreads= OMP_GET_MAX_THREADS()
6.     print *, "max threads = ",maxthreads
7.     call OMP_SET_NUM_THREADS(5)

8.    !$OMP PARALLEL private(nthreads, myid)
9.        myid = OMP_GET_THREAD_NUM()
10.      print *, "Hello from thread ", myid

11.       !$OMP MASTER
12.          nthreads = OMP_GET_NUM_THREADS()
13.          print *, "Number of threads = ", nthreads
14.       !$OMP END MASTER
15.    !$OMP END PARALLEL
16. END
```

(a)

```
$ gfortran -fopenmp basicfunc.f90 -o bf
$ ./bf
max threads = 8
Hello from thread 2
Hello from thread 0
Hello from thread 1
Hello from thread 3
Number of threads = 5
Hello from thread 4
```

(b)

Figura 4.3: Exemplo de utilização da funções básicas..

Na linha 5 da Figura 4.3a, é utilizada a função OMP_GET_MAX_THREADS(), que como podemos ver na figura 4.3b, tem como resultado o valor oito, significando que o processador utilizado possui oito threads. Em seguida (linha 7 da figura 4.3a) é definido o máximo de threads a serem usadas como cinco, assim, a seção paralela a seguir será executada por cinco threads, com IDs variando de zero a quatro, como mostrado na figura 4.3b. Na linha 11 da figura 4.3a, é selecionado o processo de ID zero para calcular e mostrar na tela o número de threads utilizadas.

4.3.2. Blocos de código para execução em paralelo

O objetivo principal do OpenMP é realizar a execução em paralelo de programas. Isso pode ser realizado com o par **!$OMP PARALLEL** e **!$OMP END PARALLEL**, que marca, respectivamente, o início e o fim de um bloco a ser processado em paralelo. É importante definir quais variáveis serão privadas (PRIVATE) ou compartilhadas (SHARED) entre as threads.

As variáveis privadas possuem valor indefinido ao iniciar o processamento da seção paralela, onde é criada uma cópia das mesmas para cada thread. Ao terminar a seção paralela, a variável privada não mantém o valor da seção paralela, pois como possui um valor único para cada thread não é possível definir o valor que deveria ser atribuído. Há também outros dois tipos de variáveis privadas, são eles FIRSTPRIVATE e LASTPRIVATE. FIRSTPRIVATE inicia a seção paralela com o valor que possuía na sequencial,

enquanto o LASTPRIVATE mantém ao sair da seção paralela o valor que teria caso a mesma fosse executada de forma sequencial, como por exemplo, a última iteração de um laço de repetição. A diferença entre os tipos PRIVATE e FIRSTPRIVATE pode ser notada na linha 13 da figura 4.4a e na linha correspondente da figura 4.4b.

```
1.  program vartype
2.     integer:: a=735,b=735,myid
3.     call OMP_SET_NUM_THREADS(1)

4.     print*,a
5.     !$OMP PARALLEL PRIVATE(a)
6.        print*,a
7.        a=10
8.     !$OMP END PARALLEL
9.     print*,a
10.    print*,char(10)

11.    print*,b
12.    !$OMP PARALLEL FIRSTPRIVATE(b)
13.       print*,b
14.       b=10
15.    !$OMP END PARALLEL
16.    print*,b
17. end program vartype
                                        (a)
- - - - - - - - - - - - - - - - - - - - - - -
$ gfortran -fopenmp vartype.f90         (b)
$ ./a.out
735
0
735

735
735
735
```

Figura 4.4: Exemplo do uso de variáveis PRIVATE e FIRSTPRIVATE.

Variáveis do tipo PRIVATE iniciam a seção paralela sem um valor definido, como pode ser visto na Figura 4.4b, onde tem seu valor alterado para 10, e o mesmo é perdido ao sair da seção. No entanto, a variável FIRSTPRIVATE inicia o processamento paralelo com o valor que possuía anteriormente, porém o mesmo é perdido ao terminar a seção.

Quando for necessário que apenas uma thread execute um deteminado código, pode-se utilizar um dos seguintes *pragma*:

- !$OMP MASTER e !$OMP END MASTER;
- !$OMP SINGLE e !$OMP END SINGLE.

Com uso do par de *pragma* **!$OMP MASTER** e **!$OMP END MASTER** é definida uma seção a ser executada apenas pela thread master, ou seja, a que possui ID igual a zero(OMP_GET_THREAD_NUM()==0). Um exemplo de seu uso pode ser observado nas linhas 11 e 14 da figura 4.3a.

Ao ser utilizado o par de *pragma* **!$OMP SINGLE** e **!$OMP END SINGLE**, o funcionamento e sintaxe é semelhante a **!$OMP MASTER**, porém o trecho de código é executado apenas pela primeira thread a alcançar a seção, independentemente de sua ID.

4.3.3. Divisão do processamento entre as threads

A divisão do processamento a ser realizado pelas threads pode ser controlada usando um dos seguintes *pragma*:

- !$OMP DO e !$OMP END DO;
- !$OMP SECTIONS e !$OMP END SECTIONS.

Pode ser usado o par **!$OMP DO** e **!$OMP END DO** para delimitar um trecho de código, composto da iteração do loop DO. Os índices do laço são divididos entre as threads. É importante notar que a variável de controle de um loop (step), é definida como PRIVATE por padrão, não sendo necessário especificar seu escopo de compartilhamento. Um exemplo de uso do **!$OMP DO** e **!$OMP END DO** pode ser observada das linhas 12 a 17 da figura 4.5a.

```
1.  program main
2.     real :: step,x,soma,pi=0.0
3.     integer:: i,id
4.     integer ::num_steps=1000000,num_threads
5.     integer, EXTERNAL:: OMP_GET_THREAD_NUM, &
    OMP_GET_NUM_THREADS, OMP_GET_MAX_THREADS

6.     step=1.0/num_steps
7.     call OMP_SET_NUM_THREADS(4)
8.     num_threads= OMP_GET_NUM_THREADS()
9.     !$OMP PARALLEL PRIVATE(x, soma)
10.       id = OMP_GET_THREAD_NUM()
11.       soma=0
12.       !$OMP DO
13.          DO i=id,num_steps
14.             x = (i+0.5)*step
15.             soma = soma+4.0/(1.0+x*x)
16.          END DO
17.       !$OMP END DO
18.       !$OMP CRITICAL
19.          pi=pi+soma
20.       !$OMP END CRITICAL
21.    !$OMP END PARALLEL

22.    print*,pi/num_steps
23. end program main
                                                (a)
- - - - - - - - - - - - - - - - - - - - - - - - - - -
$ gfortran -fopenmp pi.f90 -o pi                (b)
$ ./pi
 3.14144874
```

Figura 4.5: Exemplo do uso de !$OMP DO e !$OMP CRITICAL.

Pode ser usado a união dos *pragma* **!$OMP PARALLEL** e **!$OMP DO** (e seus delimitadores finais) para declarar de uma só vez seção paralela e o laço de repetição na mesma linha.

Com o objetivo de criar uma seção crítica para as threads em execução num ponto do código, pode-se utilizar o par de diretivas **!$OMP CRITICAL <nome>** e **!$OMP END CRITICAL <nome>** para indicar que as threads devem executar a seção uma de cada vez, ou seja, nunca haverá duas threads executando ao mesmo tempo aquele código. O campo <nome> não é necessário, porém é recomendado seu uso, uma vez que, caso diferentes seções críticas possuírem o mesmo nome, serão tratadas como uma só, o mesmo se aplica caso o nome não seja definido. Um exemplo onde uma seção crítica é necessária é onde o resultado de uma operação é acumulado em uma variável compartilhada, como pode ser visto na linha 19 da figura 4.5a, evitando que haja conflito entre diferentes threads atualizando a mesma variável simultaneamente.

O par de diretivas **!$OMP SECTIONS e !$OMP END SECTIONS** é utilizado para definir tarefas diferentes a serem executadas por cada thread. O código a ser excecutado em cada thread deve ser precedido por *!$OMP SECTION* sendo finalizado por outra diretiva igual ou por um **!$OMP END SECTIONS**. Podem ser especificadas quantas seções for desejado, sendo cada uma delas executada numa thread diferente.

4.4. Programação com MPI

Nesta seção, os conceitos básicos para uso de aplicações MPI em programas Fortran serão abordados. Além disso, as formas de compilação e execução bem como a ilustração de códigos fonte funcionais serão disponibilizados.

4.4.1. Compilação e Execução

As implementações de MPI normalmente, ao serem instaladas, fornecem uma série de *scripts* que facilitam tanto a compilação quanto a execução de programas MPI. Considerando um computador com SO Linux, Ubuntu 16.04, os seguintes pacotes devem estar presentes: 'libopenmpi-dev' e 'openmpi-bin' para o MPI[5].

O MPI foi criado para ser utilizado com memória distribuída, porém pode-se utilizá-lo apenas numa única máquina. Para sua utilização em mais de um computador, deve-se configurar um sistema para comunicação remota, usando *rsh* ou *ssh*. Maiores detalhes podem ser visualizados em [16].

A partir dos pacotes acima instalados, os scripts *mpif90* e *mpiexec* ficam disponíveis para serem utilizados. A Figura 4.6 mostra algumas possíveis formas de compilação e execução.

```
mpif90 {fonte.f90} [-o binário] [parâmetros] (a)

mpif90 primo.f90 -o primo                    (b)

mpif90 raiz.f90 -o raiz -lm                  (c)

mpiexec {-n x binário} [parâmetros] (d)

mpiexec -n 6 primo                  (e)

mpiexec -n 4 raiz 5 1234567         (f)

                  Legenda
                  {   } Obrigatório          [   ] Opcional
```

Figura 4.6: Compilação e execução de programas com MPI.

A Figura 4.6a, mostra o esquema geral de utilização do script *mpif90* (há também *scripts* para outras versões de Fortan), onde deve-se identificar o arquivo fonte a ser compilado. Após, pode-se indicar o binário resultante do processo de compilação e linkedição, colocando o argumento '-o' seguido do 'nomedobinario'. Este parâmetro não é obrigatório, mas sua utilização é recomendada, uma vez que a sua omissão irá gerar o arquivo de saída padrão (geralmente o arquivo 'a.out'). Exemplos de uso podem ser visualizados na Figura 4.6b e na Figura 4.6c, sendo que nesta última é acrescentado um argumento (podem ser mais).

A Figura 4.6d ilustra o esquema geral de utilização do *script* 'mpiexec' (*mpirun* também pode ser utilizado) na execução de programas paralelos com MPI, onde se deve indicar o nome do binário a ser executado e o número de processos a serem criados, através do parâmetro '-n x', onde 'x' é o número de processos a serem criados. A Figura 4.6e descreve um exemplo comum de uso. Pode-se informar argumentos a aplicação sendo executada através da linha de comando (Figura 4.6f).

[5]O pacote 'gfortran' deve estar instalado, para uso do compilador Fortran.

4.4.2. Conceitos de MPI

O padrão MPI oferece um conjunto de rotinas e uma padronização em relação ao funcionamento das trocas de mensagens. O número de funções varia de acordo com a versão do MPI e ultrapassa o número de 100 funções. Porém, para a compreensão do funcionamento de aplicações MPI, um número reduzido de funções já é suficiente. Para conhecer o funcionamento da escrita de programas com MPI deve-se compreender: a) Processos MPI; b) Comunicadores; c) Mensagens [15].

A execução dos programas com MPI ocorre com o disparo em paralelo de seus processos. Geralmente, os processos MPI são disparados com o auxílio de um script (vide seção 4.4.1.). A identificação dos processos MPI em execução é realizada através do *rank*, onde há uma identificação cada processo. Ele é único e individual em cada processo MPI, tendo seu valor atribuído no inicio da execução do programa. Os *ranks* variam de 0 até N-1, onde N é o número de processos com que o programa é executado [5].

Os processos MPI organizam-se em grupos ordenados, possibilitando uma comunicação eficiente. A comunicação entre os processos do grupo é realizada por meio de comunicadores, que nada mais são do que "um objeto local que representa o domínio (contexto) de uma comunicação (conjunto de processos que podem ser conectados)"[4]. Por padrão, todos os processos fazem parte de um grupo único associado ao comunicador pré-definido identificado por MPI_COMM_WORLD.

Para a comunicação o MPI definiu um formato de mensagens, divididos em duas partes: dados e envelope (Figura 4.7). A parte dos dados corresponde ao endereço de memória a ser enviado/recebido, o tipo do dado sendo comunicado e a quantidade de elementos na mensagem. O envelope deve conter a identificação do processo (a receber ou para quem enviar, utilizando para isso o *rank* do processo), o assunto da mensagem (em MPI definido por tag) e o comunicador associado [14].

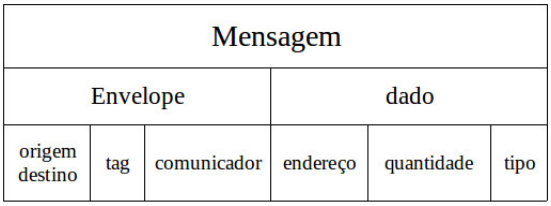


Figura 4.7: Composição das mensagens MPI.

Os tipos de dados das mensagens MPI não são tipos padrão da linguagem de programação utilizada. Ao invés, o MPI usa tipos próprios de dados para permitir a portabilidade. Esse fato é devido a uma implementação não homogênea dos tipos de dados na grande variedade de computadores os quais o MPI é portado. A Tabela 4.1 ilustra alguns dos tipos de dados utilizados no MPI e sua relação com os disponíveis na linguagem Fortran [12].

Os dois últimos tipos apresentados na 4.1, MPI_BYTE e MPI_PACKED, não possuem correspondente direto na linguagem Fortran. O primeiro, representa um inteiro de 8 bits. O segundo é útil quando se faz necessária a comunicação de dados não continuamente agrupados na memória, uma vez que quando informa-se uma quantidade de elementos, eles por padrão, devem estar em área contínua na memória. Há também um conjunto de tipos de dados opcionais e a possibilidade da criação de tipos de dados.

Tabela 4.1: Relação dos tipos de dados MPI com Fortran.

Tipo Dado MPI	Tipo dado Fortran
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

4.4.3. Uso do MPI em programas Fortran

Por ser um padrão de programação paralela, o uso do MPI com Fortran é muito similar ao uso com C/C++. Porém, algumas diferenças em relação ao uso com C/C++ podem ser encontradas, muitas em virtude das próprias linguagens. Por exemplo, Fortran não é 'case sensitive', por outro lado C/C++ é. Outra diferença é que em Fortran, se deve incluir a biblioteca 'mpif.h' (ou 'use module MPI') ao invés de 'mpi.h' (em C/C++).

A principal diferença, em relação a sintaxe é o fato que em Fortran, todas as chamadas MPI são sub-rotinas (e não funções, como C/C++). Dessa forma, as chamadas MPI em Fortran são acrescidas de um argumento: o código de retorno da chamada, um valor inteiro. Esse argumento é o último de cada chamada, não alterando a ordem dos demais argumentos. Por ser utilizada uma chamada de sub-rotina, as chamadas MPI em Fortran devem ser precedidas por 'CALL'.

Um conceito muito comum na introdução a programação MPI é o MPI-BÁSICO, que nada mais é do que um conjunto de chamadas MPI responsáveis por: inicio e término do ambiente MPI, gerenciamento de processos e envio/recebimento de mensagens síncronas ponto a ponto. A lista a seguir mostra as chamadas.

- MPI_INIT(mpierr): Inicializa o MPI;
- MPI_COMM_RANK(comm, rank, mpierr): Obtém o Rank do Processo;
- MPI_COMM_SIZE(comm, procs, mpierr): Obtém a quantidade de Processos em execução;
- MPI_SEND(msg, count, datatype, dest, tag, comm, mpierr): Envia uma mensagem;
- MPI_RECEIVE(msg, count, datatype, dest, tag, comm, status, mpierr): Recebe uma mensagem;
- MPI_FINALIZE(mpierr): Finaliza o MPI.

Todos os argumentos podem ser mapeados para inteiros, diferente de alguns objetos como *comm* e *status* em C/C++. O primeiro argumento das funções de envio e de recebimento indicam referências a memória de um tipo de dados, de acordo como o 'datatype' especificado (Tabela 4.1).

Um programa simples, com Fortran e MPI pode ser visualizado na Figura 4.8. Nele não são usadas funções de comunicação explícita de dados.

Na linha 2 da Figura 4.8a é incluída a biblioteca do MPI para a utilização das suas funcionalidades. As linhas 4 e 8 realizam o início e término do ambiente MPI no programa. Na linha 5 é usada uma chamada do MPI para atualizar a variável 'size' com a quantidade de processos executados. A linha 6 realiza a obtenção da identificação do processo, sendo armazenada na variável 'rank', por meio de outra chamada MPI.

```
1.   program hello
2.   include 'mpif.h'
3.   integer rank, size, ierror

4.   call MPI_INIT(ierror)
5.   call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
6.   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
7.   print*, 'node', rank, 'from', size,': Hello world'
8.   call MPI_FINALIZE(ierror)
9.   end                                                    (a)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
$ mpif90 primeiro.f90 -o primeiro
$ mpiexec -n 4 primeiro
node 2 from    4 : Hello world
node 3 from    4 : Hello world
node 0 from    4 : Hello world
node 1 from    4 : Hello world                              (b)
```

Figura 4.8: Programa "Ola Mundo" em MPI.

Os valores resultantes nas variáveis manipuladas nas linhas 5 e 6 podem ser visualizados através da execução da linha 7, onde os mesmos são mostrados através da função 'print'. Esses valores são dependentes da execução. Um exemplo de compilação e execução é demonstrado na Figura 4.8b, onde o programa é executado com quatro processos.

É importante salientar que o programa fonte da Figura 4.8a possui uma chamada da função 'print' gerando apenas uma linha de texto na saída. Porém em sua execução (Figura 4.8b) podem ser visualizadas quatro linhas de texto. Isso ocorre em virtude da execução do programa em quatro processos, onde em cada a linha 7 (print) é executada, sendo a saída redirecionada. Mesmo que sejam quatro linhas resultantes, todas possuem valores distintos para a identificação do processo (rank), enquanto que o número de processos sendo executados (size) é o mesmo.

Esse redirecionamento da saída padrão (ocorre também com a entrada) é realizado a partir do MPI [14], independentemente se os processos criados pelo 'mpiexec' executem num mesmo computador ou não. A saída gerada em outros processos é comunicada pela rede até o computador que realizou o disparo do programa e dessa forma não é garantida a chegada em ordem das saídas geradas. A Figura 4.8b ilustra este fato, pois a saída do computador de rank = 2 aparece antes do gerado no com rank = 1.

4.4.4. Enviando e recebendo mensagens

O programa ilustrado na Figura 4.8 serve para introduzir a programação com o MPI, porém a comunicação entre os processos não é contemplada. A comunicação entre processos MPI ocorre através de trocas de mensagens, conforme o modelo da Figura 4.7. É preciso explicitar as trocas de mensagens entre os processos por meio de chamadas MPI para envio e recebimento (Figura 4.9).

O primeiro argumento da função de envio de mensagens é uma referência a uma posição de memória onde os dados a serem enviados se encontram. Em seguida, deve-se indicar a quantidade de elementos e o tipo dos dados, definidos pelo MPI (Tabela 4.1). O próximo argumento do envio é a identificação (rank) do processo que deverá receber

a mensagem, seguido do tag (assunto) e finalizado pelo comunicador a ser utilizado. A função de recebimento possui praticamente os mesmos argumentos com apenas a adição de um para status e uma modificação no significado do primeiro e do quarto argumentos. No primeiro argumento ao invés de indicar uma área de memória de onde os dados irão ser enviados, deve-se indicar uma área de memória onde os dados devem ser recebidos. O quarto argumento ao invés de ser indicado o rank do destino deve-se indicar de quem (origem) a mensagem deverá ser recebida. O penúltimo argumento da função de recebimento, deve-se indicar uma variável para armazenar o status do recebimento, onde poderão ser encontradas informações de controle relativas a mensagem recebida.

```
1.   program segundo
2.   include 'mpif.h'
3.   integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)
4.   integer i, envia, recebe

5.   call MPI_INIT(ierror)
6.   call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
7.   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
8.   if (rank == 0) then
9.      DO i = 1, size-1, 1
10.         call MPI_RECV(recebe, 1, MPI_INTEGER, i,    &
                          tag, MPI_COMM_WORLD, status, ierr)
11.         print*, 'Received ', recebe, ' from ', i
12.     END DO
13.  else
14.     envia = rank * 123
15.     call MPI_SEND(envia, 1, MPI_INTEGER, 0,         &
                      tag, MPI_COMM_WORLD, ierr)
16.  endif
17.  call MPI_FINALIZE(ierror)
18.  end                                                  (a)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
$ mpif90 segundo.f90 -o segundo
$ mpiexec -n 4 segundo
Received  123  from 1
Received  246  from 2
Received  369  from 3                                    (b)
```

Figura 4.9: Programa envolvendo trocas de mensagens em MPI.

A inicialização e identificação dos processos que ocorre nas linhas 5 a 7 da Figura 4.9 é análoga à utilizada no exemplo da Figura 4.8a. Na linha 8 começam a ser identificadas diferenças no código a ser executado, através de uma expressão de condição onde a variável de identificação de cada processo 'rank' é verificada. A expressão verificada na linha 8 somente será verdadeira para o processo com rank = 0, dessa forma, somente um processo irá executar a porção de código compreendida entre as linhas 9 e 12 (inclusive). Os demais processos irão ter a condição da linha 8 com o resultado falso, assim, irão executar as linhas 14 e 15 e não irão executar as linhas 9 a 12.

A troca de mensagens está localizada no trecho de programa não executado de forma igual por todos os processos. O envio das mensagens ocorre na linha 10 enquanto o processo de recebimento esta localizado na linha 15. Na prática, os processos que possuem o *rank* diferente de zero enviam uma mensagem ao processo que possui *rank* igual a zero. A mensagem é composta de um valor do tipo MPI_INTEGER, tendo seu valor

associado ao *rank* do processo multiplicado de 123 (linha 14). Para completar a troca de mensagens, o processo com rank = 0 realiza o recebimento das mensagens enviadas, mostrando o conteúdo recebido (linha 11) e de quem foi recebido.

A garantia de que todas as mensagens serão recebidas é o *loop* na linha 9 até a 12. O valor inicial da variável de controle do laço 'i' é um (1), sendo a cada iteração acrescido de uma unidade, enquanto o número de processos executados seja menor ou igual a variável 'size' subtraído de 1. Esta variável de controle do laço é utilizada para definir a origem das mensagens recebidas na função de recebimento MPI_RECV (linha 15, quarto argumento). Dessa forma, primeiro será recebida a mensagem do processo 1, para após receber a mensagem do processo 2 e assim por diante. A Figura 4.9b ilustra a execução do programa fonte da Figura 4.9a com quatro processos. Na primeira linha de saída a mensagem do processo com rank = 1, seguidas pelas linhas oriundas dos demais processos, em ordem crescente de valor do rank de cada processo.

No exemplo de envio/recebimento de mensagens da Figura 4.9, onde o processo com rank = 0 recebe mensagens dos demais, ocorre uma especificação de ordem do recebimento das mensagens. Isso é definido na linha 10, pelo quarto argumento onde é especificada a variável 'i', controlada pelo laço, forçando primeiro o recebimento da mensagem do processo com rank = 1, após a do rank = 2 e assim sucessivamente. Porém, nem sempre esta situação de especificação de quem receber a mensagem primeiro é útil, podendo ocasionar espera desnecessária dos processos (bloqueio temporário). Como exemplo, pode-se citar um processo que distribua tarefas aos demais e após o recebimento de um resultado ele deva enviar outra tarefa a quem lhe enviou o resultado (típico caso de algoritmo mestre/escravo).

Para resolver a limitação supra-citada o MPI oferece no recebimento das mensagens a possibilidade da não especificação de quem enviou a mensagem ou do assunto (tag). Ao invés, utiliza-se coringas no lugar da especificação destes campos.

- O coringa 'MPI_ANY_SOURCE' pode ser utilizado no lugar de quem enviou a mensagem, causando o recebimento da mensagem de qualquer processo;
- O coringa 'MPI_ANY_TAG' pode ser utilizado para especificar que a mensagem a ser recebida pode ser de qualquer assunto (tag).

A programação de aplicações com uso de coringas proporciona uma maior flexibilidade no recebimento das mensagens. Porém, de pouco adiantaria o recebimento destas mensagens com os coringas sem poder identificar, após o seu recebimento, quem a enviou ou qual o seu assunto. Para isso, usando Fortran 90 e o MPI, pode-se usar estrutura associada ao recebimento das mensagens para recuperar estas informações. Esta estrutura é ligada a variável de *status* no recebimento das mensagens (penúltimo argumento). A Figura 4.10 contém as modificações do programa de envio e recebimento de mensagens (Figura 4.9) para uso de coringas.

```
    . . .
10.     call MPI_RECV(recebe, 1, MPI_INTEGER, MPI_ANY_SOURCE,  &
                          tag, MPI_COMM_WORLD, status, ierr)
11.     print*, 'Received ', recebe, ' from ', status(MPI_SOURCE)
    . . .
```

Figura 4.10: Modificações da Figura 4.9 para uso de coringas.

Conforme pode ser visualizado na Figura 4.10, as modificações a serem realizadas são poucas, apenas na linha 10 e na linha 11. Na chamada MPI que realiza o recebimento da mensagens (linha 10), o argumento 'i' é substituído pelo coringa MPI_ANY_SOURCE, representando que o recebimento poderá ser de qualquer processo origem. Na linha 11, é usada a função 'status', passando como argumento MPI_SOURCE, retornando assim quem enviou a mensagem.

4.4.5. Comunicando conjuntos de dados

Uma das características geralmente presentes nos programas paralelos é a manipulação de grandes conjuntos de dados. Estes conjuntos precisam ser enviados/recebidos pelos processos MPI a fim de que sejam processados. Por exemplo, caso exista a necessidade de processar um vetor de 1000 elementos em quatro processos MPI, o ideal é que cada um destes processos execute a computação sobre 250 elementos do vetor. No exemplo da Figura 4.9a, teríamos de gerar 1000 envios/recebimentos para a distribuição dos dados do vetor a ser processado, causando lentidão na execução por uso ineficiente de recursos.

O MPI fornece três mecanismos para a comunicação de conjuntos de dados: o argumento de quantidade de elementos, tipos derivados e o empacotamento de mensagens. No caso do exemplo do vetor de 1000 posições, o argumento de quantidade de elementos a serem enviados/recebidos resolve o problema [14]. A Figura 4.11 ilustra um exemplo onde são comunicados conjuntos de dados. Salienta-se que as demais alternativas do MPI podem ser utilizadas para comunicação de vetores, porém são mais indicadas para aplicações onde os dados não estão continuamente alocados na memória.

```
1. program terceiro
2. include 'mpif.h'
3. integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)
4. integer i, parte, ind, vet(0:11)

5. call MPI_INIT(ierror)
6. call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
7. call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
8. parte = 12 / (size-1)
9. if (rank == 0) then
10.     DO i = 0, 11, 1
11.         vet(i) = i
12.     END DO
13.     DO i = 1, size-1, 1
14.         call MPI_SEND(vet(ind), parte, MPI_INTEGER, i, &
                      tag, MPI_COMM_WORLD, ierr)
15.         ind = ind + parte
16.     END DO
17. else
18.     call MPI_RECV(vet, parte, MPI_INTEGER, 0,          &
                  tag, MPI_COMM_WORLD, status, ierr)
19.     print *, 'Processo', rank
20.     DO i = 0, parte-1, 1
21.         print *, vet(i)
22.     END DO
23. endif
24. call MPI_FINALIZE(ierror)
25. end                                                  (a)
```

```
$ mpiexec -n 3 terceiro
Processo  1
     0
     1
     2
     3
     4
     5
Processo  1
     6
     7
     8
     9
     10
     11                  (b)

$ mpiexec -n 4 terceiro
Processo  1
   0
   1
   2
   3
 Processo     2
   4
   5
   6
   7
 Processo     3
   8
   9
   10
   11                  (c)
```

Figura 4.11: Programa de envio/recebimento de conjuntos de dados.

Na Figura 4.11a, é definido na linha 4, um vetor de números inteiros 'vet' com 12 elementos (os índices variam de 0 a 11), dessa forma, todos os processos MPI que serão executados possuem este vetor. A linha 8, executada por todos os processos, calcula a

parte do vetor a ser comunicada tendo como variáveis o tamanho do vetor e o número de processos ativos menos um. Esta subtração se faz necessária uma vez que no algoritmo projetado, o processo com rank = 0 envia o vetor 'aos pedaços' enquanto os demais processos recebem a porção equivalente do vetor e a mostram. Convém lembrar que dependendo do número de elementos do vetor e de processos executados, o cálculo da linha 8 pode gerar um número onde nem todo o vetor seja enviado. No uso com um programa não didático, como o da Figura 4.11a, deve-se levar em consideração esses ajustes, o que extrapola o escopo deste texto.

As linhas 10 a 12 da Figura 4.11a realizam a inicialização do vetor com dados, ação executada apenas no processo com rank = 0. O envio de pedaços do vetor ocorre da linha 13 a linha 16, sendo enviadas mensagens em ordem para os processos, ou seja, primeiro a mensagem para o processo com rank = 1, após para o com rank = 2 e assim sucessivamente. Na linha 14 encontra-se a chamada MPI para envio da mensagem, tendo como destaques o primeiro e segundo argumentos. O primeiro indica a partir de qual posição da memória o vetor começará a ser enviado enquanto o segundo indica quantos elementos serão enviados (calculado na linha 8). O endereço de envio das mensagens é controlado pela variável 'ind', inicializada com 0, ou seja, para o primeiro processo serão enviadas 'parte' posições a partir do endereço da posição 0 do vetor. Logo após, a variável 'ind' é atualizada (linha 15) com seu valor acrescido da quantidade de elementos enviados para que as posições subseqüentes possam ser referenciadas num próximo envio.

Entre as linhas 18 e 22 da Figura 4.11a são realizadas as ações de recebimento das mensagens contendo um conjunto de dados, sendo executadas por todos os processos com *rank* diferente de 0. A função de recebimento está colocada na linha 18, sendo que o primeiro e segundo argumentos são os que merecem maior atenção. O primeiro argumento identifica a posição de memória que os dados devem ser copiados quando forem recebidos. Nele é colocado o vetor 'vet', que sem a identificação de índice indica que devem ser colocados a partir da primeira posição do vetor (índice 0). Serão colocadas 'parte' (segundo argumento do MPI_Recv) elementos neste vetor, não sendo preenchidas todas suas posições. Conforme já discutido, o vetor existe em todos os processos com seu tamanho completo (12). Esta é uma técnica que facilita a programação, porém ocupa mais memória que o necessário. O ideal neste caso seria a alocação dinâmica de uma área de memória para o vetor em cada um dos processos, alocando apenas a quantidade de memória a ser utilizada. Após o recebimento cada um dos processos mostra os valores recebidos (linhas 19 a 22).

Nas Figuras 4.11b e 4.11c são mostradas execuções do código fonte da Figura 4.11a com 3 e 4 processos respectivamente. Pode-se notar que a quantidade de elementos do vetor nos processos muda de acordo com o número de processos inicializados (4 com 4 processos e 6 com 3 processos). Isso ocorre devido cálculo efetuado na linha 8.

4.5. Considerações Finais

Este texto descreveu a utilização da linguagem de Programação Fortran para o desenvolvimento de programas paralelos, com o auxílio das bibliotecas OpenMP e MPI. Os conceitos introdutórios para iniciar desenvolvedores Fortran no uso da programação paralela ou desenvolvedores com experiência em programação paralela e distribuída na linguagem C/C++ foram apresentados.

Por ser um curso introdutório, muitos tópicos poderiam ser ampliados: no caso de OpenMP, uma série de *pragma* e opções não foram trabalhados. Como exemplo, pode-se citar o uso de operações de redução global (clausula reduction). O mesmo ocorre com a parte de MPI, onde diversas chamadas não foram exploradas, em especial as de comunicações coletivas.

4.6. Bibliografia

[1] CÁCERES, Edson N.; SONG, Siang W. "Algoritmos Paralelos usando CGM/PVM/MPI: Uma Introdução"(2004). Texto preparado para o XXI Congresso da Sociedade Brasileira da Computação, Jornada de Atualização em Informática. https://www.ime.usp.br/ song/papers/jai01.pdf>. Acesso Fev. 2018.

[2] CHAPMAN, Barbara; Jost, Gabrielle: PAS, R. V. D. "Using OpenMP: Portable Shared Memory Parallel Programming". https://goo.gl/3zYPrs. Acesso Fev. 2018.

[3] Universidade Estadual de Campinas; Centro Nacional de Processamento de Alto Desempenho São Paulo "Apostila de Treinamento: Introdução ao MPI (2012). https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_fortran90.pdf Acesso em Mar. 2018.

[4] Universidade Estadual de Campinas; Centro Nacional de Processamento de Alto Desempenho São Paulo. "Apostila de Treinamento: Introdução ao Fortran 90"(2012). https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_MPI.pdf Acesso em Fev. 2018.

[5] Costa, Celso Maciel et. al."Programação concorrente: Threads, MPI e PVM"(2002). In: Escola Regional de Alto Desempenho ERAD, 2002. Anais ... São Leopoldo: SBC/Instituto de Informática da UFRGS/UNISINOS/ULBRA. p.31-65.

[6] Gabriel, E.; et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation"(2004). In Proceedings, 11th European PVM/MPI Users' Group Meeting (2004), pp. 97-104.

[7] IGNÁCIO, Aníbal Alberto Vilcapona; FERREIRA FILHO, Virgílio José Martins. "MPI: Uma ferramenta para implementação paralela"(2002). Pesquisa Operacional. Rio de Janeiro, v.22, n.1, p.105-116, janeiro a junho de 2002. http://dx.doi.org/10.1590/S0101-74382002000100007

[8] Hermanns, M. "Parallel Programming in Fortran 95 using OpenMP"(2002). School of Aeronautical Engineering". Universidad Politécnica de Madrid. Spain.

[9] ISO-IEC JTC1/SC22/WG5. "The Home of Fortran Standards"(2018). https://wg5-fortran.org/ Acesso Mar. 2018.

[10] Jorgensen, Ed. "Introduction to Programming using Fortran 95"(2014). Las Vegas: University Of Nevada, 177 p.

[11] McJones, Paul. "History of FORTRAN and FORTRAN II"(2017), http://www.softwarepreservation.org/projects/ FORTRAN/index.html#By_FORTRAN_project_members Acesso em: Out. 2017.

[12] MPI FORUM "Message data"(2009). http://mpi-forum.org/docs/mpi-2.2/mpi22-report/node44.htm Acesso em: Jan 2018.

[13] OpenMP. "Welcome to the OpenMP ARB"(2014). http://www.openmp.org/wp-content/uploads/OpenMP-WelcomeGuide.pdf Acesso em: Jan. 2018.

[14] Pacheco, Peter S. Parallel programming with mpi. San Francisco: Morgan Kaufmann, 1997. 418 p.

[15] Rebonatto, M. T. "Introdução à programação paralela com MPI em agregados de computadores (clusters)"(2004). In: Anais do CBCOMP 2004, p. 939-955.

[16] Squyres, Jeff. "MPI newbie: Requirements and installation of an MPI"(2013). https://blogs.cisco.com/performance/mpi-newbie-requirements-and-installation-of-an-mpi Acesso em: Jan. 2018.

[17] Boulic, R. and Renault, O. (1991) “3D Hierarchies for Animation”, In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

[18] Dyer, S., Martin, J. and Zulauf, J. (1995) “Motion Capture White Paper”,Introdução à programação paralela com MPI em agregados de computadores (clusters). : , 2004, v. http://reality.sgi.com/employees/jam_sb/mocap/MoCapWP_v2.0.html, December.

[19] Cownie, J; Duran, A.; Klemm, M.;Lin L. "An OpenMP* Timeline"(2018). The Parallel Universe, issue 18 () https://software.intel.com/sites/default/files/managed/6a/78/parallel_mag_issue18.pdf/ Acesso Mar. 2018.

[20] Mello, A; Flores, H; Rebonatto, M; Hölbig, C; "Aprimoramento do Modelo Eta Utilizando Recursos de Computação Paralela e Distribuída."(2017). http://www.lbd.dcc.ufmg.br/colecoes/erad/2017/067.pdf Acesso em: Jan. 2018.

5
Introdução ao MPI-IO

Jean Luca Bez[1]**,**
Francieli Zanon Boito[2]**,**
Philippe Olivier Alexandre Navaux[3]

Resumo:

Na Computação de Alto Desempenho (High-Performance Computing), as operações de entrada e saída (I/O) são um gargalo para um número crescente de aplicações, podendo comprometer a escalabilidade. Em ambientes de larga escala, como clusters e supercomputadores, o uso de bibliotecas de alto-nível como MPI-IO permite obter melhor desempenho através de I/O paralelo e de operações coletivas. Estas possibilitam o uso transparente de técnicas de otimização como data sieving e two-phase I/O. Nesse contexto, este minicurso objetiva introduzir conceitos de I/O paralelo e MPI-IO, com um enfoque prático, de forma que os participantes aprendam a utilizar a biblioteca em suas aplicações.

[1]Jean Luca Bez se graduou em Ciência da Computação na Universidade Regional Integrada do Alto Uruguai e das Missões (URI - Erechim) em 2015 e concluiu seu mestrado na Universidade Federal do Rio Grande do Sul (UFRGS) em 2017. Atualmente é doutorando na UFRGS. Suas principais áreas de pesquisa são I/O parallelo, I/O forwarding e escalonamento de operações de I/O.

[2]Francieli Zanon Boito é Doutora em Ciência da Computação pela UFRGS e pela Université Grenoble Alpes, na França (2015), e Bacharel em Ciência da Computação pela UFRGS (2009). Atualmente realiza um pós-doutorado no INRIA em Grenoble, na França. Suas principais áreas de pesquisa são I/O paralelo, sistemas de arquivos paralelos e escalonamento de operações de I/O.

[3]Philippe Olivier Alexandre Navaux é Professor Titular na Universidade Federal do Rio Grande do Sul (UFRGS) desde 1973. Se graduou em Engenharia Elétrica na UFRGS em 1970. Recebeu seu mestrado pela UFRGS em 1973 e seu doutorado pela Instituto de Tecnologia de Grenoble (INPG), na França, em 1979. Ele é o coordenador do Grupo de Processamento Paralelo e Distribuído (GPPD) na UFRGS e consultor de várias entidades de fomento nacionais e internacionais como DoE (US), ANR (FR), CNPq (BR), CAPES (BR), entre outras.

6
Introduction to GPU Programming with OpenACC

Pedro Mário Cruz e Silva[1]

Resumo:

Learn how to accelerate your C/C++ or Fortran application using OpenACC to harness the massively parallel power of NVIDIA GPUs. OpenACC is a directive based approach to computing where you provide compiler hints to accelerate your code, instead of writing the accelerator code yourself. In 90 minutes, you will experience a four-step process for accelerating applications using OpenACC: (1) Characterize and profile your application; (2) Add compute directives; (3) Add directives to optimize data movement; and (4) Optimize your application using kernel scheduling.

[1]Pedro Mário Cruz e Silva é Bacharel em Matemática (1995) e Mestre em Matemática Aplicada e Otimização (1998) pela UFPE, Doutor em Computação Gráfica pela PUC-Rio (2004). Trabalhou por 15 anos no Instituto Tecgraf/PUC-Rio onde criou o Grupo de Geofísica Computacional, durante este período liderou diversos projetos de Desenvolvimento de Software, bem como projetos de Pesquisa na área de Geofísica. Tem mais de 10 anos de experiência com Métodos Ágeis de desenvolvimento de Software, fez treinamentos de Certified Scrum Master (CSM) e Certified Scrum Product Owner (CSPO). Finalizou o MBA em Gestão Empresarial na FGV-Rio. Atualmente é membro da diretoria da Sociedade Brasileira de Geofísica (SBGf) onde ocupa o cargo de Secretário de Publicações. Atualmente é Arquiteto de Soluções da NVIDIA para o segmento Empresarial para América Latina, com foco especial em Aplicações de Deep Learning para a indústria de Óleo & Gás.

7

Programação Paralela em Memória Compartilhada e Distribuída

Claudio Schepke[1]

Resumo:

Programação paralela é a divisão de uma determinada aplicação em partes, de maneira que essas partes possam ser executadas simultaneamente, por vários elementos de processamento. Os elementos de processamento devem cooperar entre si utilizando primitivas de comunicação e sincronização, realizando a quebra do paradigma de execução seqüencial do fluxo de instruções. Para tanto existem diversas interfaces de programação paralelas, disponíveis através de bibliotecas, extensões de linguagem ou diretivas de compilação. Neste sentido, o objetivo deste minicurso é permitir que o acadêmico conheça algumas das interfaces de programação paralelas existentes e que podem ser utilizadas para as arquiteturas paralelas atuais, utilizar tais interfaces para o desenvolvimento de códigos paralelos, além de avaliar o desempenho de algumas aplicações. Desta forma, estudantes iniciantes tem a oportunidade de aprender e praticar conceitos elementares de programação paralela.

[1]Claudio Schepke é professor adjunto da Universidade Federal do Pampa (UNIPAMPA), campus Alegrete/RS desde 2012. Possui graduação em Ciência da Computação pela Universidade Federal de Santa Maria (2005) e mestrado (2007) e doutorado (2012) em Computação pela Universidade Federal do Rio Grande do Sul, sendo este feito na modalidade sanduíche na Technische Universität Berlin, Alemanha (2010-2011). Claudio tem experiência na área de Ciência da Computação, com ênfase em Processamento Paralelo e Distribuído, atuando principalmente nos seguintes temas: processamento de alto desempenho, aplicações numéricas e programação paralela. Desde a graduação trabalha com aplicações paralelas, utilizando diferentes interfaces de programação paralela. Participa ativamente da ERAD/RS desde 2002, tendo sido recentemente responsável por duas edições do minicurso de Programação Paralela.

Fórum de Iniciação Científica

Alexandre Caríssimi[1],
Odorico Machado Mendizabal[2]

A Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul destaca-se por fomentar a pesquisa e desenvolver novos pesquisadores em áreas relacionadas à computação de alto desempenho e sistemas distribuídos. Neste sentido, o Fórum de Iniciação Científica, presente desde a primeira edição da ERAD/RS, em 2001, abre um espaço para que pesquisadores em formação possam divulgar seus trabalhos.

O Fórum de Iniciação Científica estimula e instiga alunos de graduação a desenvolverem suas aptidões como pesquisadores. Através da revisão de trabalhos e das sessões de apresentação oral, o fórum possibilita a troca de experiências. Além disso, o caráter regional intensifica a integração entre pesquisadores e entusiastas da área provenientes de diversas cidades da região sul do Brasil.

Neste ano foram aceitos 39 trabalhos para apresentação, um número expressivo, que demonstra o interesse pela área e o desejo de jovens pesquisadores em participar dessa comunidade. Dessa forma, a comissão organizadora gostaria de agradecer a todos os autores e orientadores pela dedicação no desenvolvimento dos trabalhos e, também, a preciosa colaboração de todos os revisores em avaliar e sugerir melhorias para contribuir na qualidade final dos trabalhos. Agradecemos ainda a comissão organizadora da ERAD pelo apoio recebido. Somos todos nós, cada um com seu trabalho e dedicação, que fazemos da ERAD o sucesso que ela é. Nosso muito obrigado!

Assim, desejamos a todos um excelente Fórum de Iniciação Científica em mais essa edição da ERAD e que sua participação neste fórum seja bastante proveitosa. Esperamos que este espaço amplie e fortaleça a nossa comunidade de pesquisa em computação de alto desempenho e sistemas distribuídos, e que abra caminhos para pesquisas mais avançadas, através da inserção de pesquisadores em cursos de pós-graduação.

[1]Graduado em Engenharia Elétrica pela UFRGS (1985), Mestre em Ciências da Computação também pela UFRGS (1989) e Doutor em Informática pelo Institut National Polytechnique de Grenoble (1999), França. Atualmente é professor Titular em regime 40/DE da Universidade Federal do Rio Grande do Sul (UFRGS), lotado no Departamento de Informática Aplicada. Integrante do Grupo de Processamento Paralelo e Distribuído (GPPD), que é vinculado ao Programa de Pós-Graduação em Ciência da Computação (PPGC) da UFRGS. Tem experiência na área de Ciência da Computação, com ênfase em Software Básico, atuando principalmente nos temas: redes de computadores, sistemas operacionais, sistemas embarcados e sistemas distribuídos.

[2]Odorico Machado Mendizabal possui doutorado e mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), obtidos em 2016 e 2005, respectivamente, e Engenharia de Computação pela Universidade Federal do Rio Grande (FURG) em 2003. Atualmente é professor Adjunto da FURG, atuando nos temas: sistemas distribuídos, tolerância a falhas e verificação de sistemas.

Análise Da Eficiência Energética de Servidores Utilizando Soluções IoT e Ferramenta de Monitoramento

David Bento, Deoclecio Fusinato, Ademir Camillo J.

[1]Serviço Nacional de Aprendizagem Industrial (SENAI)
{david_bento,deoclecio_fusinato}@estudante.sc.senai.br,
ademir.camillo@edu.sc.senai.br

Resumo. *O consumo de energia e a concentração de equipamentos em Data Centers torna-se cada vez mais comum. Analisar o consumo energético destes equipamentos torna-se uma importante ferramenta para administradores na tomada de decisão sobre como distribuir a carga de trabalho.*

1. Introdução

A tecnologia evoluiu desde os anos 70, permitindo o aumento de processamento com os servidores mainframes, nos anos 90 através da popularização dos computadores pessoais, a internet e a construção de aplicações cliente servidor. As empresas que administram o conteúdo web existente, investem em estruturas complexas para armazenamento de dados e segurança das informações de seus clientes. Os data centers (DC), constituem-se de milhares de servidores interligados, provendo acesso aos mais diversos conteúdos e serviços. Os servidores responsáveis por prover esses serviços não são necessariamente físicos.

Assim, para evitar o desperdício de recursos, a virtualização permite que um mesmo equipamento consiga prover a infraestrutura para as mais diversas necessidades. Além de reduzir o espaço físico utilizado para alocação dos servidores, também reduz a utilização de outros recursos, tais como geradores, equipamentos de refrigeração e consumo de energia.

Grandes DCs recebem a atenção para o consumo de energia elétrica devido a quantidade de servidores ligados constantemente através de UPS, sem interrupção no fornecimento de energia. Segundo [Velte et al. 2012], cerca de 2% da energia mundial é gasta por DCs. Desta forma, este trabalho tem como objetivo monitorar a relação entre a carga de processamento de servidores em um ambiente cloud e o seu consumo de energia a fim de possibilitar uma redução de gastos do consumo elétrico.

2. Trabalhos Relacionados

O crescente uso de recursos computacionais por empresas e pessoas fez com que surgisse a necessidade de grandes aglomerados de equipamentos para prover tais recursos, nomeados de DC. Segundo [Marin 2011], a maior diferença está na quantidade de equipamentos ativos nos grandes DCs existentes hoje em dia, milhares de empresas, privadas e públicas armazenam e processam seus dados nestes locais. Existem três tipos de DCs: Enterprise, Internet e Collocation. O modelo Enterprise tem como objetivo atender um único cliente ou organização. A Internet provê recursos do tipo IaaS (Infraestructure as a Service), SaaS (Sofware as a Service), PaaS (Plataform as a Service). Collocation entrega a infraestrutura para seu cliente, rede de dados, elétrica, ar condicionado, segurança, porém, os

equipamentos como servidores e periféricos são de propriedade do cliente. Entretanto, um DC pode possuir os três tipos dentro de uma mesma infraestrutura.

A centralização das informações em grandes DCs motivou as empresas a repensarem a forma de investir em TI, migrando os dados da sua infraestrutura para a nuvem. Com isso elas conseguem reduzir custos operacionais, tais como investimentos em hardware, consumo de energia e licenciamento de software. Os recursos são disponibilizados sobre demanda, então quando há necessidade de mais processamento em determinado servidor é só solicitar o recurso, ou ainda contratar um novo plano com maior capacidade. Não tem necessidade de comprar novas licenças ou gerar maiores despesas.

Segundo [Veras 2015], Cloud Computing é o conjunto de recursos virtuais facilmente utilizáveis e acessíveis, tais como hardware, software, plataformas de desenvolvimento e serviços. É de responsabilidade do provedor de serviços o gerenciamento da estrutura Cloud Computing. Os modelos de serviços oferecidos se dividem em três, SaaS, Paas e IaaS.

3. Projeto de Experimento

Com objetivo de avaliar a eficiência energética de servidores em um ambiente cloud computing, a proposta é realizar testes de stress de CPU e monitorar o consumo de energia. Os testes executados têm como propósito verificar se o consumo de energia dos servidores está atrelado à carga de processamento. Para isso foi utilizado a ferramenta stress-ng que permite alterar a carga de processamento de forma dinâmica. O stress-ng é uma ferramenta de benchmark voltada para ambientes Linux, que possui uma série de características. Como por exemplo, a capacidade de definir a quantidade de núcleos que serão utilizados, a carga de processamento em cada um deles e o tempo de execução de cada teste. Permitindo assim, a realização de testes referentes a carga de processamento de maneira controlada.

Para coletar as informações sobre o uso da CPU durante o tempo de execução, foi utilizada o software [Zabbix 2017], ferramenta de monitoramento open source. Sua estrutura de monitoramento simples, juntamente com a sua capacidade de operar em conjunto com diversas tecnologias fizeram do Zabbix a escolha ideal para o monitoramento do ambiente proposto. Em relação ao consumo de energia, os dados foram coletados através de um medidor de energia elétrica baseado em Arduino, que por sua vez enviou as informações coletadas através do protocolo SNMP para o Zabbix.

4. Protótipo de Medição e Cenário de Testes

Devido as fontes de energia elétrica dos servidores não apresentarem recursos inteligentes que possam ser monitorados diretamente via SNMP ou Agente Zabbix, foi realizada a construção de um protótipo de medidor de energia elétrica com soluções de hardware livre. Esse protótipo usa como base o Arduino Uno R3, Shield Ethernet e sensor de corrente não invasivo 100A SCT-013. O Shield Ethernet possui a capacidade de enviar os dados coletados via rede para a ferramenta Zabbix, que faz o tratamento e armazenamento. O código desenvolvido para o Arduino utiliza a biblioteca Agentuino, permitindo a comunicação e o envio de dados do Arduino para o Zabbix via SNMP.

No cenário de testes, foram utilizados 4 servidores, 1 IBM X3200 M2 e 3 IBM X3200 M3, além de um Switch Cisco 2900, utilizado para prover a interconexão entre

os dispositivos. Em cada cenário a quantidade de servidores utilizados e sua carga de trabalho alterados. No cenário 1, um (1) servidor sem processamento; Cenário 2, um (1) servidor com carga de processamento a 99%; Cenário 3, dois (2) servidores com carga de processamento a 50%; Cenário 4, três (3) servidores com carga de processamento a 33%. Assim, todos os cenários possuem o mesmo poder computacional.

Para se chegar aos valores referentes ao consumo de energia e ao valor pago por hora em cada cenário, foi aplicada a seguinte fórmula: Corrente x Tensão x Tempo em Horas / 1000 = Consumo em kWh. Como base para o cálculo do valor cobrado por hora utilizaremos o valor padrão de R$ 0,56 por kWh (CELESC, 2017).

5. Resultados

Os testes executados no primeiro cenário visam quantificar o consumo de energia e o valor pago para se manter um único servidor ligado em modo ocioso. O consumo é de 0,105 kWh, ou R$0,06/hora. A Figura 1 apresenta a tela de monitoramento do Zabbix, sendo possível verificar a carga de processamento que o um servidor utiliza estando somente ligado e a corrente elétrica consumida.

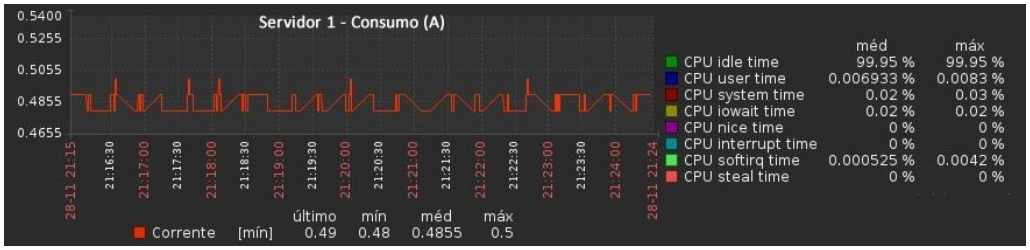


Figura 1. Monitoramento de 1 Servidor/Modo Ocioso

No segundo cenário, o objetivo é coletar os dados de consumo com o servidor operando em sua capacidade máxima. Obtendo assim, 0,185 kWh, ou R$0,10/hora. Esses resultados serão comparados aos demais cenários, verificando-se se é mais eficiente manter um único servidor ligado operando a 99% ou se é mais eficiente dividir a carga entre os demais servidores disponíveis. No terceiro cenário foi realizada a divisão da carga de processamento em dois servidores (50% em cada servidor). O consumo foi de 0,284 kWh, ou R$0,16/hora. Ao realizar a comparação com o Cenário 2, foi possível verificar que um servidor operando a 99% de carga é 54% mais eficiente do que dois servidores operando a 50%. Representando uma economia de R$ 0,06/hora.

Cen?rio	Serrvidores	Carga	Consumo	kWh	Hora	Mensal
Cen?rio 1	1 Servidor	Ociosa	0,48 A	0,105	R$0,06	R$43,20
Cen?rio 2	1 Servidor	99%	0,84 A	0,185	R$0,10	R$72,00
Cen?rio 3	2 Servidores	50%	1,29 A	0,284	R$0,16	R$115,44
Cen?rio 4	3 Servidores	33%	1,65 A	0,363	R$0,20	R$144,00

Tabela 1. Resultado Comparativo

No quarto cenário foi realizada a divisão da carga de processamento em três servidores (33% em cada servidor). O consumo foi de 0,363 kWh, ou R$0,20/hora. Comparando com o Cenário 3, foi possível verificar que um servidor operando a 99% de

carga é 96% mais eficiente do que três servidores operando a 33%, ou uma economia de R$0,10/hora.

Na Tabela 1, observa-se as comparações de consumo e gasto com energia em cada um dos cenários realizados. A Figura 2 apresenta os valores referentes ao consumo de energia dos cenários comparados, o valor gasto em cada situação e a porcentagem de economia em relação ao segundo cenário.

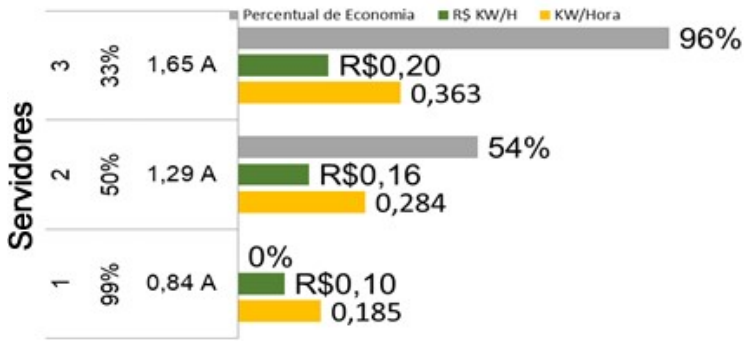


Figura 2. Consumo dos Servidores

Como resultado final, ao analisar todos os cenários e realizar a comparação entre eles foi possível verificar que um único servidor operando em sua capacidade máxima de processamento acaba sendo mais eficiente do que dois ou três servidores com cargas distribuídas. Porém caso a carga de processamento exceda os 100% suportado por um único servidor, seria necessário distribuir essa carga entre os que possuem recursos disponíveis.

6. Considerações Finais

A partir dos resultados, concluímos que deixar um único servidor operando a 99% do seu processamento é mais eficiente do que dividir a sua carga entre vários servidores. Em um cenário de DC poderíamos sugerir que servidores ociosos fossem desligados para oferecer maior economia de energia, pois o consumo de um servidor ocioso, gera sozinho uma despesa diária de R$1,44.

Como trabalhos futuros, a medição pode ocorrer com uma quantidade maior de servidores e gerenciadores de cloud, como o OpenStack atuar como orquestrador no controle da carga dos servidores, migrando máquinas virtuais conforme a carga neles alocadas e desligando os servidores ociosos.

Referências

Marin, p. S. (2011). *Data Centers: Desvendando cada passo: conceito, projeto, infraestrutura física e eficiência energética.* Editora Èrica.

Velte, A. T., Robert, E., and Gabriela, E. M. (2012). *Cloud Computing: Computação em Nuvem uma abordagem pratica*. Alta Books.

Veras, M. (2015). *Computação em Nuvem: Nova Arquitetura de TI*. Brasport.

Zabbix (2017). *Disponível em: ¡https://www.zabbix.com/¿. Acesso em: 02 dez 2017.*

Análise de Bibliotecas JavaScript e Implementação Inicial de um Editor de Processos Quânticos Elementares como Serviço

Gustavo dos Santos, Anderson Ávila, Renata Reiser

[1]Centro de Desenvolvimento Tecnológico
Universidade Federal de Pelotas (UFPEL) – Pelotas – RS- – Brazil

{gfdsantos, abdavila, reiser}@inf.ufpel.edu.br

Resumo. *Neste trabalho é apresentado um panorama sobre o estado atual do editor de processos elementares presente no ambiente VPE-qGM. É apresentado o desenvolvimento de uma solução multiplataforma em orientada a serviço. É feita uma breve discussão sobre as bibliotecas existentes para diagramação online e, é mostrado uma visão sobre uma implementação alternativa do qPE.*

1. Introdução

O acesso a computadores quânticos é limitado pelo alto custo de aquisição e manutenção. Por outro lado, a evolução deste ramo da ciência não pode ser deixado de lado. Faz-se necessário o uso de simuladores quânticos e, neste sentido, este trabalho visa mostrar os primeiros passos na construção de um editor de processos orientado a serviço.

O ambiente VPE-qGM tem como objetivo a modelagem e simulação de circuitos quânticos baseado no modelo qGM [Vieira 2015]. Dos principais componentes do ambiente VPE-qGM (Virtual Programming Environment for the Quantum Geometric Machine), existe o qPE - *quantum Process Editor*. O objetivo do qPE é ser um módulo do VPE-qGM que proporciona a diagramação de circuitos quânticos. Um circuito quântico é exportado para um arquivo formatado do tipo XML. O módulo qPE é capaz de carregar para a memória um circuito quântico salvo em um arquivo XML.

O módulo qPE possui diversas portas quânticas e têm a habilidade de manipular relações entre qubits e as portas quânticas, formando unidades de computação quântica. É escrito em Python 2 e faz uso extensivo da biblioteca wxPython para construções gráficas. Com as atualizações desta biblioteca usada como base, a compatibilidade do módulo qPE com as versões recentes do wxPython foi se perdendo. A biblioteca wxPython proporciona o desenvolvimento de interfaces gráficas a nível de sistema operacional, usa recursos gráficos deste sistema operacional para fazer a montagem da interface de usuário. Por conta destes fatores limitantes, para implementação do editor de processos, é necessária uma implementação usando uma biblioteca para desenvolvimento web.

Este trabalho propõe uma nova implementação do módulo qPE, desta vez como parte de um serviço acessível via internet. Houve um período de análise das bibliotecas e tecnologias web disponíveis atualmente para implementação de uma interface de diagramação online. Após ser decidido a biblioteca a ser usada ao longo do desenvolvimento, foi construído um protótipo. As etapas de análise e desenvolvimento dos protótipos é discutido ao longo deste trabalho, bem como os próximos passos deste projeto.

2. Motivação

O simulador quântico VPE-qGM possui um módulo gráfico para editar processos elementares. Este modulo é escrito em Python 2, e usa a biblioteca wxPython para as construções gráficas. Contém uma interface completa e possibilita a diagramação eficiente de processos quânticos. Visando o crescente uso da computação em nuvem, a motivação deste trabalho é dar os primeiros passos na implementação de um editor de processos online hospedado como um serviço, no qual pode vir a ser uma alternativa ao qPE.

A disponibilização de um software como serviço via internet viabiliza o uso de aplicações independente de plataforma. Foi optado pelo desenvolvimento do editor em tecnologias web, que pode tanto estar disponível em um serviço na internet, como também pode ser empacotado em um executável para várias plataformas. Um exemplo de ferramenta que empacota um aplicativo web em aplicativo executável em sistemas operacionais é o Electron [Electron 2017].

O objetivo com o desenvolvimento de um novo editor de processos é disponibilizalo para o acesso de usuários em geral. Para tal é necessário que a compatibilidade visual seja mantida entre o módulo qPE atual e a implementação web do qPE, ou seja, é necessário uma análise prévia de bibliotecas disponíveis atualmente que permitem o desenvolvimento completo da interface do editor de processos e que esta seja completamente funcional no navegador instalado no sistema operacional hospedeiro do usuário. A compatibilidade de funcionalidades é outro fator limitante para a escolha da biblioteca adequada ao projeto. A biblioteca deve proporcionar características como salvar o estado atual do diagrama de processos elementares, carregar o diagrama de processos a partir de um arquivo formatado corretamente ou então permitir agrupamento de processos sem perda de informações, tal que habilite a possibilidade de desfazer esta operação.

Foram analisadas diversas bibliotecas a fim de descobrir o quanto poderiam suprir os requisitos do projeto. Algumas destas bibliotecas se mostraram ser eficientes em termos de quantidade de código necessário para realizar uma determinada função e em termos das funções que conseguem executar. Uma explicação extensiva sobre todas as bibliotecas analisadas não contempla o objetivo deste trabalho. Das bibliotecas analisadas, foram escolhidas três que combinaram melhor suas funcionalidades com os requisitos necessários para o projeto a serem mencionadas nesta sessão. A seguir é feito um breve comentário sobre cada uma das bibliotecas de destaque.

- **GoJS.** Biblioteca JavaScript de diagramação em aplicativos web. Possui um rico sistema de animações e objetos visuais. Não usa qualquer outra biblioteca de JavaScript e é altamente otimizado [GoJS 2017]. Possui uma vasta lista de exemplos, dos mais variados tipos. Esta biblioteca é licenciada com base em uma ativação de licença paga, o que a torna fora do escopo do projeto.
- **JsPlumb.** Biblioteca JavaScript para manipulação de HTML. Possui uma versão comunitária e um kit de ferramentas que pode ser adquirido com uma licença. A versão comunitária é de código aberto e de livre uso [jsPlumb 2017].
- **MxGraph.** [JGraph 2017] Biblioteca de diagramação web que executa por completo no cliente. É baseado em arquivos do tipo SVG e páginas HTML, não excluindo a possibilidade de uso desta biblioteca com outras tecnologias web para construção de *web apps* dinâmicos, como React.js ou Angular.js. É de código aberto e de livre uso.

A biblioteca mxGraph atendeu a todos os requisitos, no qual mostrou-se ser eficiente ao implementar as primeiras funcionalidades. Foi definido como a biblioteca base para a construção do editor de processos proposto neste trabalho.

3. Metodologia

O desenvolvimento do protótipo se deu usando um *framework* JavaScript chamado mxGraph. Como já discutido neste trabalho, foi buscado uma biblioteca capaz de efetuar desenhos gráficos a partir de polígonos, com capacidade de obter objetos selecionados, agrupamento e capacidades gráficas para construção de interfaces de usuários avançadas em navegadores. MxGraph se mostrou competente e cumpriu com os requisitos nos testes preliminares.

Construir um editor de processos online exige que o navegador do usuário consiga manipular os objetos gráficos sem que o desempenho seja demasiado alterado. MxGraph mostrou que não exige que o navegador do usuário utilize muitos recursos de hardware do sistema operacional hospedeiro.

A construção básica implementada neste trabalho é baseada em um grafo principal, que faz a interligação entre os itens. Cada item é do tipo *Vertex* e possui um estilo, não excluindo a opção da existência de múltiplos itens de mesmo estilo. No protótipo atual existem dois estilos criados, para melhor visualização de diferentes itens.

Das operações que mxGraph habilita o usuário executar sobre o grafo, neste projeto utilizamos apenas operações de adição e remoção. Cada vez que o usuário clica em um item na barra lateral e o solta na área de trabalho, um item é adicionado ao grafo. Operações de agrupamento e desagrupamento também executam apenas as operações previamente mencionadas, onde uma operação de agrupamento remove todos os itens selecionados no grafo e os substitui por um novo item, no qual simboliza um grupo. Um grupo contém uma coleção de itens e suas informações de identificação, como um número de identificação. Um número de identificação de item no grafo é um contador interno do próprio grafo. Cada item adicionado ao grafo causa um incremento deste contador. O contador não decrementa, mesmo após remoção de itens do grafo. Um protótipo do editor pode ser visto na Figura 1.

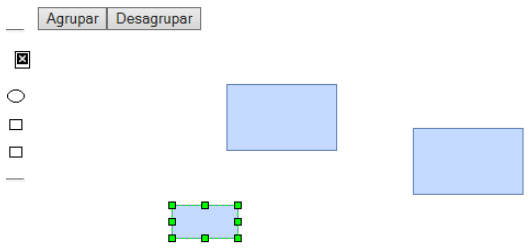


Figura 1. Protótipo do novo editor de processos elementares

O desenvolvimento do editor pode ser acompanhado através do repositório hospedado no GitHub[Santos]. É de código aberto e de livre uso, logo a colaboração da comunidade neste projeto é livre.

Um usuário que terminar o seu trabalho de edição, pode exportar seu diagrama de processos para um arquivo formatado do tipo XML. Este arquivo serve como entrada para o simulador VPE-qGM.

4. Conclusão

O desenvolvimento do editor vem se dando de forma rápida e novas funcionalidades são constantemente adicionadas ao projeto. É necessário visualizar o ciclo de desenvolvimento ao longo do tempo e, os próximos passos do grupo engloba a construção das portas quânticas com base em polígonos, que por sua vez são construídos dinamicamente no navegador do usuário. Busca-se também implementação da funcionalidade de salvar o trabalho corrente do usuário e também restaurar diagramas do usuário. Esta funcionalidade será implementada seguindo o padrão de arquivo que o simulador VPE-qGM toma como entrada. Este componente deve também receber como entrada um arquivo formatado do tipo XML e a partir deste construir o circuito de forma gráfica, mantendo todas as construções do usuário no local onde elas estavam anteriormente.

Busca-se a hospedagem do editor de processos como um serviço na internet. Uma análise das formas disponíveis no mercado para hospedagem de um aplicativo web faz parte dos próximos objetivos buscado pelo grupo.

Referências

Electron (2017). Biblioteca open soruce desenvolvida pelo o github para o desenvolvimento de aplicações desktop multi-plataforma com html, css e javascript.

GoJS (2017). Interactive javascript diagrams in html.

JGraph, L. (2017). mxgraph - an open source javascript diagramming component.

jsPlumb, I. (2017). Build flowcharts, diagrams and connectivity based applications fast.

Santos, G. Implementação web do vpe-qgm qpe.

Vieira, J. (2015). Analise e simulção dos conectivos fuzzy xor e classes derivadas via computação quântica.

Análise de desempenho da utilização de DVFS em Operações de E/S com Dispositivos HDD e SSD

Cleber C. Sartorio[1], **Pablo J. Pavan**[1], **Edson L. Padoin**[1,2]

[1]Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

[2]Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil

{cleber.sartorio,pablo.pavan,padoin}@unijui.edu.br

Resumo. *O objetivo deste trabalho é analisar se a variação da frequência do processador tem impacto no consumo e no desempenho de dispositivos de armazenamento HDD e SSD. Para tanto, testes foram realizados com o Benchmark FIO em operações de E/S. Os resultados demonstraram que os SSD são 37,56 vezes mais eficientes que o HDD em operações de leitura e 14,17 vezes em operações de escrita. No entanto, a variação de frequência do processador não apresentou impacto nas operações.*

1. Introdução

Inúmeras aplicações foram reescritas e passaram a ser executadas em arquiteturas para computação de alto desempenho. Tais aplicações geralmente lidam com grande volumes de dados e demandam de elevados tempos de processamento para alcançar os resultados. Nesse sentido, o consumo de energia tornou-se uma das principais preocupações dada a alta demanda de potência dos atuais sistemas de computação. Assim, encontrar soluções que aumentem a eficiência energética dos sistemas computacionais corresponde a um grande desafio. Almejando contornar esta situação, este trabalho busca analisar se a variação da frequência do processador tem impacto no consumo e no desempenho de dispositivos de armazenamento HDD (Hard Disk Drive) e SSD (Solid State Drive) na computação de alto desempenho.

O objetivo maior deste trabalho é analisar se a aplicação de DVFS (Dynamic Voltage and Frequency Scaling) no processador equivale a uma alternativa para reduzir o consumo de energia, sem reduzir a velocidade de processamento dos sistemas de armazenamento.

2. Trabalhos Relacionados

No passado, pesquisadores exploraram o uso de discos rígidos de múltiplas velocidades para servidores de armazenamento [Carrera et al. 2003, Gurumurthi et al. 2003, Zhu et al. 2005]. As abordagens mais recentes aplicam-se principalmente à escala dinâmica de tensão e frequência - DVFS - para diminuir o consumo de energia pelo processador durante as operações de E/S, pois elas não requerem muito poder de processamento. O DVFS tem sido uma técnica popular adotada para economizar o consumo de energia por núcleos ociosos em sistemas *multi-core* [Lee 2009], recursos inativos em ambientes em nuvem [Hosseinimotlagh et al. 2014, Younge et al. 2010] e durante as fases de comunicação das aplicações MPI [Peraza et al. 2013].

Considerando as diferentes técnicas utilizadas pelos autores, neste trabalho foi explorado a utilização de unidades de armazenamento com arquiteturas distintas, relacionando-as com a utilização da unidade processadora em diferentes frequências pré-fixadas, com o intuito de reduzir o processamento dos nós, e consequentemente economizar energia sem perca de desempenho. Essa alternativa seria complementar as dos trabalhos discutidos avaliando sua viabilidade com relação ao consumo energetico de operacões de E/S.

3. Metodologia

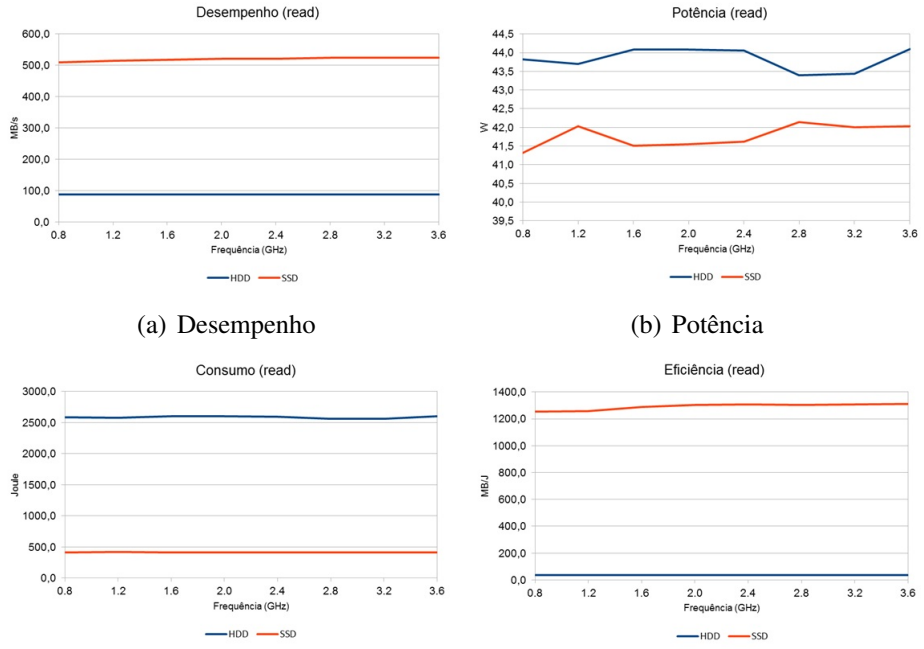
Para a realização dos testes foi selecionado o benchmark FIO, devido ao fato de permitir a simulação de diferentes tipos de cargas de IO e ajuste de vários parâmetros, incluindo a mistura de gravação/leitura [Review 2017]. Este foi configurado para executar requisições de 4 MB, sem o uso da buffer cache para não haver inconsistências nos resultados. Foram executadas 10 repetições para cada operação de escrita e leitura sequencial em um arquivo com tamanho de 5 GB, sendo limitando o tempo máximo de 60 segundos para cada operação, em virtude do numero de testes a serem executados, e pelo período de tempo necessário para concluí-los.

O ambiente de execução é composto por um computador que possui um processador Intel Core i7-4790 com frequência de *clock* mínima de 800 MHz e máxima de $4,0$ GHz[1]. Esta máquina roda em um Sistema Operacional Ubuntu versão 16.04.1 LTS com *kernel* 4.10.0-40-generic, possuindo o sistema de arquivos EXT4. Foram utilizados dois dispositivos de armazenamento. O primeiro, um SSD da marca Samsung de 256 GB, com tensão de 5 V e corrente de $0,50$ A. O segundo, um HDD da marca Western Digitalde 250 GB, velocidade de 7200 RPM, com tensão de 5 V e corrente de $0,65$ A.

Para mensuração da demanda de potência e consumo de energia de todo o sistema foi utilizado um osciloscópio Keysight Agilent Technologies modelo DSO6014A. Afim de se obter a eficiência alcançada, foi realizada a sua mensuração através do cálculo do desempenho em MB/s, dividido pelo custo energético (consumo em Joules).

4. Resultados

Na Figura 1 são apresentados os resultados dos testes com o benchmark FIO em operações de leitura.



(a) Desempenho (b) Potência

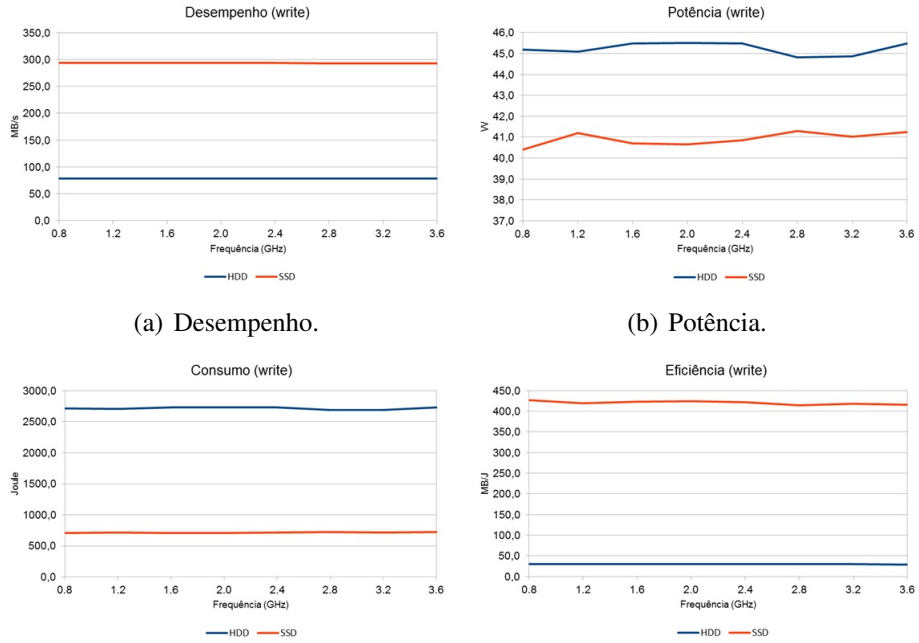
(c) Consumo (d) Eficiência

Figura 1. Resultados dos testes de Leitura.

[1]Para estes trabalho, foi desativada a tecnologia *Intel Turbo Boost*, assim a frequência máxima de operação do processador foi de $3,6$ GHz.

O desempenho mensurado no SSD em operações de leitura foi em média $5,98$ vezes maior que o alcançado com discos HDD. Utilizando HDD, a taxa de transferência foi em média de $86,8$ MB/s, enquanto que com SSD foi $519,3$ MB/s. Nestas operações de leitura, o HDD teve uma demanda média de potência de $43,8$ Watts enquanto qua a demanda do SSD foi de $41,8$ Watts. Assim, o consumo médio em operações de leitura foi de $2585,6$ Joules com HDD, e de apenas $411,9$ Joules com SSD. Ou seja, uma substituição de discos HDD por SSD representa uma redução de até $6,28$ vezes no consumo total de energia.

Na Figura 2 são apresentados os resultados dos testes com o benchmark FIO em operações de escrita.



(a) Desempenho. (b) Potência.

(c) Consumo. (d) Eficiência

Figura 2. Resultados dos testes de Escrita.

Nas operações de escrita, o desempenho mensurado com SSD foi em média $3,73$ vezes maior que o HDD. A taxa de transferência em média alcançada com HDD foi de $78,7$ MB/s, enquanto que com SSD foi $293,3$ MB/s. Nas operações de escrita, o SSD teve um demanda de potência $9,51\%$ menor que o HDD. Enquanto que o HDD demandou em média de $45,2$ Watts, o SSD demandou $40,9$ Watts. Da mesma forma, em operações de escrita o SSD consumiu menos energia. Nos testes com SSD, o consumo foi de $714,3$ Joules, enquanto que com HDD foi $2715,8$ Joules, o que equivale a uma economia de $3,80$ vezes.

Em ambos os testes, leitura e escrita, a variação de frequência do processador não apresentou impacto no desempenho e na demanda de potência.

5. Conclusões e trabalhos futuros

Os processadores e os dispositivos de armazenamento são os maiores responsáveis pelo consumo de energia dos sistemas de HPC. Assim alguns trabalhos têm adotado dispositivos SSD para armazenamento, uma vez que eles possuem desempenho superior aos discos HDD.

Considerando os resultados alcançados neste trabalho, percebe-se que a variação frequência não apresentou impacto significativo nas operacões de escrita e leitura. No entanto, percebe-se que o SSD apresentou melhor desempenho tanto nos testes de leitura quanto escrita.

Analisado a eficiência energética de operações de leitura, os testes com SSD foram $37,56$ vezes mais eficientes que com HDD. Utilizando HDD alcançou-se $34,4$ MB/J enquanto que com SSD foi $1291,1$ MB/J. O mesmo acontece com operações de escrita. SSD foi $14,17$ vezes mais eficiente alcançado $420,5$ MB/J enquanto que com HDD foi de $29,5$ MB/J.

Como trabalhos futuros, pretende-se realizar testes com operações de leitura e escrita aleatória com variação de frequência. Também pretende-se desenvolver pesquisas com outras arquiteturas de processadores, como por exemplo arquiteturas ARM e arquiteturas Intel Xeon Phi, além da possibilidade de utilizar sistema de arquivos diferentes.

Referências

Carrera, E. V., Pinheiro, E., and Bianchini, R. (2003). Conserving disk energy in network servers. In *Proceedings of the 17th annual international conference on Supercomputing*, pages 86–97. ACM.

Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., and Franke, H. (2003). Drpm: dynamic speed control for power management in server class disks. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 169–179. IEEE.

Hosseinimotlagh, S., Khunjush, F., and Hosseinimotlagh, S. (2014). A cooperative two-tier energy-aware scheduling for real-time tasks in computing clouds. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 178–182. IEEE.

Lee, W. Y. (2009). Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors. In *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 216–223. IEEE Computer Society.

Peraza, J., Tiwari, A., Laurenzano, M., Carrington, L., and Snavely, A. (2013). Pmac's green queue: a framework for selecting energy optimal dvfs configurations in large scale mpi applications. *Concurrency and Computation: Practice and Experience*, 28(2):1–20.

Review, S. (2017). Fio - teste de e/s flexível benchmark sintético. `http://www.storagereview.com/fio_flexible_i_o_tester_synthetic_benchmark`.

Younge, A. J., Von Laszewski, G., Wang, L., Lopez-Alarcon, S., and Carithers, W. (2010). Efficient resource management for cloud computing environments. In *Green Computing Conference, 2010 International*, pages 357–364. IEEE.

Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., and Wilkes, J. (2005). Hibernator: helping disk arrays sleep through the winter. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 177–190. ACM.

Análise de Desempenho de *Frameworks* de *Deep Learning*[*]

Rafael G. Trindade[1], **João V. F. Lima**[1]

[1] Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

{rtrindade,jvlima}@inf.ufsm.br

Resumo. *Este trabalho avalia o desempenho de dois frameworks de Deep Learning (Caffe e TensorFlow) em um ambiente de execução heterogêneo. A avaliação se dá ao mensurar os tempos gastos por imagem, através da variação de hiperparâmetros de duas redes profundas conhecidas. O estudo conclui que o framework TensorFlow apresenta uma menor necessidade de memória, e tempos até 49% menores em GPU e até 70% menores em CPU em relação ao Caffe.*

1. Introdução

Deep Learning (DL) apresenta-se como um subconjunto de métodos de Aprendizado de Máquina que particularmente utiliza Redes Neurais Artificais (RNA) profundas em sua composição. Para que uma RNA aprenda a realizar uma tarefa, é necessário que ela seja treinada para tal. Para auxiliar a tarefa de modelagem e treinamento dessas redes, alguns *frameworks* foram criados com o intuito de aproveitar o poder computacional provido por arquiteturas modernas. O uso desses *frameworks* permitem que o projetista não precise se preocupar com otimização computacional em nível de programação, fornecendo um nível de abstração que o permite focar especificamente na modelagem da rede em si.

O uso de um acelerador como uma GPU mostra-se útil para a realização de computações massivas, e para treinamentos de redes profundas isso não é diferente. A aplicação de filtros convolucionais, comuns nesse tipo de rede, pode ser resumida computacionalmente a multiplicações de matrizes, e, como tal, pode ser realizada de forma eficiente em arquiteturas paralelas. Dadas as diferentes configurações dessas arquiteturas, é cabível mensurar quão rápido os *frameworks* conseguem deixar o treinamento dessas redes. Este trabalho tem como objetivo avaliar o desempenho, através do tempo consumido por imagem durante tarefas de treinamento, de dois *frameworks* de DL em um ambiente de execução que possui a disposição uma GPU de última geração. Ele está organizado da seguinte maneira: a Seção 2 exibe trabalhos relacionados ao assunto; a Seção 3 introduz os dois *frameworks* avaliados neste trabalho; a seção 4 aborda a metodologia utilizada nessa avaliação; a seção 5 exibe o resultado das execuções dos treinamentos, e finalmente, a seção 6 apresenta considerações finais obtidas através deste trabalho.

2. Trabalhos Relacionados

Apesar da crescente popularidade da área, poucos estudos foram conduzidos com o intuito de investigar o desempenho de *frameworks* de DL. Shams et al. [Shams et al. 2017] avaliam o desempenho de três *frameworks* de DL – Caffe, TensorFlow e Apache SINGA, em diferentes configurações de *hardware*, em ambientes com múltiplos nodos computacionais e diferentes CPUs e GPUs, além do uso da tecnologia NVLink e do processador

[*]Trabalho desenvolvido recebendo fomento do Edital Nº 015/2017 FIPE/UFSM.

Intel Xeon Phi. O trabalho fizera uso de uma versão mais antiga do TensorFlow (versão 0.12). Como resultado, Shams et al. [Shams et al. 2017] concluem que para as arquiteturas testadas a biblioteca Caffe apresenta menores tempos de computação e melhor escalabilidade que seus concorrentes.

3. Frameworks

Este trabalho aborda dois *frameworks* de DL bem difundidos pelas comunidades de pesquisa:

- **Caffe**: Desenvolvida pela *Berkeley Vision and Learning Center*, é uma biblioteca implementada em C++ que provê um *framework* simples e customizável para DL. Provê ligações para outras linguagens, como Python e MATLAB, simplificando o treinamento e inferência de redes neurais convolucionais de propósito geral e outros modelos profundos em arquiteturas comuns [Jia et al. 2014].
- **TensorFlow**: Desenvolvida pela Google, TensorFlow permite que computações definidas por grafos de fluxo possam ser executadas com poucas modificações em uma ampla variedade de sistemas heterogêneos. O sistema é flexível e pode ser usado para expressar uma grande variedade de algoritmos, como de treinamento e inferência para modelos de DL [Abadi et al. 2016]. Também é implementado em C++ e possui Python como ligação oficialmente suportada.

4. Metodologia

Para a execução dos treinamentos que passariam pela avaliação proposta pelo trabalho, foram escolhidos dois modelos de redes profundas, bem difundidas no campo de pesquisa de DL, para a etapa de avaliação dos *frameworks*: AlexNet e GoogLeNet. Ambas as redes possuem implementações de seus modelos criados pelas equipes de desenvolvimento de ambos os *frameworks*[1][2], permitindo que possam ser treinadas e avaliadas por quem deseja estudar suas arquiteturas. Para cada conjunto de configuração (rede, *framework*, modo e tamanho de lote) fora realizada um treinamento, tendo os valores utilizados para avaliação (segundos por imagem) sido calculados como média de cada treinamento. Os seguintes hiperparâmetros das redes tiveram valores especificamente adotados para os treinamentos:

- **Tamanho de lote**: Os treinamentos foram executados com variações no tamanho dos lotes de imagens: variando, em potências de 2, de 1 a 512;
- **Épocas**: Uma época corresponde a um intervalo de iterações em que todas as imagens do conjunto de entrada tenham passado uma vez pela rede. Este trabalho adotou o uso de 10 épocas de treinamento;
- **Unidade de Processamento (Modo)**: Hiperparâmetro responsável por determinar em qual unidade de processamento a rede será treinada (CPU ou GPU), e em qual memória os dados da rede serão armazenados (RAM ou dedicada/GPU).

5. Resultados Experimentais

Os treinamentos realizados em ambos os *frameworks* foram conduzidos no seguinte ambiente de execução, denominado **lsc5**:

[1]https://github.com/BVLC/caffe/tree/master/models

[2]https://github.com/tensorflow/models/tree/master/research/slim/nets

- CPU Intel Xeon E5620, com 8 núcleos operando a 2.4 GHz;
- GPU NVIDIA GeForce GTX Titan X, com 3072 núcleos CUDA e 12 GB de memória GDDR5 dedicada;
- 11 GB de memória RAM DDR3 e 14 GB de memória *swap*;
- Caffe versão 1.0.0 e TensorFlow versão 1.4.0.

A Figura 1 exibe os resultados das execuções dos treinamentos no ambiente supracitado. Com ele, se é possível visualizar a superioridade em poder computacional da GPU com relação a CPU. Algumas constatações podem ser realizadas:

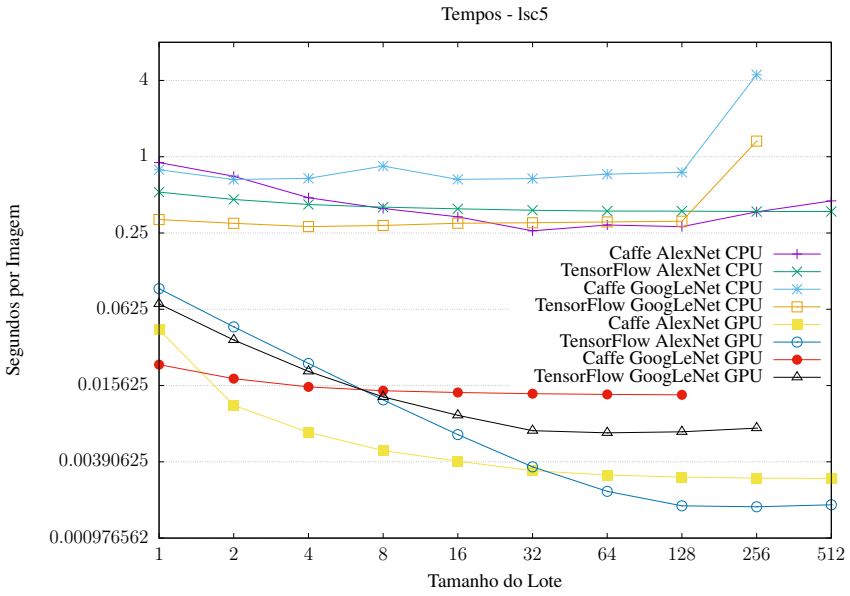


Figura 1. Estatísticas de tempo de execução no ambiente lsc5, com tempos por imagem para todas as combinações de redes, *frameworks* e unidades de processamento.

Para praticamente todos as combinações de rede e unidade de processamento o TensorFlow apresentou tempos menores em maior parte dos tamanhos de lote. Algumas exceções podem ser observadas, como os tempos relativamente semelhantes para a rede AlexNet rodando em CPU. Nas execuções da rede GoogLeNet em CPU, para ambas os *frameworks*, é possível visualizar um aumento no tempo por imagem com um lote de tamanho 256, pois, houve a necessidade dos *frameworks* recorrerem à memória *swap* para armazenar os dados internos da rede, quadruplicando o tempo médio por imagem.

Em suma, TensorFlow necessita de menos recursos de memória para armazenar os dados necessários para o treinamento. Esse fato fica parcialmente visível no gráfico, onde o *framework* consegue realizar o treinamento com um lote de tamanho 256, enquanto Caffe chegou ao limite com um lote de tamanho 128. Entretanto, isso nada se refere à quantia de alocação de memória realizada pelo *framework*: TensorFlow aloca por padrão o máximo de memória possível em GPU, mesmo sem necessitar de todo o espaço para o armazenamento dos dados intermediários das redes, e independentemente se a execução se dá em CPU ou GPU.

A Tabela 1, que exibe o número de chamadas e o tempo total gasto pelas instruções de multiplicações de matrizes em GPU, sugere que o tempo por imagem reduzido utili-

zado pelo TensorFlow se deve a um menor uso de multiplicações de matrizes (SGEMM), e à computação de algumas em lote (*Batched SGEMM*), habilidade fornecida pela biblioteca de algebra linear para GPUs NVIDIA, cuBLAS, e que confere a habilidade de realizar múltiplas multiplicações de matrizes pequenas de uma vez só, enquanto que o Caffe somente faz uso de instruções SGEMM simples.

Tabela 1. Tempos de execução e chamadas dos métodos SGEMM utilizados.

Método SGEMM	Caffe		TensorFlow			
	Comum		Comum		Batched	
	Tempo (s)	Chamadas	Tempo (s)	Chamadas	Tempo (s)	Chamadas
128x64_nn	6,90060	75264	0,016219	21	0,14928	26
128x64_nt	4,02506	71680			0,089606	21
128x64_tn	5,02266	71701				
128x128_tn	1,22427	10282	0,18607	60		
128x128_nn	1,00165	10812	0,16892	42	0,31289	45
128x128_nt	0,20359	60	0,16529	60	0,30616	42
64x64_nt					0,39549	64
64x64_nn					0,11230	21

6. Considerações Finais

Este trabalhou avaliou o desempenho de dois *frameworks* de DL em uma arquitetura híbrida dispondo de uma GPU moderna. Dois modelos de rede foram selecionados para execução dos treinamentos, e variações de hiperparâmetros foram realizados de treinamento para treinamento. Ao final das execuções, é possível avaliar uma clara vantagem em tempo computacional do *framework* TensorFlow, tanto em CPU quanto em GPU – atingindo tempos até duas vezes menores em GPU –, além de necessitar menos memória que o *framework* Caffe para o armazenamento dos dados da rede. Trabalhos futuros podem abordar diferentes arquiteturas e tamanhos maiores de lote, além de modelos de redes diferentes.

Agradecimentos

Este trabalho foi financiado pelo Fundo de Incentivo à Pesquisa – FIPE / UFSM e pela Fundação de Amparo à pesquisa do Estado do RS (FAPERGS). Agradecimentos especiais ao programa NVIDIA Hardware Grant Program que cedeu a GPU utilizada nos testes realizados nesse trabalho.

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., and Corrado, G. S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R. B., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093.

Shams, S., Platania, R., Lee, K., and Park, S. J. (2017). Evaluation of deep learning frameworks over different hpc architectures. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1389–1396.

Análise de Desempenho do Software Incompact3D em uma Arquitetura com Múltiplos Núcleos

Lucas R. de Araujo, Cristian M. Weber
Fernando E. Puntel, Andrea S. Charão, João Vicente F. Lima

[1] Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria

Resumo. *Este trabalho utiliza o software Incompact3D, situado na área de dinâmica de fluidos, para a realização de testes de desempenho em arquiteturas com múltiplos núcleos. Estas arquiteturas surgem como alternativa para a baixa disponibilidade de grandes clusters, que costumam suprir a demanda de processamento de softwares como o Incompact3D. Os resultados indicam que as arquiteturas utilizadas são capazes de executar diversos casos do Incompact3D.*

1. Introdução

Os problemas presentes na área de dinâmica dos fluidos são indiscutivelmente de grande complexidade, sendo que a resolução dos mesmos é de extrema importância para a comunidade científica. O estudo de [Laizet et al. 2010] indica que avanços nesta área de pesquisa foram conquistados expressivamente nos últimos 40 anos, devido ao advento dos computadores e avanços nos métodos numéricos. Nesse contexto, o Incompact3D é um *software* que realiza simulações com o objetivo de resolver problemas na área. Devido à complexidade dos cálculos e ao alto nível de paralelização do Incompact3D, ele caracteriza-se como um *software* típico de computação científica de alto desempenho.

O código do Incompact3D foi feito inicialmente para processadores seriais, posteriormente convertido para processadores vetoriais até chegar no modelo atual, que foi feito para processadores em paralelo [Laizet et al. 2010]. Para tal modelagem ocorre a utilização do MPI (Message Passing Interface), que é a linguagem padrão para programação paralela em sistemas distribuídos e apresenta importantes vantagens de utilização como o desempenho atingido e a sua portabilidade [Jin et al. 2011].

A necessidade de grandes *clusters* para execução de um algoritmo como o Incompact3D esbarra no problema de disponibilidade dessas máquinas para fins de pesquisa. Os trâmites administrativos e as filas de espera para utilização de grandes *clusters* acabam resultando na utilização de *desktops* para a realização de estudos prévios por parte dos pesquisadores. Todavia, tal prática está longe do ideal para pesquisa, pois as execuções se tornam extremamente demoradas. Arquiteturas paralelas com múltiplos núcleos surgem como uma alternativa de meio-termo, tanto por fatores de desempenho quanto por fatores econômicos, para a resolução de problemas na área de dinâmica dos fluidos e na área de computação de alto desempenho.

Nesse contexto, este trabalho busca conhecer o comportamento do Incompact3D em arquiteturas paralelas de múltiplos núcleos. Esse comportamento foi investigado através da análise dos dados gerados pelo Incompact3D e por ferramentas auxiliares durante os testes realizados.

2. Incompact3D

O Incompact3D é um *software* que realiza simulações envolvendo turbulência e fluidos, dada uma malha no plano cartesiano 3D. Diversas variáveis definem a complexidade e o comportamento da simulação, entre elas: o tamanho da malha no plano cartesiano, a condição de contorno, o número de colunas e linhas em que ocorrerá a decomposição da malha, o número de iterações abrangido pela simulação, entre outras.

Como apresentação geral do código, [Laizet e Li 2011] definem que o Incompact3D resolve as equações incompressíveis de Navier Stokes usando *compact schemes* de sexta ordem para a discretização espacial. O *software* divide a malha cartesiana em um número de partes definido pelo usuário, tal que esse número seja igual ao número de processos utilizados durante a execução. A partir daí existem rotinas de inicialização, rotinas que fazem parte do ciclo de iterações e rotinas de finalização da aplicação.

É no ciclo de iterações que acontecem as simulações, em função do tempo decorrido. É possível identificar, através do código e de trabalho anteriores, uma sequência de estados pela qual a malha passa através das operações de transposição, onde esses estados representam a orientação em que a malha decomposta se encontra. Há também a realização de várias operações matemáticas durante esses diversos estados, como, por exemplo, a resolução da equação de Poisson para pressão. No decorrer da execução do Incompact3D, a saída é gerada em tempo real para o usuário e pode também ser transcrita para arquivos.

3. Materiais e Métodos

Os experimentos foram realizados em duas máquinas diferentes acessadas através de uma conexão SSH (Secure Shell), realizada em *desktops*. As máquinas utilizadas são uma SGI Altix XE 210, que apresenta arquitetura com 8 núcleos (processador Intel® Xeon® CPU E5620 @ 2.40GHz) e 16GB de memória RAM e uma NUMA SGI UV2000 de arquitetura com 48 núcleos (processador Intel® Xeon® CPU E5-4617 @ 2.90GHz) e 512GB de memória RAM. As duas máquinas são apresentadas neste trabalho, respectivamente, como LSC5 E BLADE01. Ambas as máquinas utilizam o sistema operacional GNU/Linux (distribuição Debian) e a implementação do MPI utilizada foi a Open MPI, na versão 2.0.2. A ferramenta EZTrace [Trahay et al. 2011], que é configurada para executar junto ao Incompact3D, foi utilizada para obter dados do uso de funções do MPI, referentes a trocas de mensagens entre os processos, durante a execução do Incompact3D.

Dados os ambientes de execução do Incompact3D, o *software* foi configurado de diversas formas. As principais mudanças ocorreram no tamanho da malha do problema, que é alterada a partir de mudanças em quaisquer das três malhas do plano cartesiano 3D (X, Y e Z). O número de processos utilizados durantes as execuções também foi alterado, o que reflete na decomposição da malha. O número de iterações realizadas durante a execução também foi configurado de diferentes formas, o que reflete no tempo consumido pela simulação. O número de iterações utilizado na maioria dos casos foi o de 100 iterações, exceto em casos que buscavam explorar o comportamento dessa variável em diferentes situações.

4. Resultados e Discussão

A abordagem inicial realizada se baseou em testes feitos sem o auxílio de qualquer ferramenta, onde o Incompact3D foi configurado para execução e o próprio apresentou o

tempo total gasto nas iterações, ao final das execuções. A Figura 1 ilustra os resultados desses testes nas máquinas LSC5 e BLADE01 para uma configuração de malha com 4210688 (128x257x128) nós e uma variação entre 2 e 48 processos utilizados. O gráfico indica que para um mesmo número de processos, ainda assim há divergência no tempo de execução.

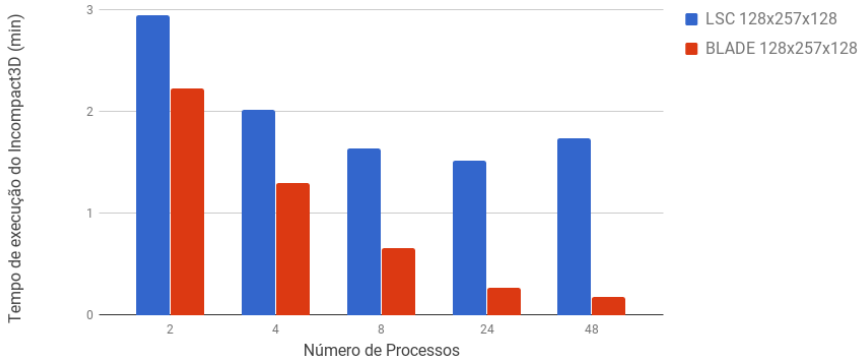


Figura 1. Tempo total de cada execução em função do número de processos.

Esta divergência é explicada pela diferença de arquitetura entre as máquinas LSC5 e BLADE01, principalmente devido a diferença de frequência entre os processadores de cada máquina. Nos casos de 2, 4 e 8 processos as duas máquinas operam com todos os processos simultaneamente, porém a máquina LSC5 executa o Incompact3D mais lentamente devido a menor frequência de seu processador. Já nos casos de 24 e 48 processos a diferença é ainda maior e a explicação é simples, dado que enquanto a BLADE01 consegue utilizar os processos criados simultaneamente, a máquina LSC5 não tem essa capacidade. Logo, o revezamento entre os processos ocasiona um tempo de espera para a execução de outros na máquina LSC5.

A outra abordagem de testes visou observar a interação entre os processos durante a execução do Incompact3D, que ocorre através de funções do MPI. A interação mais observada foi a MPI_ALLTOALLV (onde todos os processos mandam dados para todos os outros e recebem dados de todos os outros), que é bastante utilizada durante as operações de transposição. Nessas operações, ocorre uma troca massiva de dados, a fim de alterar o estado atual de decomposição da malha do Incompact3D.

Através da ferramenta EZTrace foi possível monitorar a porcentagem de tempo ocupada por essas operações. Esses dados são apresentados na Figura 2 e mostram que, apesar de diminuir o tempo total de execução, o aumento no número de processos resulta em um aumento na proporção de tempo ocupado por operações de transposição. Isso se deve ao fato de que mais processos precisam se comunicar entre si, gerando um aumento na duração das operações de transposição. O alto crescimento nos casos de 24 e 48 processos na máquina LSC5 se devem ao tempo de espera pelo qual os processos passam enquanto não estão sendo executados.

Outra variável observada durante os testes foi a de número de iterações. Dessa forma, alguns testes foram conduzidos a fim de verificar qual a curva de crescimento de tempo em relação a curva de aumento do número de iterações. Para experimentos com 200 iterações, há um aumento de tempo em torno de 0.08 segundos por iteração, o que passa para mais de 0.1 segundo para o maior caso testado (600 iterações). A variação de tempo

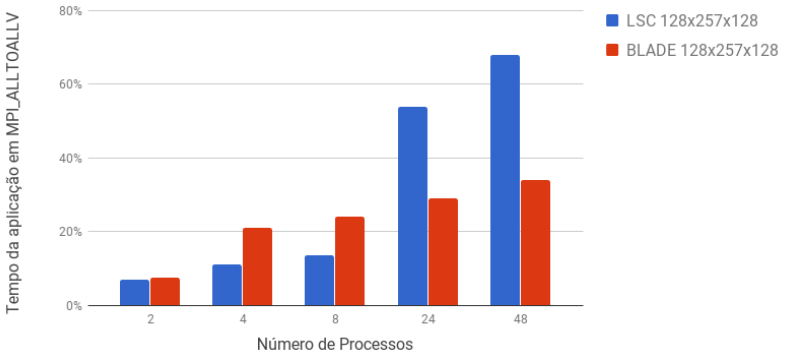


Figura 2. Porcentagem de tempo da comunicação entre os processos em relação ao tempo total.

apresentada é em relação a uma curva de crescimento linear do tempo em relação a curva do número de iterações. Há então uma tendência de aumento para casos maiores, devido a variação observada nesse intervalo. O que deixa claro que não há um comportamento constante por parte das iterações, ainda que a complexidade do problema seja a mesma.

5. Considerações Finais

A partir dos resultados obtidos nos testes é possível afirmar que as máquinas LSC5 e BLADE01 foram capazes de executar com sucesso o Incompact3D nos casos propostos. Também observou-se que as operações de transposição não se beneficiam do aumento do número de processos, ocupando cerca de 34% da execução com 48 processos na BLADE01 contra cerca de 7% no caso de 2 processos. Por outro lado, devido a diminuição de tempo mostrada na figura 1 é possível afirmar que outras operações se beneficiam desse aumento. Em trabalhos futuros, espera-se investigar mais profundamente o comportamento de cada iteração do Incompact3D e maneiras de melhorar o desempenho das operações de transposição.

6. Agradecimentos

Os autores agradecem ao financiamento e recursos cedidos pela FAPERGS-CNPQ PRONEX, pois foram imprescindíveis para a realização da pesquisa e do presente trabalho.

Referências

Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., e Chapman, B. (2011). High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562 – 575.

Laizet, S., Lamballais, E., e Vassilicos, J. (2010). A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution dns of fractal generated turbulence. *Computers & Fluids*, 39(3):471 – 484.

Laizet, S. e Li, N. (2011). Incompact3d: A powerful tool to tackle turbulence problems with up to O(105) computational cores. *International Journal for Numerical Methods in Fluids*, 67:1735–1757.

Trahay, F., Rue, F., Faverge, M., Ishikawa, Y., Namyst, R., e Dongarra, J. (2011). EZTrace: a generic framework for performance analysis. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Newport Beach, CA, United States.

Análise de Desempenho na Transmissão de Dados Criptografados Utilizando o Protocolo OSGP

Iago S. Ochôa, Douglas A. Santos, Valderi R. Q. Leithardt

Laboratory of Embedded and Distributed Systems – LEDS
Universidade do Vale do Itajaí (UNIVAL)
Caixa Postal 360 – 88.302-202 – Itajaí – SC – Brasil

{iago.ochoa, douglasas}@edu.univali.br, valderi@univali.br

Abstract. *This paper describes an encrypted data transmission simulation model for performance analysis. The model was developed in MATLAB and consists of using symmetric and asymmetric key cryptography to encode the data to be transmitted. The technique used to perform data transmission was the PLC technique. At the end of the simulations it was possible to evaluate the execution time for the tests made.*

Resumo. *Este artigo descreve um modelo de simulação de transmissão de dados criptografados para análise de desempenho. O modelo foi desenvolvido em MATLAB e consiste em usar criptografias de chave simétrica e assimétrica para codificar os dados a serem transmitidos. A técnica utilizada para realizar a transmissão de dados foi a técnica de PLC. Ao fim das simulações foi possível avaliar o tempo de execução para os testes realizados.*

1. Introdução

É importante enfatizar que as redes elétricas estão passando por sua maior transformação desde o seu surgimento. A rede atual está sendo substituída por um conjunto de sistemas digitais denominados *smart grid* [Falcão 2010]. Esses sistemas são mais eficientes, confiáveis e sustentáveis. Um exemplo de aplicação é a medição automatizada da energia elétrica. Com o advento das redes *smart grid* foi possível substituir o ser humano por um sistema de medição automatizado que pode receber e enviar informações às concessionárias responsáveis pelo fornecimento de energia.

Atualmente, várias organizações fabricam microcontroladores com aplicações específicas para o segmento *smart grid*. Essas aplicações devem estar de acordo com as normas que regem a segurança desse segmento. Com o aumento do poder de processamento ao longo dos anos, percebeu-se que um tipo mais seguro de criptografia poderia ser usado nesses sistemas.

Para o cenário de simulação, um modelo de transmissão de dados ponto-a-ponto foi desenvolvido usando a técnica de comunicação PLC com a modulação GMSK (*Gaussian Minimum Shift Key*) para enviar os dados pela linha de energia elétrica. A criptografia de chave assimétrica escolhida foi o RSA sendo amplamente aplicada nos dias atuais [Gomez 2012]. A criptografia simétrica utilizada neste trabalho foi o AES 128 bits, recomendado pelas normas. O sistema consiste na avaliação de desempenho da transmissão de dados criptografados.

Este artigo está estruturado da seguinte forma: a seção 2 apresenta os trabalhos correlatos. A seção 3 descreve o modelo de simulação proposto. A seção 4 apresenta os resultados obtidos com os testes. Por fim, a seção 5 apresenta as conclusões obtidas e os trabalhos futuros.

2. Trabalhos Correlatos

[Uludag et al. 2016] apresenta um protocolo seguro de comunicação de dados que usa criptografia RSA para transmissão de dados que são coletados em dispositivos de medição de energia. Sua conclusão aponta para a criação e uso de um protocolo próprio, pois os protocolos existentes não podem abordar todos os dispositivos que estão emergindo.

[Shijo and Sankaranarayanan 2017], desenvolveram uma análise de desempenho de protocolos de segurança de redes *smart grid.* Foi usado o simulador NS-2 para criar uma rede de medidores inteligentes e realizar os testes. Eles concluem que AES e ECC são válidos para uso. Sua conclusão enfatiza que é necessário transmitir os dados do medidor inteligente através de um *gateway* seguro para fornecer segurança resistente a ataques de DDoS (*Distributed Denial of Service*). Em [Abdallah and Shen 2017] é discutido o uso de *clusters* de dados em vez de relatórios individuais de clientes. Seu esquema provou garantir a comunicação leve e, consequentemente, um bom desempenho computacional.

Fundamentado na literatura pesquisada, este trabalho propõe um modelo de transmissão que usa o protocolo OSPG (*Open Smart Grid Protocol*), que é o mais usado no mundo [OSGP Alliance 2017]. As criptografias utilizadas serão, para chave assimétrica o RSA e para chave simétrica o AES. O AES foi escolhido devido à OSGP padronizar o uso da criptografia RC4, o que provou ser inseguro [Jovanovic and Neves 2015]. O RSA foi escolhido porque a maioria dos microcontroladores voltados para o segmento *smart grid* suporta apenas criptografia de chave simétrica devido ao seu poder de processamento. A modulação usada para transmissão dos dados é a GMSK, que foi escolhida devido às vantagens contra o ruído nas linhas de energia [Santos 2008]. A Tabela 1 apresenta a comparação entre os trabalhos relacionados e este trabalho.

Tabela 1. Comparação dos trabalhos correlatos

Autor	**Características**		
	Protocolo	**Tipo de criptografia**	**Algoritmo**
[Uludag et al. 2016]	Próprio	Assimétrico	RSA
[Shijo and Sankaranarayanan 2017]	Nenhum	Simétrico	AES
[Abdallah and Shen 2017]	Nenhum	Assimétrico	NTRU
Modelo Proposto	OSGP	Assimétrico/Simétrico	RSA/AES

3. Modelo Proposto

O modelo foi implementado em MATLAB devido ao fato de ele possuir blocos de funções prontas para modulação de sinal no domínio da frequência. Dois códigos foram criados separadamente, o primeiro consiste em modular e transmitir o sinal, o segundo é responsável pela criptografia dos dados. Depois de validar os dois códigos, ambos foram integrados em um só para criar o sistema. A Figura 1 ilustra os principais passos do sistema proposto. Primeiramente o dado é criptografado utilizando o modo ECB de cifra de

blocos. Após isso o dado é modulado usando a modulação GMSK, para se assemelhar a um cenário real foi aplicado um ruído branco sobre a rede de transmissão. Após o término da aplicação do ruído é feita a demodulação e decriptação do dado transmitido. O tempo de medição é medido para todo o processo.

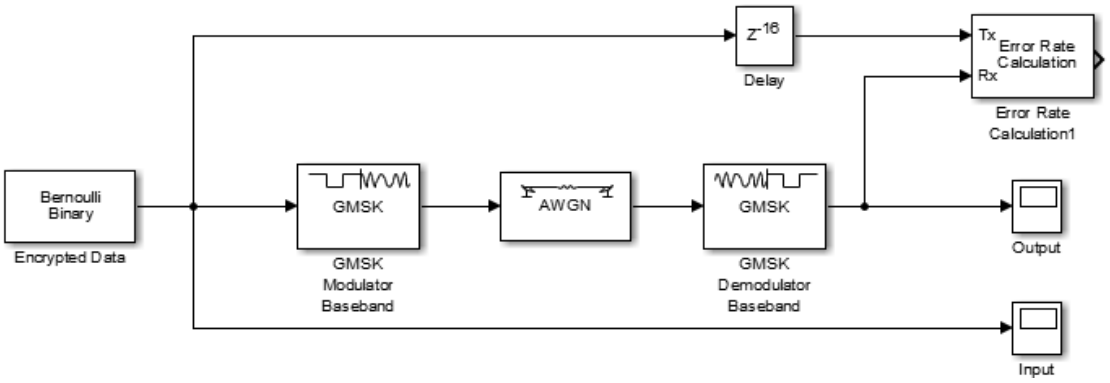


Figura 1. Diagrama de Blocos do algoritmo principal no SIMULINK.

4. Testes e Resultados Preliminares

Quanto ao tempo de execução, o algoritmo AES apresentou melhor desempenho. Mesmo usando uma chave pequena para o algoritmo RSA, é possível perceber que o algoritmo é mais lento do que o AES. A Tabela 2 apresenta a comparação entre os dois códigos executados no MATLAB. Cada código transmitiu os dados cem vezes, com isso foi obtido o tempo de execução mostrado na Tabela 2. O tempo de execução mostrado na Tabela 2 é o tempo total para as cem transmissões, no AES com tamanho de 6 bytes e no RSA com tamanho de 3 bytes.

Tabela 2. Resultados obtidos

Algoritmo	**Modo ECB**	**Tempo de execução**
AES 128 bits	16 bits	8,8521 segundos
RSA	8 bits	108,2148 segundos

Os testes foram realizados em um notebook Acer Aspire E1-572-6638 com memória de 8 GB DDR3 1333 MHz. A placa mãe deste modelo é uma Hannstar V5WE2 LA-9532P com processador Intel Core i5 4200U com frequência de operação de 1.6 GHz até 2.3 GHz. O sistema operacional usado na simulação foi o Windows 10 Pro 64 bits com o software MATLAB R2016a.

5. Conclusões e Trabalhos Futuros

O algoritmo AES foi adaptado de uma biblioteca em C disponibilizada pela *Texas Instruments* para os microcontroladores voltados para o segmento *smart grid*. A escolha de usar esse algoritmo específico foi devido ao fato de ser otimizado para rodar de maneira otimizada em microcontroladores voltados para este segmento. A adaptação do algoritmo em C ao MATLAB funcionou corretamente, mas algumas mudanças foram desenvolvidas para funcionar igual, o que pode ter afetado o desempenho.

A criptografia RSA utilizada tem uma chave pequena devido ao poder de processamento limitado dos microcontroladores voltados para o segmento *smart grid*. O uso de chaves extensas tornou as simulações impraticáveis devido ao tempo de encriptação e decriptação de dados. É importante ressaltar que o algoritmo RSA não está otimizado para microcontroladores. Para tanto, foi utilizado um algoritmo geral para realizar a simulação.

Com o desenvolvimento deste trabalho foi possível identificar o tempo de execução de cada algoritmo de criptografia para uma rede PLC simulada. Conforme os valores obtidos referente ao tempo de execução de cada um dos algoritmos, foi possível identificar que o algoritmo de chave simétrica provou ser mais eficiente que o algoritmo de chave assimétrica nesta simulação.

Para trabalhos futuros pretende-se evoluir o modelo de transmissão ponto-a-ponto para uma NAN e uma WAN, para assim, realizar testes de desempenho de criptografia nesses tipos de redes que se assemelham a um cenário prático.

6. Agradecimentos

Os resultados deste trabalho foram possíveis devido ao apoio recebido através do projeto de pesquisa e extensão da Universidade do Vale do Itajaí através do artigo 170 - pesquisa de projetos. Também agradecemos o uso da infra-estrutura fornecida pelo Laboratório de Sistemas Integrados e Distribuídos (LEDS) da UNIVALI.

Referências

Abdallah, A. and Shen, X. (2017). Lightweight security and privacy preserving scheme for smart grid customer-side networks. *IEEE Transactions on Smart Grid*, 8(3):1064–1074.

Falcão, D. M. (2010). Integração de tecnologias para viabilização da smart grid.

Gomez, S. (2012). Implementation of rsa algorithm. Disponível em: http://www.mathworks.com/matlabcentral/fileexchange/38439-implementation-of-rsa-algorithm. Acesso em 17/11/2017.

Jovanovic, P. and Neves, S. (2015). Dumb crypto in smart grids: Practical cryptanalysis of the open smart grid protocol. *IACR Cryptology ePrint Archive*, 2015:428.

OSGP Alliance (2017). Open smart grid protocol. Disponível em: http://www.osgp.org/en/technical. Acesso em 17/11/2017.

Santos, T. L. (2008). Power line communications.

Shijo, M. and Sankaranarayanan, S. (2017). Performance analysis of security protocols in smart energy meter system. 12(19):8294–8315.

Uludag, S., Lui, K. S., Ren, W., and Nahrstedt, K. (2016). Secure and scalable data collection with time minimization in the smart grid. *IEEE Transactions on Smart Grid*, 7(1):43–54.

Análise de segurança em infraestruturas de rede de provedores de nuvens computacionais OpenStack

Nicolas Peter Lane, Charles Christian Miers

[1] Departamento de Ciência da Computação (DCC)
Centro de Ciências Tecnológicas (CCT) – Universidade do Estado de Santa Catarina (UDESC)

nicolas@colmeia.udesc.br, charles.miers@udesc.br

Resumo. *O presente trabalho propõe uma solução de segurança intitulada Honeypot as a Nova Node (Haa2N). A solução viabiliza a análise de infraestruturas de rede de provedores em nuvens computacionais OpenStack tradicionais e de alto desempenho. É apresentada a sua arquitetura genérica, aplicabilidade e um exemplo de caso de uso em nuvens computacionais OpenStack Newton, que atualmente encontra-se sob experimentação.*

1. Introdução

A computação em nuvem é uma forma de computação bem estabelecida, que é impactada pelo exercício de diversas áreas, dentre elas destaca-se a segurança. Por sua vez, a segurança em nuvens computacionais abrange a infraestrutura de nuvem do cliente e do provedor [5, 7]. A infraestrutura de um cliente corresponde aos recursos computacionais contratados de um provedor de nuvem computacional (*e.g.*, OpenStack, CloudStack, OpenNebula, Amazon AWS, *etc.*). Enquanto a infraestrutura do provedor corresponde aos demais recursos de computação que não estão presentes na infraestrutura do cliente (*e.g.,* domínio público, domínio de controle e domínio de convidados). O que é crítico, uma vez que a forma como o serviço é ofertado ao cliente, implica sob a responsabilidade com segurança atribuída às partes [1, 2]. Responsável por tornar ora o cliente, ou o provedor, o ator que desempenha função predominante quanto a segurança na infraestrutura. Nesse sentido, nuvens *Infrastructure-as-a-Service* (IaaS) atribuem ao cliente maior liberdade com sua infraestrutura, o que torna a sua contribuição para a execução de medidas de segurança na nuvem mais significativa do que para nuvens *Software-as-a-Service* (SaaS), cenário onde o provedor possui tal atribuição. Especificamente, nuvens computacionais IaaS compreendem uma forma de oferta de serviços genérica da qual as demais podem ser reproduzidas. Sendo pertinente pois mesmo para infraestruturas de nuvens SaaS, o que existe e carece de métricas de segurança afeta a nuvem holísticamente [1, 7].

De modo que faz-se relevante o que ocorre nas infraestruturas tanto de clientes quanto de provedores. Contudo, foi identificada na literatura uma carência de estudos voltados à segurança da infraestrutura de provedores de uma nuvem computacional (*e.g.,* domínio de controle, orquestração dos recursos físicos, *etc.*) [6, 7]. Tal fato implica em uma maior probabilidade das infraestruturas dos provedores de nuvens computacionais serem comprometidas, o que se deve a carência de ferramentas que auxiliem na investigação de padrões maliciosos, podendo afetar diretamente na qualidade dos serviço providos aos seus clientes [3]. Portanto, o presente trabalho prima viabilizar aos provedores de nuvens computacionais abertas OpenStack, uma solução que capacite-os à realização de análises de segurança sob o tráfego de controle, dentro de sua infraestrutura

de nuvem. Isso é feito com o intuito de possibilitar ao provedor o aperfeiçoamento de sua infraestrutura de rede e do serviço provido aos seus clientes.

2. Definições

2.1. Domínio de controle

O domínio de controle faz-se presente na infraestrutura do provedor de uma nuvem computacional e compreende as redes de controle e de armazenamento [5, 7]. Sob condições ideais a natureza do tráfego de uma rede de controle deve ser correspondente aos serviços, suas comunicações e as requisições dos diferentes usuários da nuvem (*e.g.,* cliente, administrador). Contudo, a relatada carência de estudos de segurança na infraestrutura do provedor, implica que de dois, um ou o outro ocorre, a inexistência de risco tangente, ou a sua existência. Nesse contexto, levando a existência de nuvens computacionais sujeitas às atividades maliciosas e que podem prejudicar o serviço provido ao cliente [3]. Neste contexto a motivacão do foco da investigação científica se dá pela carência de meios de mensuração, a respeito do que ocorre nesse domínio de rede.

2.2. *Honeypots* de baixa interatividade

Os *honeypots* são ferramentas de segurança passivas, imunes contra falsos positivos, negativos e incapazes de proteger infraestruturas de redes contra ataques, mas que oferecem meios para a realização de análises de segurança [4, 8]. Seu *modus operandi* consiste da oferta de serviços e sistemas operacionais (SOs) (*e.g.,* emulados, reais), que convidam intencionalmente atacantes a comprometê-lo, a fim de coletar dados a respeito do ataque e do próprio, sendo capazes de descrevem a interação do atacante com a ferramenta durante o ataque e a sua intenção. Adicionalmente, o tráfego capturado por *honeypots* é sempre de natureza maliciosa em função da forma como o mesmo é implantado. Nesse sentido, existe uma relação qualitativa entre coleção de dados, o nível de interação e o risco atrelado a infraestrutura de rede onde a solução é implantada, sendo atreladas às classes (*e.g.,* baixa, média e alta interatividade) [8]. Das quais, dentre essas classes, o presente trabalho utiliza-se de uma solução de *honeypot* de baixa interatividade (HBI). Similarmente, a solução pode ser implantada em redes de pesquisa para identificar novas ameaças cibernéticas e em redes de produção para o aperfeiçoamento da infraestrutura de rede de organizações.

Os HBI consistem de soluções com menor complexidade de uso, administração e baixo risco à infraestrutura de rede onde são implantados por não oferecerem serviços genuinos à interação do atacante. Essa limitação restringe o serviço que é provido ao atacante à emulação e capacidade de coleção de dados às circunstâncias do ataque e não ao ataque em si (*e.g.,* IP da máquina emissora, IP da máquina receptora, horário do ataque, *Time to Live* (TTL) do SO utilizado, *etc.*). O que torna HBI interessantes para ambientes computacionais complexos por não serem intrusivos (*e.g.,* contêineres, nuvens computacionais tradicionais e de alto desempenho) [4]. Enquanto proveem meios suficientes para obter conhecimento da origem dos ataques, tornando possível em específico, caso seja interno à infraestrutura da nuvem, a sua identificação e correlação aos atacantes via análises de segurança sob os dados obtidos.

3. Arquitetura do Haa2N

A arquitetura do *Honeypot as a Nova Node* (Haa2N) consiste de uma máquina física com SO, na qual reside um HBI responsável por prover o serviço de criação de instâncias de máquinas virtuais (MVs), semelhantemente a um nó de computação Nova. Tendo a oferta de seu serviço realizada pela mesma porta TCP/IP do serviço de computação Nova, TCP/IP 8774 [5]. No entanto, o atacante pode unicamente interagir com o serviço emulado. O nó de Haa2N é conectado fisicamente à rede de controle por meio de uma interface de rede, tornando-o acessível a partir dessa rede. Não obstante, esse nó não oferta um serviço genuino de computação Nova, de modo que o nó de controle Nova não o agrega à base de dados *Structured Query Language* (SQL), que é usada durante a execução do escalonador-Nova para determinar qual o nó de computação vai armazenar e computar a nova instância. Nesse sentido, o nó de Haa2N é ignorado pelo escalonador-Nova, o que faz com que seu serviço seja unicamente interagido por meio de técnicas inconvencionais de acesso à rede com a nuvem computacional, que não utilizem-se da *web dashboard* do OpenStack. De modo que o *modus operandi* do nó de Haa2N e as condições para a interação com o seu serviço, tornam todo tráfego recebido de natureza maliciosa. O serviço provido pelo nó de Haa2N é configurável, o que permite a sua modificação ou a simulação de outros serviços específicos aos propósitos da equipe de segurança do provedor (*e.g.,* identificação, resposta, monitoramento, coleção de malware, identificação de motivações, *etc.*).

4. Projeto de implementação

O projeto de implementação consiste das tecnologias que se fazem necessárias para a implementação e aplicação da solução de Haa2N dentro do ambiente de experimentação, são elas:

- Nuvem computacional/*release*: OpenStack/Newton.
- Sistema operacional: OpenBSD.
- Solução de HBI: Honeyd.
  - *Listener script*: serviço de criação de máquinas virtuais (MVs).

O escolha da *release* Newton do OpenStack é motivada pela forma de desenvolvimento do OpenStack focar-se em criar e consolidar os serviços básicos, enquanto adiciona novos. O que não inviabiliza a implantação da solução de Haa2N em *releases* posteriores do OpenStack. O OpenBSD 6.1 é o SO cuja escolha justifica-se por ser robusto e fazer uso ativo de boas práticas de segurança. Adicionalmente, o OpenBSD residirá no hardware de uma máquina física e sobre este SO será instalada uma solução de HBI.

Por fim, o Honeyd é a solução de HBI a ser instalada no OpenBSD, escolhido por ser uma solução com arquitetura aberta, o que permite a implementação de serviços específicos e adicionais aos objetivos dos times de segurança de uma organização. Essa liberdade é implementada por meio de um *listener* que corresponde a um *script* no Honeyd. No caso específico da solução deste presente trabalho, o *listener* implementado dispõe do serviço de criação de MVs do Nova, na qual envia uma mensagem de erro ao atacante, enquanto faz a coleção dos dados.

5. Ambiente de testes

A fim de obter dados estatísticos da correlação, a experimentação decorre em dois ambientes: a experimentação em uma nuvem de testes, e na nuvem de produção. O primeiro

teste tem por objetivo verificar se a implantação está correta, além de mitigar o máximo possível das vulnerablidades da arquitetura, antes de tê-la no segundo ambiente.

6. Considerações & Trabalhos futuros

A partir da teoria de HBI e do serviço implementado no nó de Haa2N é possível contribuir com a identificação de ataques internos e externos, enquanto a correlação resume-se aos ataques internos à rede de controle. De modo que a correlação é condicionada a uma razão inversamente proporcional ao tamanho da infraestrutura do provedor da nuvem computacional, na qual quanto menor for, mais precisão existe. O aspecto da correlação é probabilístico por depender de aspectos da organização (*e.g.,* quantidade de funcionários, regime de trabalho, *etc.*) e o entendimento obtido pela equipe de segurança por meio da análise, que aumenta de complexidade de acordo com o tamanho da infraestrutura do provedor.

A experimentação inicialmente é conduzida em uma pequena infraestrutura de nuvem privada de alto desempenho OpenStack, presente no Laboratório de Processamento Paralelo e Distribuído (LabP2D). Posteriormente é a intenção dos autores, a implantação dessa implementação em uma infraestrutura de nuvem computacional de grande porte, como a disposta pela Yellow Circle, onde os mesmos testes serão aplicados e um estudo comparativo dos resultados vai ser conduzido.

7. Agradecimentos

Os autores agradecem o apoio do Laboratório de Processamento Paralelo e Distribuído (LabP2D) no CCT da Universidade do Estado de Santa Catarina (UDESC). As opiniões expressas no presente artigo são de responsabilidade dos autores e não refletem a oficial política da UDESC.

Referências

[1] CSA. Security Guidance For Critical Areas of Focus in Cloud Computing v4.0. 2017.

[2] Ryan Ko et al., editor. *The cloud security ecosystem: technical, legal, business and management issues*. Syngress is an imprint of Elsevier, Waltham, MA, USA, 2015. OCLC: 900028079.

[3] Ronald L. Krutz et al. *Cloud security: a comprehensive guide to secure cloud computing*. Wiley, Indianapolis, Ind, 2010. OCLC: 699803939.

[4] Mohssen Mohammed et al. *Honeypots and Routers: Collecting Internet Attacks*. CRC Press, 1st edition, 2015.

[5] OpenStackPike. OpenStack Docs: Pike, 2017.

[6] Y. Qiu et al. A Secure Virtual Machine Deployment Strategy to Reduce Co-residency in Cloud. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 347–354, 2017.

[7] Tiago Rosado et al. An Overview of Openstack Architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, IDEAS '14, pages 366–367, New York, USA, 2014. ACM.

[8] L Spitzner. *Honeypots: tracking hackers*. Addison-Wesley, Boston, 1st edition, 2003.

Análise do Consumo Energético de uma Aplicação de Alto Desempenho em Dinâmica de Fluidos

Cristian M. Weber, Lucas R. de Araujo
Fernando E. Puntel, Andrea S. Charão, João Vicente F. Lima

[1] Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria

Resumo. *Com o progresso do desempenho de supercomputadores ao longo dos anos, o consumo energético passou a ser um aspecto muito relevante. O presente trabalho visa verificar a eficiência energética do Incompact3D, um software importante da área de dinâmica de fluidos. Os dados foram coletados a partir de diferentes configurações de execução da aplicação, retratando um comportamento de consumo energético até então desconhecido para o Incompact3D.*

1. Introdução

Durante muito tempo, a comunidade de *High Performance Computing* (HPC) manteve foco na obtenção de um maior poder de processamento, destacando como motivos de tal melhora o aumento de *clock* e uso da tecnologia de múltiplos processadores. No entanto, estes fatores são contribuintes de um consequente aumento de gasto energético. Dessa forma, os sistemas HPC passaram a ser avaliados não só pelo desempenho, mas também pela eficiência de consumo de energia, assim ampliando a discussão e a quantidade de estudos a respeito da relação entre esses dois aspectos.

Uma área de aplicação de computação de alto desempenho que está em grande expansão é a área de dinâmica de fluidos, que envolve o desafiador problema de física clássica de fluxos turbulentos, caracterizado por movimentos complexos e desordenados em uma ampla escala de tempo e espaço [Laizet e Li 2011]. Neste cenário, o *software* Incompact3D é uma importante ferramenta que provê a possibilidade de estudo da área.

Devido à complexidade do problema a ser tratado por este *software*, é essencial o uso da aplicação em sistemas multiprocessados, a fim de evitar um gasto elevado de tempo de execução. Para este tipo de aplicação, o uso de máquinas com arquitetura do tipo *cluster* e servidores é recomendado. De fato, o *Incompact3D* é fortemente beneficiado por uma paralelização baseada no padrão *Message Passing Interface* (MPI) para comunicação entre processos, já tendo sido executado em até O(10^5) núcleos [Laizet e Li 2011] distribuídos em um grande *cluster*.

Embora já existam análises de desempenho do Incompact3D, seu consumo energético ainda não foi alvo de investigações. Assim, neste contexto, será apresentado e comparado neste trabalho o impacto sobre o consumo energético de diferentes situações e configurações da execução do Incompact3D em um sistema de múltiplos núcleos.

2. Trabalhos Relacionados

A importância do estudo da relação entre eficiência energética e HPC é reafirmada pela variedade de pesquisas neste tema. Fora do contexto de dinâmica de fluidos, em

[Bez et al. 2015] foi analisado o consumo energético de uma aplicação com alto custo de processamento em um *cluster* não convencional, na área de simulação de eventos geofísicos. Os resultados se mostraram favorecidos ao usar *flags* de otimização *energy to solution*, com uma redução de consumo de mais de 50%.

Em [Shafik et al. 2015] é explorada a interface de programação multiprocessada OpenMP juntamente com a aplicação de *Dynamic Voltage and Frequency Scaling*, a fim de alcançar uma minimização de energia consumida. Através da abordagem inovadora proposta por este trabalho, foi verificado que é possível alcançar uma economia de energia de até 17% em relação as abordagens já existentes.

3. Incompact3D

O estudo sobre o comportamento de escoamentos é de interesse de diversas áreas. Na prática, a física desses escoamentos é inconsistente e desordenada e tem aspecto comportamental turbulento, dependente de tempo e espaço. Dessa forma, a ampliação de conhecimento de tal aspecto é muito importante para entender os resultados que são afetados. O escoamento de um fluido é descrito pelas equações diferenciais de Navier Stokes. Para resolver tais equações é exigido uma alta capacidade de processamento de cálculos, ainda mais em casos turbulentos. Assim, o Incompact3D é uma poderosa ferramenta capaz de realizar a simulação de casos de escoamentos altamente complexos através de um grande nível de paralelização. Esta ferramenta utiliza como método a simulação numérica direta (DNS), que compõe uma solução numérica precisa e se baseia na resolução das equações que descrevem a dinâmica de fluidos.

A alta paralelização da solução do problema permitida no uso do Incompact3D, assim como a decomposição da malha do plano cartesiano 3D, é alcançada através da integração com a biblioteca de comandos de decomposição 2D e uma interface de Transformações Rápidas de Fourier (2DECOMP&FFT) [Li e Laizet 2010]. Assim, os passos *convection/diffusion*, *velocity divergence* e *pressure gradient* [Laizet e Li 2011] da resolução das equações de Navier Stokes se beneficiam de rotinas de transposição que envolvem a comunicação entre processos, disponibilizadas por essa biblioteca auxiliar.

4. Materiais e Métodos

A máquina utilizada para a coleta de dados é uma NUMA SGI UV2000 de arquitetura com 48 núcleos de processamento (Intel® Xeon® CPU E5-4617 @ 2.90GHz) e 512GB de memória RAM. As execuções paralela realizadas usam MPI com a versão 2.0.2 da distribuição OpenMPI. A coleta de dados a respeito do gasto energético foi feita utilizando o LIKWID [Treibig et al. 2010] na versão 4.2.1.

A escolha do LIKWID como instrumento de monitoramento se dá pelo fato de ser fácil de ser utilizado, além de apresentar uma portabilidade considerável. Núcleos múltiplos podem ser medidos simultaneamente e é adequado tanto para arquiteturas de processadores x86 Intel como AMD, em um ambiente Linux. Dentre as diversas ferramentas de monitoramento disponíveis a partir do LIKWID, foi usado neste trabalho a *likwid-powermeter*. Essa ferramenta permite consultar a energia total consumida no período de execução monitorado através de contadores RAPL disponíveis a partir da criação pela Intel da arquitetura *SandyBridge*. Além disso, é possível definir os *sockets* a serem utilizados pela aplicação.

Os testes visaram a comparação em diferentes situações de execução. Em relação ao Incompact3D, foram utilizados valores distintos das variáveis relativas à dimensão da malha do plano cartesiano 3D para diferentes complexidades do problema. Além disso, também variou-se a quantidade do número de processos a serem utilizados pelo MPI.

5. Resultados e Discussão

Em um primeiro momento, foi realizada a coleta de dados do consumo energético a partir de diferentes número de processos, a fim de examinar o comportamento da aplicação em diferentes situações de paralelização. Os valores analisados em cada um dos oito experimentos feitos para cada situação de execução se aproximaram suficientemente das médias calculadas, tornando-se desprezível a apresentação das variabilidades em relação a média final. Os valores obtidos são apresentados na Tabela 1 e são referentes a uma média aritmética de cada um dos oito *sockets* da máquina utilizada. A partir destes dados, é possível verificar uma diminuição na energia consumida por *socket* e do tempo de execução baseados no aumento do uso dos núcleos disponíveis para a aplicação do Incompact3D. Além disso é importante destacar o aumento da potência consumida que, no entanto, se opõe à redução do gasto energético, a qual é preservada devido à compensação da diminuição do tempo de execução da aplicação com a exploração do uso de mais núcleos. É notável também, a estabilização da energia consumida verificada a partir do uso de 32 processos, que, porém, possui um consumo de potência inferior ao consumo da quantidade máxima de processos usados neste experimento.

Número de Processos	Energia Consumida (Joules)	Potência Consumida (Watt)	Tempo de execução (Segundos)
8	2549,47	41,91	60,83
12	1899,32	46,03	41,26
24	1579,15	57,06	27,67
32	1398,04	64,70	21,61
48	1395,90	75,02	18,60

Tabela 1. Energia e potência consumida referente ao número de processos utilizados

O Incompact3D permite ao usuário configurar o tamanho da malha do problema a ser resolvido. Dessa forma, foram realizados testes com o objetivo de verificar o impacto dessas alterações no gasto energético. Novamente, os dados representam uma média de cada um dos 8 *sockets* em uma execução de 48 processos. A partir dos resultados obtidos em duas configurações distintas, que se encontram na Tabela 2, foi verificado que, ao ser feito um aumento de aproximadamente 13,46 vezes do volume da malha da configuração 1 para a configuração 2, a variação do consumo energético não se mostra proporcional. De fato, foi constatado um crescimento de mais de 15.7 vezes da energia consumida ao modificar os valores das variáveis de dimensão nx, ny e nz.

6. Considerações Finais

A partir dos testes realizados neste trabalho, pôde-se observar um comportamento de consumo energético do Incompact3D, que até então não havia sido investigado. Tais resultados reforçam a ideia de que o usuário deve estar ciente do efeito negativo sobre

	Dimensões (nx, ny, nz)	Nós da malha 3d	Energia Consumida (Joules)
Configuração 1	128, 257, 128	4.210.688	1395,9
Configuração 2	256, 865, 256	56.688.640	22010,2

Tabela 2. Relação da energia consumida e o tamanho da malha de duas configurações

o consumo de energia ao simular problemas com larga escala da malha cartesiana ao aproveitar o benefício de uso de supercomputadores na resolução de tais problemas. Em trabalhos futuros, poderão ser explorados métodos para otimização do gasto energético que não apresentem efeitos negativos consideráveis no desempenho, principalmente visando a utilização de malhas maiores.

7. Agradecimentos

Os autores agradecem a FAPERGS-CNPQ PRONEX pelo financiamento provido para a realização do trabalho.

Referências

Bez, J., Bernart, E., dos Santos, F., Schnorr, L., e Navaux, P. (2015). Análise da eficiência energética de uma aplicação hpc de geofísica em um cluster de baixo consumo.

Laizet, S., Lamballais, E., e Vassilicos, J. (2010). A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution dns of fractal generated turbulence. *Computers & Fluids*, 39(3):471 – 484.

Laizet, S. e Li, N. (2011). Incompact3d: A powerful tool to tackle turbulence problems with up to O(105) computational cores. *International Journal for Numerical Methods in Fluids*, 67:1735–1757.

Li, N. e Laizet, S. (2010). 2decomp & fft-a highly scalable 2d decomposition library and fft interface. In *Cray User Group 2010 conference*, pages 1–13.

Shafik, R. A., Das, A., Yang, S., Merrett, G., e Al-Hashimi, B. M. (2015). Adaptive energy minimization of openmp parallel applications on many-core systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, PARMA-DITAM '15, pages 19–24, New York, NY, USA. ACM.

Treibig, J., Hager, G., e Wellein, G. (2010). Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA.

Análise do Controle de Congestionamento TCP em *Data Centers* de Nuvens IaaS

Guilherme Xavier[1], Arthur Schuelter[1], Guilherme Koslovski[1]

[1]Universidade do Estado de Santa Catarina – UDESC – LabP2D

{guilherme.carvalho, arthur.schuelter}@edu.udesc.br
guilherme.koslovski@udesc.br

Resumo. *O compartilhamento justo dos recursos de rede é foco comum no desenvolvimento de protocolos e data centers. Especificamente em nuvens de IaaS, nas quais os inquilinos podem configurar máquinas virtuais, a justiça de compartilhamento é afetada pelos diversos algoritmos para controle de congestionamento no TCP. Assim, o presente trabalho analisa o desempenho dos algoritmos quando executados em um cenário heterogêneo, comum nas nuvens IaaS.*

1. Introdução

As nuvens computacionais que ofertam infraestruturas como serviço são comumente compartilhadas por múltiplos inquilinos. A reserva de recursos ocorre através de máquinas virtuais (MVs) dinamicamente disponibilizadas aos clientes de acordo com a configuração previamente definida. Ou seja, o gerenciamento interno do sistema operacional, bibliotecas e aplicações é realizado pelo contratante. Como resultado, é comum a execução concorrente de versões distintas de sistemas, entre eles, os protocolos de controle de congestionamento do TCP.

Algoritmos de controle de congestionamento inferem informações da rede para aumentar ou diminuir o volume de dados trafegados. Bem como, tratam os possíveis gargalos nos equipamentos de encaminhamento de pacotes, que além da perda dos mesmos, acarretam na diminuição da vazão de tráfego útil, aumento da latência e degradação do serviço oferecido pelo provedor IaaS. No entanto, ao combinar diversos algoritmos, o princípio de justiça de compartilhamento é perdido e as MVs apresentam desempenhos diferentes na ocorrência de congestionamento em *data centers*.

Nesse contexto, o presente trabalho apresenta uma análise sobre a justiça de compartilhamento e desempenho obtido ao combinar diferentes versões dos protocolos para controle de congestionamento. O estudo é realizado sobre o *Mininet* [Karamjeet Kaur 2014], uma ferramenta para prototipação e análise de redes virtualizadas de alto desempenho. Destarte, o artigo está organizado da seguinte maneira: a Seção 2 revisa conceitos importantes da literatura para a contextualização da problemática tratada; enquanto a Seção 3 trata da análise experimental, descrevendo os testes realizados e os resultados obtidos. A Seção 4 é reservada às conclusões e perspectivas de continuidade.

2. Revisão de Literatura

2.1. Algoritmos para Controle de Congestionamento TCP

Dado um ambiente compartilhado, é desejável que todos os usuários sejam mantidos suficientemente satisfeitos pelo serviço recebido. Advindo deste princípio, a justiça é implementada pelo TCP para promover esse equilíbrio, mesmo que existam diferentes algoritmos para controle de congestionamento operantes na rede [Floyd 1994]. No entanto, tais

algoritmos utilizados pelas MVs não são capazes de manter a rede homogênea para todos os usuários, uma vez que, cada mecanismo implementa as características do protocolo da sua maneira. Os proeminentes neste quesito são: *Cubic*, *Reno*, *BBR* e *Vegas*.

O *Cubic* é um algoritmo de crescimento rápido da janela de transmissão, seguindo uma função cúbica. Para promover justiça, estabiliza o crescimento da janela quando este se aproxima do limite da rede. Por sua vez, o *Reno* possui ênfase na reação do algoritmo frente à perda de pacotes, aplicando métodos de recuperação de vazão quando pacotes são perdidos e limitação do crescimento da janela quando a rede está congestionada. O algoritmo *BBR*, desenvolvido pela *Google*, fornece alta vazão e justiça em ambientes congestionados e com alta latência. Por fim, o *Vegas*, diferente das outras implementações, tenta detectar o congestionamento antes de ocorrer a perda de pacotes. Ele estabelece valores de vazão máxima e mínima, executando de maneira eficiente dentro desse intervalo.

2.2. *Explicit Congestion Notification* - ECN

O ECN é uma das alternativas elaboradas para que se reduza a perda de pacotes em um *data center*. O mecanismo notifica os emissores e receptores sobre o congestionamento na rede por meio de marcações nos pacotes transmitidos. Tais marcas sinalizam o congestionamento da rede ao chegar aos pontos finais da conexão, fazendo com que o fluxo de transmissão seja reduzido e a perda de pacotes seja amenizada antes mesmo que ocorra [Bauer et al. 2011].

2.3. Trabalhos Relacionados

Atualmente alguns projetos apresentam propostas para viabilizar a equidade em *data centers*. O AC/DC (*Administrator Control over Data Center*) propõe que os administradores da rede sejam capazes de alterar os algoritmos de controle de congestionamento sem alterar as configurações dos inquilinos da rede [He et al. 2016]. Já o DCTCP (*Data Center TCP*) sugere a utilização de um único algoritmo otimizado no *data center* [Alizadeh et al. 2010]. Assim, cada nó pertencente a rede acredita estar utilizando seu próprio protocolo, por conta disso a rede permanece homogênea. Por fim, o VCC (*Virtualized Congestion Control*) aprimora o funcionamento dos algoritmos e facilita as atualizações por meio dos *hypervisors*. Os quais recebem um algoritmo determinado e distribuem sua tradução entre os *hosts*, fazendo com que componentes legados operem junto aos benefícios de uma nova implementação [Cronkite-Ratcliff et al. 2016].

3. Análise Experimental

3.1. Cenário de Testes

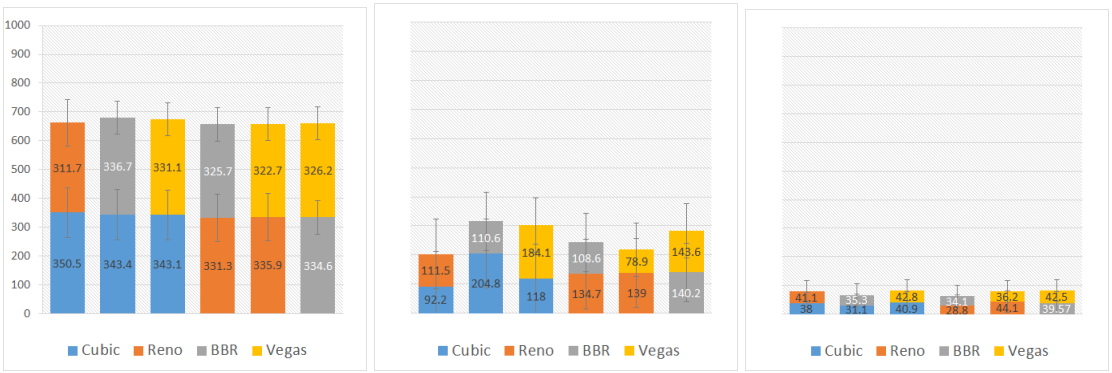
Para simular o cenário de concorrência por recursos em uma nuvem IaaS, um ambiente de testes foi criado utilizando a ferramenta *Mininet*. A topologia adotada foi a de Halteres, ou seja, MVs foram interconectadas por 2 *switches*, onde os enlaces foram configurados com 1 Gbps, sendo que a disputa por recursos ocorre na conexão dos *switches*. Para cada par cliente-servidor condicionado na execução, foi atribuído um algoritmo TCP diferente, dentre os elencados na Seção 2.1. Ainda, foram determinadas duas abordagens de controle de congestionamento. Na primeira, os algoritmos executavam sem o suporte do ECN, enquanto na segunda, ele estava presente junto ao RED (*Random Early Detection*). Um algoritmo que atua no controle de filas do roteador, selecionando pacotes aleatórios para descartar, prevenindo assim, o surgimento de filas. Este, foi configurado segundo a

literatura de [Alizadeh et al. 2010]. O tamanho do *buffer* foi estabelecido em 10^6, sendo que para o mínimo da extensão da fila de marcação foi dado $9 * 10^4$ e para o máximo foi adicionando 1 a este último valor. Os pacotes permitidos por rajada de descarte foram 61, com probabilidade 1 e tamanho médio do pacote em 1500.

Dada a infraestrutura, 3 casos foram preparados com diferentes composições: a primeira composta por 4 MVs, a segunda por 8 e a última por 16. Optou-se por desenhar essas 3 organizações para analisar o comportamento dos algoritmos em ambientes com configurações distintas, por exemplo, clientes que reservaram MVs replicadas (com o mesmo algoritmo). Por conseguinte, amostras (10 execuções) foram colhidas e os resultados compõem uma média com desvio padrão de aproximadamente 70 Mbps. Vale ressaltar que, para a elaboração dos gráficos elaborados foi utilizado um intervalo de confiança de 95%.

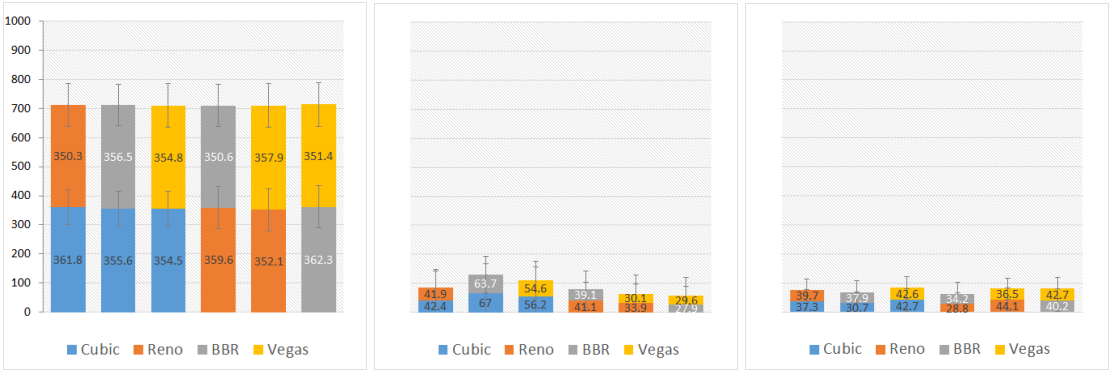
3.2. Discussão dos Resultados

Os resultados para a execução sem suporte ECN e com suporte são apresentados pelas Figuras 1 e 2, respectivamente. Vale ressaltar que, a análise realizada utilizou o emulador *Mininet*, cujos resultados produzidos podem se afastar, levemente, de situações reais.



(a) 4 MVs. (b) 8 MVs. (c) 16 MVs.

Figura 1. Compartilhamento do enlace de gargalo sem suporte ECN.



(a) 4 MVs. (b) 8 MVs. (c) 16 MVs.

Figura 2. Compartilhamento do enlace de gargalo com suporte ECN.

Ao avaliar os dados, nota-se que em uma distribuição com poucas MVs competindo pelo enlace (Figuras 1(a) e 2(a)), a vazão utilizada pelos pares é demasiada-

mente maior se comparada com os valores obtidos com conjuntos maiores (Figuras 1(c) e 2(c)). Uma vez que a quantidade de componentes na infraestrutura influencia a vazão de cada par. Pois, a vazão depende da capacidade dos enlaces da rede e não do número de máquinas virtuais.

Quanto aos algoritmos, as versões *Cubic* e *Reno* proporcionaram uma divisão próxima do ideal, enquanto o *BBR* e *Vegas*, apresentaram dispersões significativas no compartilhamento do canal. Ainda, é possível observar que para a maior parte dos casos com ECN ativo (Figura 2), houve uma aproximação nas métricas. Desse modo, o ideal de justiça foi devidamente visualizado através do comportamento similar dos algoritmos heterogêneos envolvidos. De modo geral, a aplicação de ECN contribuiu para que os recursos em rede fossem distribuídos com maior equilíbrio entre os inquilinos solicitantes.

No entanto, neste experimento, a quantidade de pacotes perdidos não foi devidamente otimizada, dada a agressividade no descarte dos pacotes oriundos da configuração dos *switches*. Em ambientes virtualizados de IaaS, a aplicabilidade de métodos que buscam a justiça são de suma importância quando se objetiva um comportamento homogêneo e eficiente do sistema.

4. Considerações Finais

A análise de justiça realizada sobre os algoritmos de controle de congestionamento indica que aplicação do ECN em uma infraestrutura distribuída, tal qual a de *data centers* de nuvens IaaS, aproximou as taxas de vazão entre as MVs, tornando o sistema mais justo e homogêneo no compartilhamento de recursos disponíveis em rede. Quanto aos trabalhos futuros, experimentos com outras versões de algoritmos para controle de congestionamento são previstos, bem como estudos sobre a virtualização deste gerenciamento.

Agradecimentos: Financiado pela UDESC e FAPESC, sendo desenvolvido no LabP2D.

Referências

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). In *Proc. of the SIGCOMM Conference*.

Bauer, S., Beverly, R., and Berger, A. (2011). Measuring the state of ecn readiness in servers, clients,and routers. In *Proc. of the SIGCOMM Conf. on Internet Measurement Conference*, IMC '11.

Cronkite-Ratcliff, B., Bergman, A., Vargaftik, S., Ravi, M., McKeown, N., Abraham, I., and Keslassy, I. (2016). Virtualized congestion control. In *Proc. of the SIGCOMM Conference*.

Floyd, S. (1994). Tcp and explicit congestion notification. *SIGCOMM Comput. Commun. Rev.*, 24.

He, K., Rozner, E., Agarwal, K., Gu, Y. J., Felter, W., Carter, J., and Akella, A. (2016). Ac/dc tcp: Virtual congestion control enforcement for datacenter networks. In *Proc. of the SIGCOMM Conference*.

Karamjeet Kaur, Japinder Singh, G. S. (2014). Mininet as software defined networking testing platform.

Análise do Software Paralelo AHF para Identificação de Estruturas em Cosmologia

Ana Luísa V. Solórzano[1], **Andrea S. Charão**[1]
Renata S. R. Ruiz[2], **Haroldo F. de Campos Velho**[2]

[1] Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria

[2]Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais

Resumo. *O estudo da formação de estruturas do Universo a partir de dados cosmológicos suscita a criação de programas paralelos. Um deles é o AHF, que é paralelizado usando MPI e OpenMP. Neste trabalho, investiga-se o seu comportamento em execuções com OpenMP em uma arquitetura multicore. Para o caso considerado, não houve vantagem de desempenho com o uso de OpenMP, o que motiva uma avaliação mais aprofundada de sua paralelização.*

1. Introdução

Análises de grandes quantidades de dados simulados e observacionais têm sido amplamente empregadas em Cosmologia, que estuda a origem, estrutura e evolução do Universo. Um dos aspectos investigados nesta área é a formação de estruturas cósmicas, tais como, estrelas, galáxias e aglomerados de galáxias [Madsen 1996]. Para isso, dado um grande conjunto de partículas proveniente de simulações ou observações, é necessário identificar e classificar grupos de partículas interligadas. Isso é conhecido como *halo finding* [Gill et al. 2004].

O interesse da comunidade científica, aliado à evolução de algoritmos e ferramentas computacionais, fez com que surgissem diversos algoritmos e programas chamados *halo finders*. Este tipo de *software*, muitas vezes, é desenvolvido por especialistas em Cosmologia com conhecimento em programação. Do ponto de vista cosmológico, essa variedade de códigos tem suscitado questões sobre semelhanças e diferenças de seus resultados [Knebe et al. 2013]. Do ponto de vista computacional, observa-se que muitos desses códigos empregam ferramentas de programação como OpenMP [Chapman et al. 2007] ou MPI [Pacheco 1996], porém não é comum encontrar análises sobre seu desempenho em arquiteturas paralelas.

Neste trabalho, analisa-se o desempenho do *software* Amiga Halo Finder (AHF), que é um dos códigos citados por especialistas na área que proporciona a sua paralelização com OpenMP e MPI. Nesta pesquisa exploratória, tem-se como objetivo geral compreender o comportamento do AHF em uma arquitetura paralela *multicore*, usando métricas e critérios não observados em publicações sobre este código.

2. AHF – Amiga Halo Finder

O Amiga Halo Finder é um *software* para encontrar estruturas de *halos* e *sub-halos* em simulações cosmológicas, a partir de propriedades definidas pelo usuário, realizando

execuções seriais ou paralelas [Knebe e Knollmann 2009]. O seu fluxo de trabalho consiste em encontrar as estruturas, identificar partículas relacionadas a elas, remover as não relacionadas e calcular suas propriedades físicas e estatísticas.

Para isso, o programa utiliza um arquivo de texto intitulado `AHF.input`, que é passado por parâmetro em sua execução para definir propriedades quanto à classificação das partículas, e um arquivo de dados proveniente de simulações cosmológicas com ferramentas como Gadget-2[1], AMIGA[2], e outras desde que no fomato ASCII. Para estas, o arquivo de dados deve conter um cabeçalho com seus parâmetros cosmológicos, que pode ser inserido utilizando um *script* em Shell disponibilizado pelo *software*. Entretanto, como o *script* insere alguns valores padrões relacionados a esse tipo de simulação, é necessário configurá-los no próprio código para resultados mais precisos.

Cada execução gera arquivos de saída contendo informações diversas, sendo o principal deles um arquivo de log que relata os valores de entrada, o número de *halos* encontrados, o tempo de execução do algoritmo *halo finder*, o tempo de execução total do programa e o consumo de memória. Os demais arquivos listam os parâmetros utilizados, as propriedades de cada *halo* e as partículas associadas a ele.

Diferente de outros *halo finders* que predefinem os grupos de partículas a serem processados em paralelo, o AHF divide o volume de dados entre os processadores durante a sua execução [Kim e Park 2006, Knebe e Knollmann 2009]. Em suas primeiras versões, essa paralelização era feita com o padrão MPI, já a versão atual também disponibiliza a opção com OpenMP e MPI com OpenMP, definida por *flags* de compilação.

Em um trabalho anterior, seus autores reportam o desempenho de execuções paralelas com MPI considerando diferentes valores para o balanceamento de carga entre as máquinas e para o parâmetro que define as zonas de fronteira entre partículas [Knebe e Knollmann 2009]. Entretanto, trabalhos que relatem o desempenho da execução paralela para as novas opções não foram encontrados.

3. Experimentos

Para obter o arquivo de dados observacionais, utilizou-se uma amostra do Consórcio Virgo, proveniente do modelo cosmológico LCDM (Lambda Cold Dark Matter) do projeto de Simulações de Escala Intermediária. A amostra define 1.048.576 partículas, cada uma com massa de 6.86 x 10^{10} Msun/h e parâmetros cosmológicos (Ω_M , Ω_Λ , h, σ_8) = (0.3, 0.7, 0.7, 0.9). Os dados, bem como os seus parâmetros numéricos e cosmológicos, encontram-se disponíveis em: `http://wwwmpa.mpa-garching.mpg.de/Virgo/data_download.html`.

Um ambiente propício para pesquisadores em cosmologia realizarem investigações preliminares sobre o desempenho de seus algoritmos é o de arquiteturas paralelas *multicore*. Esse ambiente é mais comum e acessível em comparação ao uso de *clusters*, que podem gerar filas de espera, por exemplo. Assim, optou-se por realizar 30 execuções do programa para 1, 2 e 4 *threads* com o padrão OpenMP.

A máquina utilizada contém um processador Intel® Core i5-5200U de dois cores físicos e quatro virtuais, com 32KB de cache L1, 256KB de cache L2 e 3MB de cache L3,

[1]http://wwwmpa.mpa-garching.mpg.de/gadget/
[2]http://popia.ft.uam.es/AMIGA/

e 4 GB de memória. Sistema operacional Ubuntu 16.04.3 LTS, versão do Linux 4.4.0-103, e compilador gcc versão 5.4.0.

Visando aprofundar a compreensão do código e de sua execução paralela, foram utilizadas duas ferramentas. A primeira foi a Understand[3], que analisa códigos fonte de projetos e oferece uma API para a investigação, edição e visualização de gráficos sobre o projeto. A segunda foi a Intel VTune Amplifier[4], uma ferramenta da Intel para analisar o perfil de execução de códigos, de modo a identificar gargalos quanto ao uso da CPU em execuções *multithreading*.

4. Resultados e Discussão

A execução dos experimentos resultou nos dados observados na Tabela 1. Os tempos calculados foram obtidos no arquivo de saída de log do programa, o qual relata separadamente o tempo do *halo finder*, que consiste na identificação das estruturas cosmológicas, e o tempo total, que também engloba a manipulação do arquivo de dados e outras computações relacionadas às propriedades das partículas. Visando legitimar esses valores, eles foram comparados a execuções utilizando o comando `time` do Unix.

Threads	Média(s) Tempo *halo finder*	Média(s) Tempo Total
1	24.70	66.71
2	24.56	66.67
4	24.53	66.74

Tabela 1. Médias dos tempos de execução

A partir desses resultados, realizaram-se execuções com o Intel Vtune, onde se observou que o programa faz pouco uso do processamento paralelo durante a sua execução, justificando o desempenho observado. Partindo disso, investigou-se quais funções do AHF requerem um maior tempo de processamento e como as diretivas OpenMP são aplicadas.

4.1. Análise da arquitetura do AHF

Com o Intel Vtune, constatou-se que o maior processamento está na preparação dos dados a serem identificados pelo algoritmo de *halo finding*. Nessa etapa, é aplicada a técnica de AMR (Adaptative Mesh Refinement) para classificar as partículas hierarquicamente em uma estrutura de árvore. Essa é a árvore que será percorrida pelo algoritmo, para definir as subestruturas de *halos* e *sub-halos*.

Com o Understand, analisaram-se as funções que fazem o refinamento dos dados. A mais importante delas é a `AMR_hierarchy`, que recursivamente prepara os dados a serem identificados e invoca a função mais custosa do programa, que refina os dados a partir das densidades das partículas e do número de partículas permitidas por nó da árvore, cujo valor é definido pelo usuário no arquivo `AHF.input`. Essa etapa do programa não faz uso de processamento paralelo.

[3]https://scitools.com/

[4]https://software.intel.com/en-us/intel-vtune-amplifier-xe

4.2. Análise da paralelização do AHF

As diretivas OpenMP são aplicadas em laços utilizados nos arquivos de escrita dos dados de saída e no arquivo responsável por percorrer a árvore. Porém, a maioria dos laços paralelizados processam cargas estáticas e têm grande parte das variáveis utilizadas definidas como privadas, o que interfere no desempenho já que resulta em cópias de variáveis entre as *threads* e pode resultar em *threads* ociosas, dependendo do tamanho de carga a ser distribuída.

Porém, mesmo executando de forma serial, o AHF apresenta um bom desempenho em comparação a outras ferramentas, conforme relatado em [Knebe et al. 2013]: dentre oito ferramentas correlatas, o AHF teve o segundo menor tempo de execução para dois arquivos de dados com tamanhos distintos, sendo até 84 vezes mais rápido comparado ao de pior desempenho.

5. Considerações Finais

Com base nas análises sobre o *software* paralelo AHF em um ambiente *multicore* utilizando OpenMP, observou-se que o programa não apresenta vantagens de desempenho em execuções com 2 e 4 *threads* em comparação à execução serial, para os dados analisados. Estima-se que isso ocorra devido aos autores do trabalho priorizarem o seu bom funcionamento e completude em relação a outros *halo finders*, oferecendo como fator adicional a possibilidade de sua execução em ambientes paralelos. Assim, como trabalhos futuros, planeja-se estudar o código fonte do programa a fim de rever e alterar a forma de paralelização para o padrão OpenMP em busca de um maior desempenho, além de realizar testes com MPI.

6. Agradecimentos

Os autores agradecem ao PIBIC-CNPq-INPE pelo suporte na forma de bolsa de Iniciação Científica para o projeto de nº 800012/2016-0.

Referências

Chapman, B., Jost, G., e van der Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, USA.

Gill, S. P. D., Knebe, A., e Gibson, B. K. (2004). The evolution of substructure - i. a new identification method. *mnras*, 351:399–409.

Kim, J. e Park, C. (2006). A new halo-finding method for n-body simulations. *ApJ*, 639:600–616.

Knebe, A. et al. (2013). Structure finding in cosmological simulations: the state of affairs. *Monthly Notices of the Royal Astronomical Society*, 435(2):1618–1658.

Knebe, A. e Knollmann, S. R. (2009). Ahf: Amiga's halo finder. *The Astrophysical Journal Supplement*, 182(2):608–624.

Madsen, M. S. (1996). In *The Dynamic Cosmos - Exploring the Physical Evolution of the Universe*, New York, NY, USA. Chapman & Hall.

Pacheco, P. S. (1996). *Parallel Programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Avaliação de Desempenho de Bibliotecas Java para o Protocolo Paxos

Paola Pereira[1], Rodrigo H. Müller[1], Cristina Meinhardt[2], Odorico M. Mendizabal[1]

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande - FURG
Rio Grande – RS – Brazil

{paolapereira,rodrigo.muller,odoricomendizabal}@furg.br

[2]Universidade Federal de Santa Catarina - UFSC

cristina.meinhardt@ufsc.br

Resumo. *Sistemas tolerantes a falhas normalmente adotam técnicas de replicação, como a Replicação Máquinas de Estados. Nesta técnica, um protocolo de ordenação garante que as réplicas executem uma mesma sequência de comandos, garantindo consistência. Devido ao bom desempenho, Paxos é um dos protocolos mais utilizados. Este artigo expõe uma análise de desempenho de bibliotecas Java para o Paxos, destacando a vazão alcançada por cada uma.*

Introdução

Tolerância a falhas é uma propriedade fundamental, especialmente em serviços críticos com requisitos de alta disponibilidade. Um dos meios de prover esta tolerância é através da replicação do serviço em diversas instâncias. Dentre as técnicas de replicação, a Replicação Máquina de Estados [Schneider 1990], ou replicação ativa, destaca-se pela simplicidade e garantia de consistência forte. Nesta abordagem, todos os comandos (requisições de clientes) são executados na mesma ordem por todas as réplicas, de forma determinística, garantindo que todas as réplicas passem pelos mesmos estados. Para garantir a entrega ordenada de requisições entre todas as réplicas, pode-se utilizar protocolos de difusão atômica ou de consenso. Neste sentido, o protocolo Paxos [Lamport 1998] destaca-se tanto pela sua relevância acadêmica quanto pela aplicação em ambientes corporativos [Burrows 2006, Rao et al. 2011], apresentando bom desempenho.

Embora o Paxos apresente um bom desempenho comparado com outros protocolos de ordenação, o custo de comunicação comparado com primitivas de comunicação tradicionais (não-confiáveis) é alto. Decisões de projeto para diferentes implementações do protocolo também influenciam o desempenho. Neste trabalho é feita uma análise de implementações Java de código aberto para o protocolo Paxos.

Paxos

Paxos [Lamport 1998] é um algoritmo de acordo que busca obter consenso de um valor entre as réplicas. Um algoritmo de consenso garante que apenas um dentre os valores propostos seja escolhido, passo fundamental para desenvolver aplicações utilizando Replicação Máquina de Estados. No Paxos existem três papéis distintos: *proposer*, *acceptor* e *learner*. O *proposer* propõe valores, que poderão ser aceitos pelos *acceptors*, desde que estes não tenham se comprometido com outro valor para a mesma instância

de consenso. Após um conjunto de *acceptors* concordarem com um valor, os *learners* aprenderão o valor escolhido. Assim, um processo nunca recebe um valor, a menos que este tenha sido aprendido. Cada réplica pode atuar em um dos papéis ou em mais de um, simultaneamente. O protocolo de consenso atua em duas fases, geralmente chamadas de fase 1 e 2. Cada uma destas é subdividida em duas fases (1a, 1b, 2a e 2b). Na fase 1, o *proposer* envia uma requisição do tipo *accept*, informando nela um identificador único, *i*, solicitando aos *acceptors* que, naquela época, ele possa propor um valor, se já não houver algum. Quando obter o aval de um quórum de *acceptors* ($N/2+1$, sendo N o número de *acceptors*), o *proposer* passa à fase 2, na qual envia uma requisição *commit*, informando um valor. Somente após o término da segunda fase, que o valor é efetivamente replicado.

S-Paxos[1]**:** O S-Paxos [Biely et al. 2012] foi desenvolvido utilizando como base o JPaxos, uma implementação *multi-threaded* do protocolo Paxos. O S-Paxos procura aliviar a carga sobre o coordenador (*proposer* no protocolo Paxos). Para isto, as mensagens são divididas em duas categorias: disseminação das requisições dos clientes, e ordenação das mensagens. Enquanto esta última fica a cargo do coordenador, as primeiras podem ser realizadas por qualquer réplica. Desta forma, todas as réplicas recebem pedidos dos clientes, encaminhando-as às demais.

BFT-SMaRt[2]**:** O BFT-SMaRt [Bessani et al. 2014] oferece implementação do protocolo Paxos com código-fonte aberto, desenvolvido na linguagem Java. Uma de suas principais características é a opção de tolerar falhas do tipo colapso (*crash*) ou arbitrárias (bizantinas). No primeiro modo, são necessárias $2f+1$ réplicas para suportar até f falhas. Ao suportar falhas bizantinas, porém, são utilizadas $3f+1$ réplicas. As mensagens são enviadas às réplicas através de canais seguros, utilizando um par de chaves assimétricas.

URingPaxos[3]**:** O URingPaxos [Benz et al. 2014] também implementa o protocolo Paxos em Java, com código aberto. Nesta implementação, *proposers, acceptors* e *learners* estão dispostos em uma topologia de anel. Esta abordagem favorece a vazão, dado o bom uso da largura de banda disponível. Também é possível configurar múltiplos anéis, formando grupos de *multicast*. Para efeitos de comparação, neste trabalho, é utilizado apenas um anel, caracterizando o protocolo *Ring Paxos*.

Avaliação

Para comparar as bibliotecas, um serviço gerenciador de travas distribuídas do tipo leitores/escritores foi implementado e replicado utilizando cada uma das implementações do Paxos (S-Paxos, BFT-SMaRt e URingPaxos). O modelo de falhas admitido é por colapso, logo, para tolerar 1 falha, todos os sistemas foram configurados com 3 réplicas (são necessários $2f+1$ *acceptors* para tolerar f falhas).

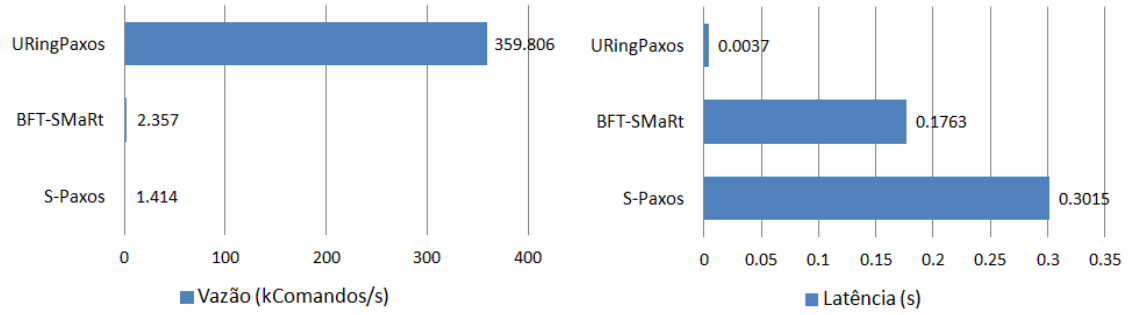
Todas requisições do serviço possuem um comando do tipo carácter (bloquear ou liberar), um identificador da trava do tipo inteiro (admitindo-se 1 milhão de valores) e um booleano indicando se é escrita ou leitura, totalizando aproximadamente 5 bytes. Foram utilizados computadores com processador Core i5-4570@3.20 GHz e memória RAM de 8 GB, interligados por um *switch* Ethernet. Para todos os testes, a JVM (máquina virtual Java) teve o *Heap* configurado com 6GB.

[1]https://github.com/nfsantos/S-Paxos

[2]https://github.com/bft-smart/library

[3]https://github.com/sambenz/URingPaxos

O objetivo dos testes foi encontrar a vazão máxima alcançada por cada biblioteca. Foram executados até 80 clientes por gerador de carga, começando com um computador e, gradativamente, aumentando o número até ser observada a degradação do sistema. Todos os experimentos executaram por um período de 4 minutos. Assim que o cliente obtém uma resposta, sorteia e envia outro comando para ser processado. A latência é calculada nos clientes, enquanto a vazão é medida nas réplicas. A latência, dada em segundos, corresponde à média do nonagésimo percentil de cada cliente, sendo os valores computados a cada 50 requisições. A vazão, dada em comandos por segundo, é a média entre as vazões registradas em cada réplica. A Figura 1(a) exibe a vazão máxima registrada com uso de cada uma das bibliotecas, enquanto a Figura 1(b) apresenta as latências resultantes.



(a) (b)

Figura 1. Comparativo de desempenho: (a) vazão máxima; (b) latência.

Com o S-Paxos observou-se uma vazão máxima de aproximadamente 1.400 comandos/s, com latência de 0,30 segundos. Após esse ponto, com a adição de mais clientes há degradação de desempenho, com o aumento da latência e sem acréscimo na vazão. O fator limitante da biblioteca parece ser a sua implementação, visto que ainda havia grande parte dos recursos disponíveis (memória, processamento e rede).

Para o BFT-SMaRt alcançou-se uma vazão máxima de 2.360 comandos/s, embora não tenha ocorrido deterioração na latência (0,18 segundos). Ao adicionar mais clientes, réplicas sofriam com escassez de memória, tornando o sistema inoperante. Conforme relatado na literatura, embora o Netty (biblioteca utilizada na comunicação do BFT-SMaRt) ofereça boa escalabilidade, ele aumenta o consumo de recursos, podendo tornar-se um gargalo de desempenho. Em [Hammerton et al.], autores observaram um aumento no consumo de memória de até 3 vezes mais usando o Netty comparado com RMI. Com maior poder computacional ou ajustes na configuração, acredita-se que a vazão obtida com o BFT-SMaRt possa aumentar, visto que não houve degradação da latência nos experimentos. Em [Bessani et al. 2014], por exemplo, autores computam vazão superior a 90.000 comandos/s com o BFT-SMaRt.

Já com o URingPaxos obteve-se uma vazão máxima de 360.000 comandos/s e latência inferior a 0,004 segundos. Mesmo empregando todos os clientes disponíveis, não foi possível chegar ao ponto de degradação do sistema. À medida em que clientes eram adicionados, a vazão aumentava e a latência pouco alterava. Além disso, grande parte dos recursos permaneceram disponíveis, como memória, processamento e rede. O desempenho superior dessa implementação deve-se ao aproveitamento eficiente dos recursos. Ao contrário dos demais, o URingPaxos utiliza uma topologia em anel, gerando menos troca

de mensagens, otimizando a utilização da rede e evitando custos adicionais como enfileiramento em *buffers*, e emprega o Apache Thrift, diminuindo os custos com serialização e desserialização de dados. Além disso, assim como o BFT-SMaRt, suas mensagens são enviadas em lotes, diminuindo consideravelmente o número de interrupções para envio e recebimento de mensagens.

Conclusão

Com o trabalho realizado foi possível comparar o desempenho de três implementações Java para o protocolo Paxos: S-Paxos, BFT-SMaRt e URingPaxos. Os testes possibilitaram observar o ponto de degradação de apenas uma das bibliotecas: o S-Paxos. Em relação ao BFT-SMaRt e URingPaxos a carência de memória e número limitado de máquinas clientes, respectivamente, impediram a observação desses sistemas em estresse. Com trabalhos futuros pretende-se ampliar a investigação sobre a limitação de memória constatada no BFT-SMaRt, executar experimentos que possibilitem encontrar todos os pontos de saturação e observar o comportamento das diferentes bibliotecas em relação ao número de falhas toleradas. No presente trabalho foram realizados testes com apenas 3 réplicas (tolerando 1 falha).

Agradecimentos

Este trabalho foi realizado com recursos do projeto PRONEX SISCHIP2/FAPERGS (16/2551-0000 497-8).

Referências

Benz, S., Marandi, P. J., Pedone, F., and Garbinato, B. (2014). Building global and scalable systems with atomic multicast. In *Proceedings of the 15th International Middleware Conference*, pages 169–180. ACM.

Bessani, A., Sousa, J. a., and Alchieri, E. E. P. (2014). State machine replication for the masses with BFT-SMaRt. In *Proceedings of the 44th International Conference on Dependable Systems and Networks*, DSN '14, pages 355–362.

Biely, M., Milosevic, Z., Santos, N., and Schiper, A. (2012). S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 111–120. IEEE.

Burrows, M. (2006). The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA. USENIX Association.

Hammerton, M., Trevathan, J., Myers, T., and Read, W. An evaluation of software connection methods for heterogeneous sensor networks. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 7(4):550–557.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169.

Rao, J., Shekita, E. J., and Tata, S. (2011). Using paxos to build a scalable, consistent, and highly available datastore. *Proc. VLDB Endow.*, 4(4):243–254.

Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319.

Avaliação do Emprego de Algoritmos de Árvores de Decisão para a Classificação de Recursos na IoT

Felipe Camargo Gruendemann, Juan Burtet, Renato Dilli, Ana Marilza Pernas, Adenauer Yamin

[1]Laboratory of Ubiquitous and Parallel Systems – UFPEL
Pelotas, RS - Brasil

{fcgruendemann, jburtet, renato.dilli, marilza, adenauer}
@inf.ufpel.edu.br

Resumo. *A IoT é um ambiente de alta escalabilidade que disponibiliza uma quantidade crescente de recursos. Nesse cenário, a tarefa de descoberta e seleção destes recursos é um desafio em aberto. Este artigo apresenta uma alternativa para o processo de classificação dos recursos descobertos utilizando algoritmos de Árvores de Decisão. Os testes realizados com a ferramenta WEKA caracterizaram que estes algoritmos podem atingir bons níveis de acurácia.*

1. Introdução

A Internet das Coisas (IoT – *Internet of Things*) é caracterizada por ser um ambiente que apresenta uma grande quantidade de objetos, os quais efetuam conexões transitórias na Internet e resultam em um número elevado de serviços disponibilizados [Perera 2017]. Este cenário demonstra a relevância do desenvolvimento de estudos e pesquisas para cooperar com os mecanismos de seleção de recursos visando uma alta escalabilidade em ambientes de constituição heterogênea [de Andrade Junior et al. 2014].

No cenário atual existem mais de seis bilhões de objetos conectados à Internet e constantemente fornecendo serviços aos clientes, havendo a previsão de ultrapassar 100 bilhões até 2025 [BIS Research 2017]. Na medida em que as "coisas" aprovisionem uma ou mais funcionalidades, a abundância de serviços cresce. Tornando uma tarefa difícil e demorada a seleção adequada de recursos que atendam às demandas exigidas dentre uma grande quantidade de recursos.

Para realizar a seleção do melhor recurso nesta infraestrutura provida pela Internet, o processo de descoberta de recursos precisa analisar tanto os requisitos funcionais quanto os não-funcionais. Assim, após a devida identificação e localização dos recursos, é necessário classificá-los em ordem de Qualidade de Serviço (QoS).

A aprendizagem de máquina tem como objetivo entender a lógica dos programas para aprimorar a performance das máquinas em determinadas tarefas, através de experiência prévia [Suchithra and Ramakrishnan 2015]. Deste modo, as abordagens de aprendizagem revelam-se oportunas em diversos setores, principalmente na resolução de problemas relacionados a mineração de dados.

Considerando todos esses fatores, tomou-se como objetivo central deste trabalho explorar alternativas para contribuir com a arquitetura de software do mecanismo de descoberta de recursos do *middleware* EXEHDA, em desenvolvimento no grupo de pesquisa, com foco no aprendizado de máquina.

2. Classificação de Recursos Explorando Árvores de Decisão

Existem variados tipos de algoritmos de Aprendizagem de Máquina que podem ser classificados de acordo com sua forma de aprendizagem. Para o contexto da Descoberta e Classificação de Recursos, onde o problema é, baseado em um conjunto de atributos de Qualidade de Serviço (QoS), decidir qual o melhor recurso disponível classificando-os de acordo com suas características, é importante escolher um algoritmo que possua a melhor acurácia.

Como um algoritmo de aprendizagem pode ser classificado de acordo com o tipo de aprendizagem que ele realiza, uma árvore de decisão pode se encaixar nos seguintes tipos de Aprendizagem:

- Indutiva: a aprendizagem indutiva é aquela que aprende uma função ou regra geral a partir de determinado par de entrada-saída;
- Supervisionada: na aprendizagem supervisionada o agente aprende uma função que faz o mapeamento da entrada para a saída através da observação dos pares de entrada-saída dados como exemplo.
- Classificação: a aprendizagem de classificação é caracterizada por possuir um conjunto finito de saídas possíveis e essas já são conhecidas. Quando este conjunto possui apenas dois valores ele é chamado de classificação booleana ou binária.

Identificou-se então que os algoritmos de Árvore de Decisão são uma opção conveniente para a classificação de recursos. Esta situação é consequência de tais algoritmos serem adequados para problemas que envolvem a classificação baseada em decisão e aprendizado adaptativo em um conjunto de treinamento. A árvore de decisão é uma solução bem conhecida para implementar essas táticas, e é uma ferramenta de modelagem de decisão que mostra graficamente o processo de classificação de uma determinada entrada para determinados rótulos de classe de saída[Witten et al. 2016].

3. Avaliações Realizadas

A revisão bibliográfica realizada nesse trabalho apontou que o cenário da Internet das Coisas tende a crescer nos próximos anos, elevando a quantidade de recursos e serviços disponíveis. Consequentemente o processo de descoberta e classificação desses recursos enfrenta vários desafios. Considerando a perspectiva de emprego do *middleware* EXEHDA na IoT, ficou caracterizada a importância de buscar métodos eficientes para seleção de recursos para o *middleware*.

A metodologia adotada para o trabalho foi a análise de alguns algoritmos de Árvores de decisão. Os algoritmos analisados foram aplicados sobre o *dataset Quality of Web Service* (QWS) [Al-Masri and Mahmoud 2007] usando a ferramenta de *Data Mining* WEKA[1]. O objetivo da análise foi verificar a acurácia dos algoritmos na classificação de serviços considerando os valores de atributos de QoS, bem como examinar o tempo de construção de modelo de cada algoritmo selecionado. A escolha dos algoritmos de *Machine Learning* foi baseada em [Dilli et al. 2017].

Utilizando a ferramenta WEKA foram analisados alguns dos algoritmos de árvore de decisão que o software dispõe: FT, J48, LMT, RandomForest, RandomTree, SimpleCart. A classificação dos serviços foi realizada pelos algoritmos a fim de classificar o

[1]https://www.cs.waikato.ac.nz/ml/weka/

Tabela 1. Avaliação dos Algoritmos

Algoritmo	Instâncias	Tempo	Acertos	Erros	Acurácia
LMT	800	2,02s	707	93	88,37%
	900	2,29s	798	102	88,66%
	1000	2,89s	882	118	88,2%
	Acurácia média:				88,41%
Random Forest	800	0,79s	695	105	86,87%
	900	0,48s	789	111	87,66%
	1000	0,49s	896	104	89,6%
	Acurácia média:				88,04%
Random Tree	800	0,1s	634	166	79,25%
	900	0,1s	724	176	80,44%
	1000	0,1s	827	173	82,7%
	Acurácia média:				80,79%
J48	800	0,02s	654	146	81,75%
	900	0,015s	715	185	79,44%
	1000	0,02s	848	152	84,8%
	Acurácia média:				81,99%
SimpleCart	800	0,21s	631	169	78,87%
	900	0,075s	728	172	80,88%
	1000	0,07s	842	158	84,2%
	Acurácia média:				81,31%
FT	800	0,14s	693	107	86,62%
	900	0,15s	797	103	88,55%
	1000	0,19s	883	117	88,3%
	Acurácia média:				87,82%

atributo “Service Classification” dentre outros cinco atributos do conjunto de dados analisado. Para execução dos testes não foram alteradas as configurações padrão da ferramenta WEKA. E estes foram realizados sobre *datasets* de 800, 900 e 1000 recursos para cada um dos algoritmos mencionados. Os resultados dos testes podem ser vistos na tabela Avaliação dos Algoritmos.

A Tabela 1 especifica para cada algoritmo analisado: o número de instâncias presentes no *dataset*, o tempo de construção de modelo, número de acertos, número de erros e a acurácia para cada teste realizado. A Tabela 1 apresenta também a média da acurácia que cada algoritmo obteve nos testes.

Nesse sentindo pode se observar que o algoritmo com a melhor média de acurácia foi o LMT. Contudo esse foi também o algoritmo com o mais elevado tempo de construção de modelo. Outros algoritmos que obtiveram acurácia elevada foram o Random Forest e o FT, mantendo um tempo de construção de modelo bastante inferior comparado ao LMT.

Para a realização de todos os testes foi utilizada uma máquina com processador Intel Core i7-4510U CPU @ 2,00GHz e memória RAM DDR3 de 8GB.

4. Considerações Finais

A contribuição central desse trabalho foi explorar algoritmos de Árvore de Decisão, tendo como perspectiva seu emprego na Classificação de Recursos na IoT para contribuição na arquitetura de software do mecanismo de descoberta de recursos do *middleware* EXEHDA.

Os testes com os algoritmos de Árvore de Decisão aplicados à classificação de recursos apontaram uma acurácia média acima de 80%, o que por se tratar de uma pré-classificação atende as demandas do grupo referentes a otimização de desempenho do *middleware* quanto ao ranqueamento de recursos [Dilli et al. 2017]. Também ficou caracterizado que os algoritmos LMT e Random Forest foram os que apresentaram melhores resultados. Sendo que o LMT teve um maior tempo de execução e acurácia média superior se comparado ao Random Forest.

Dessa maneira é possível afirmar que Algoritmos de Árvores de decisão constituem uma alternativa a ser considerada para emprego no serviço de Classificação de Recursos do *middleware* EXEHDA.

A proposta para trabalhos futuros é realizar outros testes envolvendo conjuntos de dados maiores explorando para isto um gerador de *datasets*, e também avaliar outras alternativas de Aprendizagem de Máquina, como os *Support Vector Machines* (SVMs).

Referências

Al-Masri, E. and Mahmoud, Q. H. (2007). QoS-based discovery and ranking of Web services. In *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, pages 529–534.

BIS Research (2017). Global Sensors in Internet of Things (IoT) Devices Market, Analysis & Forecast: 2016 to 2022. Technical report.

de Andrade Junior, N. V., Bastos, D. B., and Prazeres, C. V. S. (2014). Web of Things: Automatic Publishing and Configuration of Devices. *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, pages 67–74.

Dilli, R., Filho, H. K., Pernas, A. M., and Yamin, A. (2017). EXEHDA-RR: Machine Learning and MCDA with Semantic Web in IoT Resources Classification. *WebMedia '17, October 17–20, 2017, Gramado, Brazil*.

Perera, C. (2017). Sensing as a Service for Internet of Things: A Roadmap.

Suchithra, M. and Ramakrishnan, M. (2015). A survey on different web service discovery techniques. *Indian Journal of Science and Technology*, 8(15):1–5.

Witten, I. H., Frank, E., and Hall, M. A. (2016). The WEKA Workbench. *Morgan Kaufmann, Fourth Edition*, pages 553–571.

Avaliação Experimental de Mecanismos de Tolerância a Falhas no HDFS

Iago C. Corrêa, Paulo V. M. Cardoso, Patrícia Pitthan Barcelos

Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria – UFSM

{icorrea,pcardoso,pitthan}@inf.ufsm.br

Resumo. *O uso de sistemas computacionais de alto desempenho robustos impulsionou a pesquisa sobre técnicas de tolerância a falhas. Porém, poucos trabalhos se preocupam em validar tais mecanismos, de forma a observar se os mesmos atendem aos seus objetivos. Este trabalho testa e analisa o funcionamento de técnicas de tolerância a falhas implementadas pelo sistema de arquivos do Apache Hadoop, submetendo-os a experimentos com indução de falhas.*

1. Introdução

Ferramentas voltadas para o processamento de *Big-Data* necessitam de mecanismos tolerantes a falhas, capazes de criar as condições necessárias para que a execução de aplicações ocorra de forma segura e sem comprometimento de suas informações, mesmo na ocorrência de falhas. Neste contexto, uma das ferramentas que se destaca é o Apache Hadoop [Foundation 2014], a qual possui mecanismos de *Data Replication* e *Checkpoint and Recovery*. Porém, a configuração dos atributos desses mecanismos é definida de forma estática e não pode ser modificada sem que o Hadoop e todos os seus serviços sejam interrompidos.

Visto que diferentes aplicações possuem demandas exclusivas, o processos de tolerância a falhas podem apresentar comportamentos distintos em cada cenário. Com uma metodologia de definição estática, a escolha de um atributo adequado consiste em um grande desafio que busca o equilíbrio ideal dessa escolha, a fim de oferecer confiabilidade ao sistema sem interferir no desempenho das aplicações.

Nesse sentido, o presente trabalho tem por finalidade testar e analisar os mecanismos de replicação de dados e checkpoint do sistema de arquivos do Hadoop, o HDFS (*Hadoop Distributed File System*). Serão explorados experimentos com variações nos atributos de configuração dos mecanismos utilizando-se o *Replication Factor* (RF), para a replicação, e o *Checkpoint Period* (CP) para o *checkpoint*.

O artigo está estruturado da seguinte maneira: a segunda seção aborda a estrutura da arquitetura do Hadoop, bem como alguns mecanismos de tolerância a falhas implementados. A Seção 3 explica a metodologia aplicada na fase de experimentação, enquanto a Seção 4 exibe os resultados obtidos e uma discussão sobre os mesmos. Por fim, a Seção 5 apresenta considerações finais e trabalhos futuros.

2. Apache Hadoop

O Apache Hadoop é um projeto *open source* voltado para o processamento distribuído de grandes quantidades de dados [White 2012] e que utiliza o modelo de programação *MapReduce* [Dean and Ghemawat 2008] para esse processamento. A arquitetura do Hadoop

utilizada nesse trabalho (versão 2.7.3) baseia-se na estrutura mestre-escravo composta por diversos módulos, dentre os quais se destacam: o *Hadoop Distributed File System* (HDFS), seu sistema de arquivos distribuído e o *YARN*, um *framework* para manutenção do ambiente e da aplicação. Este trabalho enfoca os mecanismos de tolerância a falhas implementados pelo HDFS [Foundation 2014].

2.1. Tolerância a Falhas no HDFS

Desenvolvido para suportar quantidades massivas de dados e oferecer um alto nível de confiabilidade, o HDFS faz a separação entre dados e metadados, organizando-os de forma distinta. A composição do sistema de arquivos é dividida em dois tipos de nós. Sendo o *NameNode* (NN) um único nó mestre, é de sua responsabilidade mapear todos os arquivos, gerir os metadados do HDFS e realizar a distribuição dos blocos dos arquivos. Os *DataNodes* (DNs) (pode haver mais de um) por sua vez, realizam o armazenamento propriamente dito dos arquivos em formato de blocos de mesmo tamanho. O HDFS implementa em sua arquitetura os seguintes mecanismos de tolerância a falhas: mensagens *Heartbeat*, replicação de blocos e *CheckPoint and Recovery* [Foundation 2016].

O HDFS utiliza-se de mensagens *Heartbeat* para manter o NN ciente do funcionamento dos DNs. Periodicamente, os DNs ativos enviam avisos ao NN a fim de atualizá-lo sobre seu estado. Caso o NN deixe de receber esses avisos de um DN por um determinado período de tempo, o NN o define como falho (*dead*), deixando de enviar instruções ao mesmo.

Com o objetivo de garantir a consistência dos dados, o HDFS realiza a replicação de blocos através do *Replication Factor* - (RF). Com isso, na possível falha de algum nó, os dados perdidos podem ser recuperados a partir dos blocos replicados em outros nós. Esse mecanismo visa assegurar que aplicações não tenham sua execução comprometida.

O *Checkpoint and Recovery* (CR) é considerada uma técnica de tolerância a falhas em que, tem-se por proposta, recuperar o sistema a um estado anterior à uma falha. Consiste em uma técnica reativa que é explorada em diversos sistemas de arquivos distribuídos que prezem pela confiabilidade dos dados e pelo bom andamento das execuções. No HDFS, periodicamente o *namespace* é replicado em um arquivo chamado `FSImage` e armazenado no sistema de arquivos do NN. Para evitar o custo de processamento necessário para reescrever o arquivo, um *log* de edições, denominado `EditLog`, também é mantido entre os arquivos do NN. Quando o processo de *checkpoint* inicia, um *merge* entre o `FSImage` e o `EditLog` é realizado, mantendo assim a cópia do *namespace* do HDFS atualizada.

3. Metodologia

A metodologia aplicada na fase de experimentação consistiu, basicamente, na execução de um *benchmark* disponibilizado pelo framework Hadoop ao mesmo tempo em que falhas foram forçadas dentro do HDFS. O *benchmark* utilizado foi o `TestDFIO`, cujo comportamento caracteristicamente *I/O bound* permite analisar as consequências de uma falha para o HDFS.

Antes da execução de cada teste, a estrutura do Hadoop era composta por 5 DNs e 1 NN, no modo pseudo-distribuído. A cada execução, o processo responsável por gerenciar um dos DNs era interrompido (através do comando `kill` do Linux) após estar,

aproximadamente, na metade do seu tempo de execução total. A ausência de mensagens *HeartBeat* por parte do DN falho, fazia com que o Hadoop esperasse um tempo X (em milissegundos) até definir o DN como falho. O processo então resultava na perda dos dados contidos no DN e na ativação dos mecanismos de tolerância a falhas do HDFS. O tempo X é calculado pela fórmula: $2*heartbeat.recheck.interval+10*heartbeat.interval = X$. Tanto o $heartbeat.recheck.interval$, quanto o $heartbeat.interval$ são configurações base do HDFS e têm valor *default* 300000 e 3, respectivamente. Porém, ambos foram ajustados para os valores 2000 e 2 para que um DN fosse marcado como falho após 20 segundos sem enviar mensagens *Heartbeat* ao NN.

Com a falha inserida, o efeito dos mecanismos de tolerância a falhas testados atuam de forma distinta no desempenho dos testes. O *checkpoint* acontece de forma periódica em um tempo previamente definido nos arquivos de configuração do HDFS. Logo, um intervalo de tempo menor implica em uma quantidade maior de *checkpoints*, resultando em um aumento no tempo de execução. O fator de replicação dos blocos também é estipulado previamente nos arquivos e é colocado em prática em paralelo às escritas dos arquivos no HDFS. Após a interrupção de um DN, o HDFS começa a replicar os blocos que não respeitam o fator de replicação estipulado previamente. Destinando recursos para realizar replicações extras, o tempo de execução da aplicação tende a aumentar, já que esse processo requer um uso extra de processamento pelo *framework*.

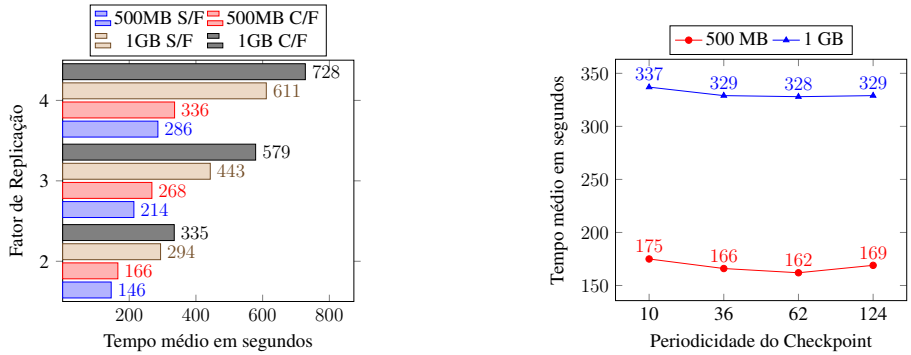
Para analisar separadamente o impacto causado pelos mecanismos de replicação e *checkpoint* do HDFS, foram elaborados dois cenários de testes. No primeiro, a periodicidade dos *checkpoints* foi fixada em 10 segundos e o fator de replicação sofreu acréscimos variando entre 2, 3 e 4. No segundo cenário de testes, o fator de replicação foi fixado em 2 e a periodicidade dos *checkpoints* variou entre 10, 36, 62 e 124 (intervalo em segundos entre um *checkpoint* e outro). Em ambos os cenários de testes, foram realizadas interrupções de um DN durante a execução do *benchmark*, simulando-se falhas permanentes.

Os experimentos foram executados em uma máquina do Laboratório de Sistemas de Computação da UFSM, com a seguinte configuração: disco de 750 GB, processador Intel Core 2 Quad 2.33GHz, duas memórias DIMM DDR3 Síncrono 1066 MHz de 2 GB, rodando sistema operacional Ubuntu 17.04, Kernel linux 4.10.0-42-generic.

4. Experimentos e Resultados

Embora o principal mecanismo de tolerância a falhas do HDFS seja a replicação de blocos, o objetivo deste trabalho é observar, de modo experimental, as consequências da alteração da periodicidade do *checkpoint* na execução das aplicações, além da variação do fator de replicação. Em ambos os cenários de testes descritos na seção anterior, foram realizadas 20 execuções do `TestDFSIO` em modo de escrita para cada configuração. Foram feitos testes com escritas de 10 arquivos de 500MB e 1GB, visando observar o comportamento das execuções. O tempo de execução foi coletado via código e, para a criação dos gráficos da Figura 1, foi utilizada a média aritmética das execuções.

O gráfico da Figura 1(a) exibe as médias dos tempos de execução referentes aos testes dos *benchmarks* com fator de replicação variável. O gráfico apresenta os valores de testes realizados com e sem a indução de falhas. Já o gráfico da Figura 1(b) representa os testes com *checkpoint* variável. Este gráfico mostra apenas medições de testes com indução de falhas.



(a) Fator de Replicação variável (b) Checkpoint variável

Figura 1. Gráficos com as medições de tempo.

Observando-se o gráfico da Figura 1(a), percebe-se que existe um aumento no tempo médio de execução dos testes com falhas em comparação com as execuções sem falhas. Esse aumento pode ter sido causado pela nova replicação dos blocos faltantes que pertenciam ao DN interrompido. Ainda, a diferença de tempo de execução entre os testes com falhas e sem falhas cresce conforme é aumentado o fator de replicação dos blocos no HDFS, já que demanda mais tempo para replicar maiores quantidades de blocos. No gráfico da Figura 1(b), observa-se que o aumento na periodicidade dos *checkpoints* gera uma pequena redução no tempo de execução das aplicações. Isto se deve ao fato de que intervalos maiores de tempo entre os *checkpoints* geram menos *merges* entre os arquivos `FSImage` e `EditLog`.

5. Considerações Finais

A definição estática de atributos para os mecanismos de tolerância a falhas implementados pelo HDFS é um grande desafio, já que escolhas inapropriadas podem gerar problemas de confiabilidade e desempenho. Além disso, a especificação estática de um período ideal é uma tarefa complexa, pois eventuais mudanças necessitam da interrupção do sistema.

Este trabalho apresentou os principais mecanismos de tolerância a falhas implementados pelo framework, avaliando, de forma experimental, o comportamento dos métodos de replicação de blocos e do checkpoint com diferentes variações. Através de dois cenários de testes, foi possível observar o impacto dos métodos no tempo de execução do *benchmark* `TestDFSIO`. Como trabalhos futuros, serão realizados testes com outros *benchmarks*, além de estender a experimentação para execução do Hadoop em modo totalmente distribuído [Foundation 2016].

Referências

Dean, J. and Ghemawat, S. (2008). *MapReduce: simplified data processing on large clusters*. Commun. ACM, 51(1):107–113.

Foundation, A. (2014). *What is the Apache Hadoop?* http://hadoop.apache.org/index.pdf.

Foundation, A. (2016). *Hadoop: Cluster Setup*. https://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-common/ClusterSetup.html.

White, T. (2012). *Hadoop: The definitive guide*. "O'Reilly Media, Inc.".

Avaliando o desempenho do PyTorch sobre GPUs embarcadas

**Angelo Nery Vieira Crestani, Paulo Silas Severo de Souza,
Wagner dos Santos Marques, Marcos Paulo Konzen, Fábio Diniz Rossi**

[1]Instituto Federal de Educação, Ciência e Tecnologia Farroupilha (IFFar)
Campus Alegrete – RS 377, KM 27 – Alegrete – RS – Brasil

angelo.vcrestani@gmail.com, {paulo.souza,wagner.marques}@email.com

{marcos.konzen,fabio.rossi}@iffarroupilha.edu.br

Resumo. *A aprendizagem de máquina emergiu com a necessidade de atender a uma demanda crescente por processamento de grandes volumes de dados de maneira rápida e precisa. Esta abordagem consiste no reconhecimento de padrões, a fim de treinar modelos sobre dados com características específicas para a realização de predições sobre novos dados com as mesmas características. Várias bibliotecas foram propostas a fim de suprir a necessidade de desempenho. Assim, esse trabalho apresenta uma avaliação da biblioteca de programação PyTorch em GPUs embarcadas. Os resultados indicam que a utilização das GPUs proporciona desempenho de 87,6 vezes melhor quando comparado com execução sobre CPUs.*

1. Introdução

Aprendizagem de máquina (*machine learning*) [Condie et al. 2013] consiste no ramo da computação que visa reconhecimento de padrões através do treino de modelos sobre dados com certas características, e depois deste modelo treinado, infere predições ou classificações sobre novos dados de mesma característica. O crescente volume, complexidade e variedade de dados associado ao aumento no poder computacional com baixo custo, fez com que técnicas de aprendizagem de máquina ganhem um novo impulso no que tange à análise desses dados de forma rápida, precisa e automática. Uma das técnicas de aprendizagem de máquina que está fortemente ligada a análise de grandes volumes de dados é a aprendizagem profunda (*deep learning*) [Deng and Yu 2014]. Essa técnica é baseada em redes neurais convolucionais, ou seja, apresenta redes neurais multicamadas inspiradas em modelos biológicos do cortex visual, onde neurônios corticais individuais respondem a estímulos de regiões restritas representando sub-regiões da visão. Isso pode melhorar o desempenho das aplicações pois permite um melhor balanceamento da quantidade de filtros por estágio e por profundidade. Porém, isso exige grande poder de processamento.

A fim de sanar este problema, bibliotecas de programação foram desenvolvidas com o objetivo de atender a demanda por processamento e análise de grandes volumes de dados via aprendizagem de máquina, e ao mesmo tempo executar tal processamento sobre arquiteturas computacionais atuais que oferecem grandes quantidades de processadores gráficos (*GPUs - Graphic Processor Units*) [Harris 2008]. A biblioteca avaliada por este trabalho é o PyTorch[1], que permite características de alto nível, tais como computação de

[1]http://pytorch.org/

tensores e redes neurais profundas. As métricas avaliadas foram: desempenho, consumo de energia e EDP (*Energy-Delay Product*) [Blem et al. 2013], que avalia o equilíbrio entre desempenho e economia de energia). O artigo está organizado da seguinte maneira: a Seção 2 apresenta o ambiente de testes, *benchmarks* e parâmetros avaliados; a Seção 3 apresenta e discute os resultados obtidos; finalizando na Seção 4 com nossas conclusões e trabalhos futuros.

2. Ambiente de Execução

A Figura 1 apresenta o testbed utilizado neste trabalho - NVidia Jetson TX2 - um *System On Chip* (SoC) que combina os processadores NVidia Denver 2 *dual-core* 2.0-GHz e ARM Cortex-A57 2.0-GHz *quad-core*, além de uma GPU NVIDIA Pascal integrada de 256 núcleos. O SoC provê uma memória DRAM de 8GB. Além disso, o SoC provê leitura de métricas de leituras digitais da potência, tensão e corrente via um sensor de corrente INA226. O sistema operacional presente no SoC, trata-se do Ubuntu 16.04 LTS (kernel 4.4.15-tegra). A medição da corrente de energia em miliamperes foi obtida através do circuito integrado INA 226 (sendo que esses valores foram posteriormente convertidos para Joule), e os resultados são médias de 10 execuções.

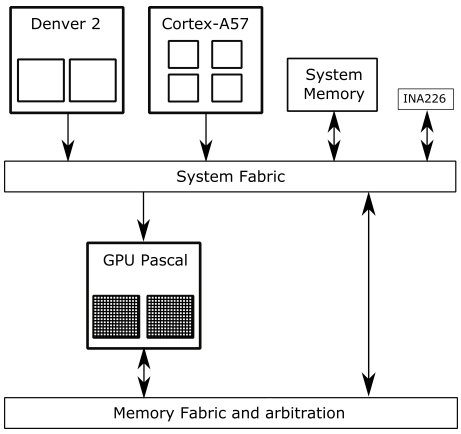


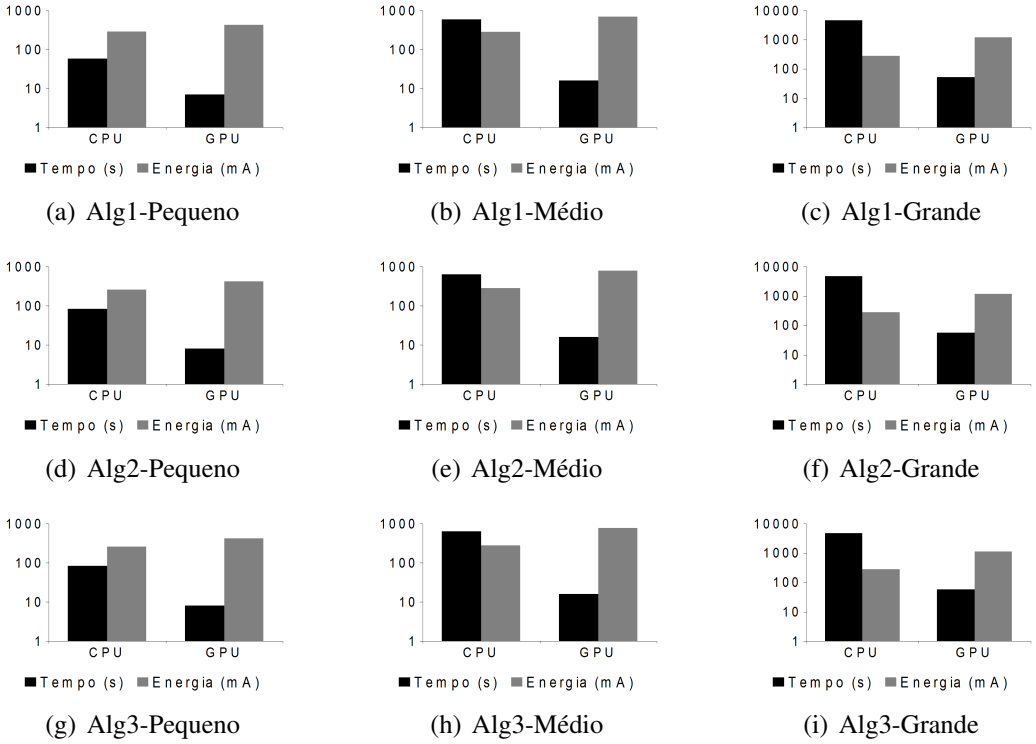
Figura 1. Arquitetura da NVidia Jetson TX2

As aplicações consideradas nesta avaliação foram: *Alg1 - PyTorch Tensors*: é uma implementeção do algoritmo *PyTorch* que segundo [Paszke et al. 2017] caracteriza-se pela utilização de tensores sendo esses generalizações de uma matriz que pode ser indexada em n-dimensões. A utilização deste algotitmo provém ao PyTorch possibilidades como a utilização de GPUs para a aceleração de cálculos numéricos atravéz da definição de um novo tipo de dados. *Alg2 - PyTorch: Variables and autograd*: fornece classes e funções que implementam a diferenciação automática de funções com valores escalares arbitrários para todas as operações em tensores. A diferenciação automática pode ser usada para automatizar a computação de caminhos em redes neurais com o uso de variáveis, podendo definir redes complexas que podem ser executadas em GPUs [Paszke et al. 2017]. *Alg3 - PyTorch* com funções adicionais: permite definir novas funções com subclasses do *torch.autograd*, modificando as funções para realizar operações específicas como, por exemplo, implementar uma função de ativação chamada *Rectified Linear Units* (ReLU), de modo que a rede neural processe os dados com base nessa função de ativação.

A Tabela 1 apresenta os três tamanhos de entrada utilizados nos testes, além de seus respectivos parâmetros: tamanho do lote (N), dimensão de *input* (D_in), dimensão da camada oculta (H) e dimensão de *output* (D_out).

Tabela 1. Entradas de dados utilizadas nos 3 algoritmos.

Tamanho	N	D_in	H	D_out
Pequeno	64	15000	100	150
Médio	192	20000	300	200
Grande	384	40000	600	400



(a) Alg1-Pequeno (b) Alg1-Médio (c) Alg1-Grande

(d) Alg2-Pequeno (e) Alg2-Médio (f) Alg2-Grande

(g) Alg3-Pequeno (h) Alg3-Médio (i) Alg3-Grande

Figura 2. Diferentes tamanhos de entrada com 500 interações.

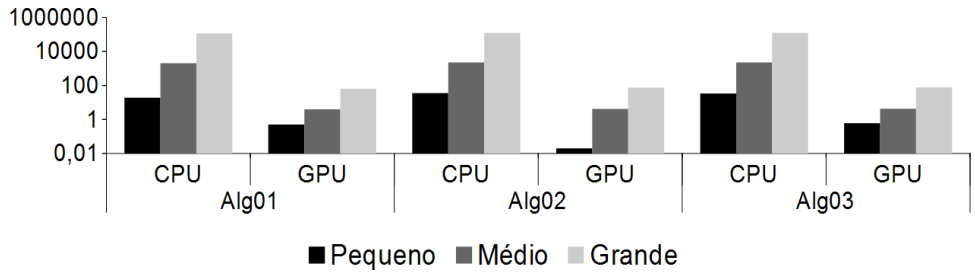


Figura 3. Cálculo do EDP.

3. Resultados

Os resultados indicam que a utilização da GPU gera um aumento significativo no desempenho das aplicações. Todas as implementações, mesmo com diferentes volumes

de dados, apresentaram aumento de desempenho, tendo o primeiro algoritmo com grande carga de trabalho (Figura 2(c)) obtido o resultado mais significativo, com melhora de 87,6 vezes no seu tempo de execução. Os resultados do consumo de energia e do desempenho das demais aplicações podem ser observados junto à Figura 2. O aumento de desempenho obtido através da utilização da GPU ocorre pelo fato de que o PyTorch integra bibliotecas da NVidia que foram desenvolvidas a fim de maximizar o desempenho deste framework com diferentes cargas de trabalho. É possível observar também que as aplicações apresentaram melhora de performance mediante o aumento da carga de trabalho. Isto ocorre devido ao grande número de cores disponíveis pelo embarcado adotado.

Nesse sentido, considerando o acréscimo no desempenho das aplicações e o aumento no consumo de energia, o cálculo do EDP (dado pelo produto do consumo de energia em Joules e o tempo de execução) foi aplicado com o intuito de verificar o equilíbrio entre o consumo de energia e desempenho. Para todos os algoritmos testados, existe melhoria expressiva no EDP em relação à execução apenas sobre CPUs (Figura 3). Os resultados coletados indicam que a utilização da GPU é uma alternativa viável para execução deste framework. Apesar da sua utilização ocasionar o acréscimo do consumo de energia do embarcado, a sua adoção gera aumento relevante no desempenho das aplicações. Assim, mesmo com o aumento no consumo de energia, o dispositivo executa suas tarefas de maneira mais eficiente, o que ocasiona os resultados positivos em EDP.

4. Conclusão e Trabalhos Futuros

A aprendizagem de máquina emergiu com a crescente necessidade de análise de dados de maneira rápida e precisa. Logo, diversas bibliotecas de programação estão sendo desenvolvidas para atender tal demanda. Nesse sentido, este trabalho realizou uma análise de desempenho da biblioteca PyTorch sobre GPUs embarcadas. Os resultados indicam ganho de até 87,6 vezes na utilização da GPU em relação à CPU. Assim, GPUs apresentam-se como uma alternativa viável para a execução da referida biblioteca. Como trabalhos futuros, almeja-se a avaliação das bibliotecas TensorFlow e Theano.

Referências

Blem, E., Menon, J., and Sankaralingam, K. (2013). Power struggles: Revisiting the RISC vs. CISC debate on contemporary arm and x86 architectures. In *Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, pages 1–12.

Condie, T., Mineiro, P., Polyzotis, N., and Weimer, M. (2013). Machine learning for big data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 939–942, New York, NY, USA. ACM.

Deng, L. and Yu, D. (2014). Deep learning: Methods and applications. *Found. Trends Signal Process.*, 7(3–4):197–387.

Harris, M. (2008). Many-core gpu computing with nvidia cuda. In *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, pages 1–1, New York, NY, USA. ACM.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Avaliando o Uso de Fog-Computing no Servidor de Borda do Middleware EXEHDA

Leonardo Silveira João[1]*; Huberto Kaiser [1]; Maurício Pilla[1]; Renato Dilli[1]
Ana Pernas[1]; Verônica Tabim[2]; Adenauer Yamin[12]

[1]LUPS – Laboratory of Ubiquitous and Parallel Systems - UFPEL
[2]G3PD - Grupo de Pesquisa em Processamento Paralelo e Distribuído - UCPEL

{ldrsjoao,hkaiser,pilla,renato.dilli,marilza,adenauer}@inf.ufpel.edu.br,
veronica.tabim@sou.ucpel.edu.br

Resumo. *Este trabalho tem como objetivo central a avaliação da arquitetura do Servidor de Borda do middleware EXEHDA concebida para aquisição e processamento distribuído de dados contextuais na IoT, materializando assim a abordagem de Fog Computing no EXEHDA. As avaliações realizadas apresentaram resultados promissores e apontam para continuidade dos esforços de estudo e pesquisa no tema.*

1. Introdução

A qualificação dos sistemas distribuídos, associada ao fato das tecnologias modernas viabilizarem o surgimento de soluções embarcadas de custo baixo e pequeno consumo energético, criou um cenário oportuno para a Computação Ubíqua (UbiComp).

A Internet das Coisas (*Internet of Things* – IoT) com sua natureza distribuída, constitui uma infraestrutura oportuna para o cenário típico das aplicações ubíquas. A IoT tem como característica central a interoperação entre objetos, pessoas, animais, etc., através da troca de dados e comandos, onde cada um possui um dispositivo embarcado, dotado de um identificador único, utilizando a Internet o meio de comunicação [Perera et al. 2013].

Diferentes desafios surgem no provimento de suporte para as aplicações direcionadas a IoT, dentre estes temos o gerenciamento das informações coletadas através de dispositivos heterogêneos, e a interpretação destas informações considerando o contexto de interesse das aplicações. As informações coletadas no sentido de promover o entendimento do contexto de interesse das aplicações, denominam-se neste trabalho de dados contextuais.

Neste sentido, esse trabalho tem o objetivo de avaliar a abordagem de Fog Computing desenvolvida para o Servidor de Borda do Middleware EXEHDA, responsável pela aquisição de dados contextuais, na perspectiva dos custos de comunicação envolvidos.

Na perspectiva da Fog Computing, os equipamentos necessitam possuir suporte para lidar com as demandas de processamento contextual tanto locais, como aquelas introduzidas por outros equipamentos que constituem a infraestrutura de *Fog Computing*. As características essenciais da *Fog Computing* [Bonomi et al. 2012] [Hong et al. 2013]

*Bolsista PIBIC/FAPERGS

são: (i) proximidade com a borda, (ii) organização hierárquica, (iii) consciência de localização, (iv) distribuição geográfica densa, (v) escalabilidade, (vi) suporte a mobilidade, e, (vii) aplicações em tempo real.

2. Servidor de Borda do EXEHDA: Avaliação da Operação em Fog Computing

A *Fog Computing* é implementada nas bordas computacionais do EXEHDA, transferindo parte do processamento para os dispositivos localizados no ambiente do usuário final, sendo assim considerada uma nuvem computacional mais próxima da ocorrência dos eventos de interesse.

Uma discussão da arquitetura proposta para o Servidor de Borda do EXEHDA, e a discussão das suas funcionalidades de Fog Computing pode ser encontrada em [João 2017].

Este artigo avalia a arquitetura proposta em [João 2017] empregando um estudo de caso direcionado a Viticultura de Precisão (VP), avaliando as possíveis contribuições decorrentes da utilização da *Fog Computing*. A viticultura é uma frente de exploração agrícola que possui grande potencial econômico e social em diversas regiões do Brasil, e em particular do Rio Grande do Sul. A Viticultura de Precisão (VP) pode ser entendida como a gestão da variabilidade temporal e espacial das áreas cultivadas com o objetivo de melhorar o rendimento econômico da atividade agrícola [Braga and PINTO 2009].

Nesse sentido, uma das questões fundamentais na produção de vinhos de alta qualidade é o momento certo de irrigar. Para determinar o momento correto de irrigar é necessário avaliar as variáveis físicas do ambiente. No cenário em estudo, as variáveis contextuais consideradas são as seguintes: tensão de água no solo, temperatura e umidade do ar. Os comandos de atuação disparados através do processamento de regras, são os seguintes: alertas visuais e sonoros, envio de mensagens (SMS/e-mail) e atuação de transdutores elétricos para os acionamentos dos sistemas de irrigação.

A Figura 1 representa um vinhedo típico da região sul do Brasil com suas respectivas zonas de manejo. Visando uma qualificação da informação e validação de consistência, foi considerado o uso de diversos sensores distribuídos ao longo das zonas de manejo. Cada zona de manejo é equipada com um Servidor de Borda capaz de gerenciar os gateways e sensores atribuídos a estes, para que o controle de irrigação não seja inviabilizado por eventuais quedas de conexão de rede perda de contato com o Servidor de Contexto.

Figura 1. Zonas de manejo em um parreiral
Fonte: O Autor

Através desse cenário pretende-se avaliar à capacidade de operação em *Fog Computing*. Atualmente, grande parte das áreas rurais dependem de conexões móveis através de rede de celular para conexão à Internet, muitas vezes sujeitas à conexões instáveis, baixa largura de banda e volume de dados mensais limitados através de franquias. Esses aspectos apontam para a necessidade de uso moderado do tráfego de dados através da Internet e o uso de estratégias que garantam a operação em momentos em que esse acesso não é possível.

3. Resultados e Discussões

O ambiente computacional concebido para avaliação do cenário é baseado no *Common Open Research Emulator*(CORE). O CORE[1] é um framework *open-source* que visa a emulação de ambientes computacionais, permitindo testes em ambientes computacionais largamente distribuídos sem a necessidade de implantações de custosos hardwares reais. A seguir as avaliações feitas no cenário de testes:

- **Operação com FOG:** realiza uma operação de filtragem e agregação de dados contextuais, publicando a média dos valores coletados, em períodos de 1 hora. Além de qualificar o controle do ambiente, as operações de publicação dos eventos e média da tensão do solo para o Servidor de Contexto são antecipadas, de maneira a preservar o histórico e o registro de que um evento ocorreu e que foi necessária uma atuação sob o ambiente.
- **Operação sem FOG:** publica os dados dos sensores quando a medição é realizada e a média é realizada no Servidor de Contexto, nesse caso também não há a noção de eventos, ou seja, não há históricos de que houve ou não irrigação, pois nesse caso não há suporte a registro de eventos originados de regras;

Foram realizadas duas avaliações: (i) aquisições regulares, e (ii) leituras em intervalos reduzidos cuja descrição está a seguir.

A primeira avaliação realizada, **aquisições regulares**, foi considerado a aquisição de dados contextuais dos sensores de tensão do solo em intervalos de 5 minutos. A Tabela 1, apresenta os resultados encontrados, onde foram agrupados em quatro intervalos de tempo: hora, dia, semana e mês. É possível analisar o volume de dados trafegado, considerando o emprego da *Fog Computing* para os diferentes intervalos de tempo previstos.

Tabela 1. Período de Acúmulo de Dados

	Sem Fog	**Com Fog**
1 Horas	0,044 MBytes	0,001 MBytes
1 Dia	1,066 MBytes	0,02 MBytes
7 Dias	7,462 MBytes	0,14 MBytes
30 Dias	31,968 MBytes	0,566 MBytes

Conforme o aumento do período observado com o emprego da *Fog Computing*, a economia de recursos se mostra mais presente. Isso ocorre devido a comunicação do Servidor de Borda e os controladores de atuação, não necessitarem de intervenção do Servidor de Contexto no ambiente em questão.

[1]https://www.nrl.navy.mil/itd/ncs/products/core

A segunda avaliação realizada, **leituras em intervalos reduzido**, aumentou a frequência de aquisição das grandezas físicas do ambiente, seja porque há uma necessidade de manter um rigoroso controle sobre o ambiente, ou para manter um registro das possíveis variações das informações contextuais. Utilizando intervalos pequenos para a aquisição de informações contextuais, pode ocasionar problemas operacionais em arquiteturas que trabalham com o conceito de inteligência centralizada.

Na Tabela 2 é apresentado a comparação ao utilizar um estratégia de *Fog Computing* quando a necessidade de verificação mais frequente às variáveis contextuais. No cenário, utilizou-se dados de 30 dias, onde no ambiente com o emprego de *Fog Computing* são realizadas publicações para fins históricos a cada hora, enquanto no ambiente sem *Fog Computing* as publicações são realizadas de acordo com a coleta dos dados.

Tabela 2. Intervalo de Aquisição de Dados Contextuais

	Sem Fog	**Com Fog**
1 Minuto	159 MBytes	0,556 MBytes
5 Minutos	31,968 MBytes	0,556 MBytes
10 Minutos	15,984 MBytes	0,556 MBytes

4. Conclusão e Trabalhos Futuros

Avaliações feitas com o estudo de caso exploraram as funcionalidades em relação a gerência dinâmica para aquisição de dados contextuais e seu processamento distribuído. Os resultados obtidos se mostraram oportunos na redução de volume de dados transmitidos ao Servidor de Contexto, promovendo a ciência de contexto empregando as bordas computacionais de maneira distribuída.

Como trabalhos futuros, destacaríamos a implantação do SB-FOG na EMBRAPA Clima Temperado, substituindo a antiga versão do Servidor de Borda que somente interopera com o Servidor de Contexto, e por fim disponibilizar publicamente a API do SB-FOG, facultando o emprego da mesma por terceiros.

Referências

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

Braga, R. and PINTO, P. (2009). Alterações climáticas e agricultura. *Inovação e Tecnologia na Formação Agrícola*, 12(2):34–56.

Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., and Koldehofe, B. (2013). Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM.

João, L. R. S. (2017). SB-FOG: Uma Proposta para o Servidor de Borda do Middleware EXEHDA Direcionada à Fog Computing. Master's thesis, Universidade Federal de Pelotas, Brasil, Pelotas.

Perera, C., Zaslavsky, A. B., Christen, P., and Georgakopoulos, D. (2013). Context aware computing for the internet of things: A survey. *CoRR*, abs/1305.0982.

Avaliando Transacional Boosting para Haskell

Jonathas A. O. Conceição[1], **André R. Du Bois**[1]

[1]Universidade Federal do Rio Grande do Sul (UFPel)
Pelotas – RS – Brazil

{jadoliveira,dubois}@inf.ufpel.edu.br

Resumo. *Transactional Boosting oferece uma maneira de transformar ações linearmente concorrentes em ações transacionalmente concorrentes, possibilitando assim sua composição com outras ações transacionais. Este trabalho disponibiliza uma extensão do STM Haskell para a aplicação de trasacional boosting às funções, possibilitando que ações linearmente concorrentes de alto desempenho sejam adicionadas aos seus blocos transacionais.*

1. Introdução

Memórias Transacionais, do inglês *Software Transacional Memory* (STM), é uma alternativa de alto nível à sincronização por *locks*. Nas Memórias Transacionais, todo acesso à memória compartilhada é agrupado como transações que podem executar de maneira concorrente. Se não houve conflito ao acesso da memória compartilhada ao fim da transação um *commit* é feito, tornando assim o conteúdo da memória público para o sistema. Caso ocorra algum conflito um *abort* é executado em todo bloco descartando qualquer alteração à memória. Diferente da sincronização por *locks*, transações podem ser facilmente compostas e são livres de *deadlocks* [Harris et al. 2008].

Os conflitos ocorrem quando duas ou mais ações de escrita ou leitura são feitas no mesmo endereço de memória. Entretanto essa forma de detecção conflitos pode gerar falsos conflitos levando a uma perda de desempenho. Por exemplo onde há duas transações diferentes modificando partes diferentes de uma lista encadeada [Herlihy and Koskinen 2008]. Embora essas ações não conflitem, o sistema detecta um problema já que uma transação modifica memória que outra transação lê, esse tipo de detecção de conflito pode gerar grande perda de performance quando se utiliza certos tipos de estruturas encadeadas. Por outro lado, utilizando sincronização por *locks* ou algoritmos *lock-free*, pode-se alcançar um alto nível de concorrência ao custo de complexidade no código. *Transactional boosting* oferece uma maneira de compor ações transacionais com estruturas de dados concorrentes oferecendo assim uma solução para esses falsos conflitos do STM.

STM Haskell é uma biblioteca adicionada ao *Glasgow Haskell Compiler* (GHC) em 2008 e provê primitivas para o uso de memórias transacionais em Haskell. Utilizando uma biblioteca de memórias transacionais própria, toda escrita em Haskell, uma implementação de *Transactional Boosting* para STM Haskell foi feita para testar as vantagens de desempenho dessa técnica em uma linguagem puramente funcional [Du Bois et al. 2014]. Devido aos resultados promissores apresentados por esse estudo, o objetivo do corrente trabalho é disponibilizar uma primitiva de alto nível na implementação padrão do STM Haskell, permitindo assim a aplicação de *Transactional Boosting* de maneira nativa no compilador GHC.

2. Transactional Boosting

Transactional Boosting é uma técnica utilizada para transformar objetos linearmente concorrentes em objetos transacionalmente concorrentes [Herlihy and Koskinen 2008], permitindo assim sua utilização dentro dos blocos atômicos do STM. Nessa técnica, os objetos são tratados como caixas-pretas e ambos conflitos e registros à memória são tratados logo que são encontrados. O processo de adicionar *transactional boosting* ao compilador GHC envolve duas partes: a criação de uma interface Haskell em alto nível, que provê uma primitiva que possa ser usada pelo programador para realizar o *boost* de uma função Haskell, e uma camada principal no *RunTime System* (RTS), no baixo nível do GHC, escrita em C.

A interface de alto nível é uma primitiva que permite aplicar o *Transactional Boosting* a uma função linearmente concorrente. Para que seja criado uma versão *boosted* da função, é necessário que essa tenha um inverso, assim o sistema STM terá os mecanismos necessários em caso de *commit* e *abort*. A primitiva então envolve a função numa ação STM permitindo sua chamada dentro do bloco transacional. A primitiva de *boost* tem o seguinte protótipo e argumentos:

```
boost :: IO(Maybe a) -> (a -> IO ()) -> IO () -> STM a
```

Os argumentos são (1) uma ação (do tipo **IO**(**Maybe** a)), i.e., a função original que vai ser executada; (2) Uma ação de desfazer (do tipo a $->$ **IO**()), usada para reverter a ação executada em caso de *abort*; (3) Um *commit* (do tipo **IO**()), que é usado para tornar público a ação feita pela versão *boosted* da função original.

O RTS pode ser visto como três grandes subsistemas: Armazenamento, responsável pelo *layout* de memória e *garbage collector*; Execução, como o código Haskell é executado; O Escalonador, que gerencia threads e dá suporte *multicore*. As implementações desse trabalho aconteceram principalmente nas partes de Armazenamento e Execução. Quanto ao Armazenamento temos os *Heap Objects*, um aspecto central do armazenamento das funções em execução do RTS. Estas estruturas de dados seguem o *layout* da Figura 1. A primeira parte desses objetos é chamado de *Info Pointer*, que aponta para o *entry code* da estrutura; A segunda é o *Payload* onde ficam os dados carregados pela estrutura; O *entry code* é um código estático responsável por avaliar o *Heap Object*. Imediatamente antes do *entry code* encontra-se a *Info Table* do objeto, que contém informação utilizada pelo RTS. Na parte de Execução temos as chamadas *Primitive Operations*, estas são operações que por alguma impossibilidade ou por uma questão de desempenho são implementadas diretamente no RTS, e este é o caso da maioria das funções do STM.

Para dar suporte ao *boost* um novo *Heap Object* foi criado para armazenar a função a ser executada, o *StgBoostSTMFrame*. A estrutura adicionada possui referências em seus campos para as funções passadas para lidar com o *abort* e *commit*. A primitiva então irá executar a ação passada, armazenar as referências às funções auxiliares numa instância do *StgBoostSTMFrame* e por fim associa-lo à transação atual. Em caso de *commit* ou *abort* do bloco as funções associadas à primitiva de *boosting* são executadas.

3. Exemplo

Tomemos como exemplo para a aplicação de *Transacional Boosting* uma estrutura de conjuntos, como em [Herlihy and Koskinen 2008]. Uma implementação de conjuntos

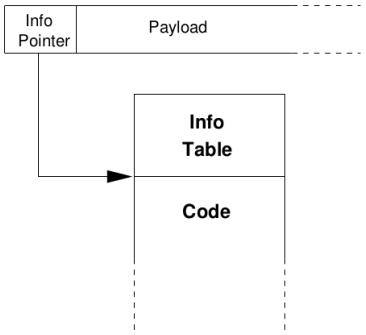


Figura 1. *Layout* de um *Heap Object* [Marlow and Peyton Jones 1998].

normalmente oferece três funções, *add*, *remove* e *contains*. Para a versão *boosted* de conjunto apresentada aqui foi utilizada uma lista encadeada *thread safe*. Na implementação é importante garantir que se uma transação está trabalhando num elemento, nenhuma outra transação vai utilizar o mesmo elemento. Isso pode ser alcançado utilizando uma tabela hash para associar um *lock* para cada elemento do conjunto. Vários modelos de tabela hash *thread safe* em Haskell foram apresentados em [Duarte et al. 2016], deles o algoritmo de *lock fino* foi utilizado para nossa implementação.

Para adicionar um elemento ao conjunto devemos adquirir o *lock* associado ao elemento e então inseri-lo na lista encadeada. Como a lista pode conter elementos duplicados é necessário verificar se o elemento já não está contido antes de inseri-lo. Se um elemento foi inserido e a transação abortar, o elemento deve ser removido e o *lock* liberado. Caso a transação termine sem conflitos é necessário apenas liberar o *lock*. Para remover um elemento é preciso adquirir o *lock* associado e então deletar o elemento da lista. Para reverter um *remove* o elemento deve ser devolvido ao conjunto e o *lock* liberado. O *commit* assim como antes precisa apenas liberar o *lock*. Por fim o *contains* precisa apenas realizar o *lock* no elemento e então conferir se ele está na lista. Tanto para o *commit* como para o *abort* o *contains* precisa apenas liberar o *lock* adquirido.

4. Experimentos e Resultados

Os experimentos foram executados numa máquina com processador Intel Core i7, frequência de 3.40GHz, 4 cores físicos e 4 lógicos, 8GiB de memória RAM. O sistema operacional foi um Ubuntu 14.04, a versão compilada do GHC foi a 7.10.3.

Para testar a biblioteca de conjuntos implementada usando *transactional boosting* foi utilizado uma lista de 2000 operações geradas aleatoriamente, bem como um conjunto inicial de 2000 elementos. Utilizando o mesmo número de threads e *cores* cada *thread* recebia uma lista de 2000 operações. Dois tipos de listas eram gerados em execuções separadas: Lista de leitura, contendo 40% de operações de `add` e `remove` mais 60% de operações de `contains`; Uma lista de escrita contendo 75% de operações de `add` e `remove` mais 25% de `contains`, permitindo uma visualização dos falsos conflitos na variação do desempenho. No gráfico é apresentado com tempo em escala logarítmica as médias de 30 execuções para os dados números de cores e threads. Pela Figura 2, pode-se observar que a utilização do *Transactional Boosting* resulta numa estrutura de conjuntos de desempenho bem superior em comparação à alternativa feita puramente com o STM Haskell.

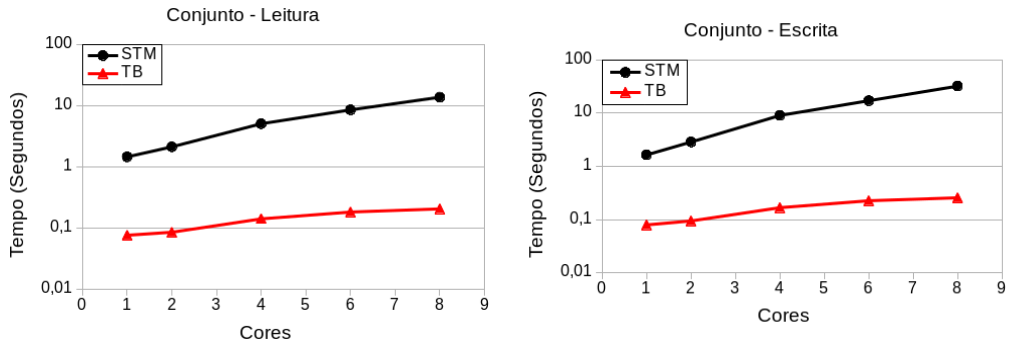


Figura 2. Tempo de execução de 2000 operações para cada core (A esquerda: 40% add e remove + 60% contains; A direita: 75% add e remove + 25% contains).

5. Conclusões e Trabalhos Futuros

A primitiva desenvolvida neste estudo, oferece uma maneira direta de compor objetos linearmente concorrentes com objetos transacionalmente concorrentes, de uma maneira que independe de qualquer tratamento do STM. *Transactional Boosting* é uma alternativa de baixo nível ao controle de concorrência e requer que o programador preserve as propriedades da estrutura transacional, além disso seu mau uso pode resultar em problemas como *deadlock*s. Entretanto se utilizada por programadores experientes pode ser usada para desenvolver bibliotecas concorrentes de alto desempenho capazes de serem adicionadas aos blocos transacionais oferecendo uma solução ao problema de falsos conflitos.

Para trabalhos futuros visamos um estudo da Semântica Formal e uma análise categórica da primitiva para uma melhor formalização da sua utilização em Haskell e a implementação de casos de teste mais robustos.

Referências

Du Bois, A., Pilla, M., and Duarte, R. (2014). Transactional boosting for haskell. In Quintão Pereira, F., editor, *Programming Languages*, volume 8771 of *Lecture Notes in Computer Science*. Springer International Publishing.

Duarte, R. M., Du Bois, A. R., Pilla, M. L., and Reiser, R. H. S. (2016). Comparando o desempenho de implementações de tabelas hash concorrentes em haskell. *Revista de Informática Teórica e Aplicada*, 23(2).

Harris, T., Marlow, S., Jones, S. P., and Herlihy, M. (2008). Composable memory transactions. *Commun. ACM*, 51(8).

Herlihy, M. and Koskinen, E. (2008). Transactional boosting: A methodology for highly-concurrent transactional objects. In *Proceedings of the 13th Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA. ACM.

Marlow, S. and Peyton Jones, S. (1998). The new ghc/hugs runtime system.

Comparação de Desempenho do Workload YCSB em Raspberry PI B+ e 3

Guilherme Souza S.[1], **João Vitor V. T. de Oliveira** [1], **Mauricio L. Pilla**[1]

[1] Universidade Federal de Pelotas (UFPEL)
Caixa Postal 354 – 96010-610 – Pelotas – RS – Brazil

{gdsds, jvvtdoliveira, pilla}@inf.ufpel.edu.br

Resumo. *Atualmente 1,4% da energia mundial produzida é consumida por datacenters. Cujas máquinas possuem uma composição da hardwares convencionais com um alto consumo de energia. Existem, porém, dispositivos de arquitetura ARM cuja característica principal é o baixo consumo e, assim como os dispositivos convencionais, também atendem as demandas computacionais de uma nuvem. Portanto, esse trabalho tem como objetivo comparar a eficiência energética entre dois dispositivos de arquitetura ARM, Raspberry PI B+ e PI 3.*

1. Introdução

A computação em nuvem é um modelo de negócios que disponibiliza o acesso a uma gama de recursos computacionais de maneira ubíqua, conveniente e sob demanda [Mell and Grance 2011] tendo sua estrutura física composta de máquinas convencionais com alto consumo de energia, situadas dentro de um *datacenter*. Portanto, o consumo de energia é uma das principais preocupações dos provedores de sistemas de nuvem, tendo em vista que o custo tanto financeiro quanto ambiental de manutenção da nuvem reflete nos preços oferecidos pelo provedor, gerando repercussão em todas as fases do planejamento do *datacenter* da nuvem. Esse consumo anual acumulado dos *datacenters* ao redor do mundo é estimado em 26 GW, correspondendo a cerca de 1,4% da energia mundial consumida [A. Uchechukwu and Shen 2014].

Com base nisso, em [Oliveira and Ataides 2017] foi desenvolvido um comparativo entre uma máquina de *hardware* convencional e um dispositivo de arquitetura ARM com baixo consumo de energia, a fim de averiguar a viabilidade de se trocar parte das máquinas de um datacenter por dispositivos especializados em baixo consumo de energia, respeitando sempre o *service-level-agreement*. Em sequência, neste artigo propõe-se a comparação entre dispositivo utilizado no trabalho anterior (Raspberry Pi Model B) e sua versão atualizada (Raspberry Pi 3 Model B).

Foram utilizadas as métricas de vazão de operações (medida em operações por segundo), consumo energético (medido em Watts) e operações poPr unidade de consumo (medido em operações por segundo por watt) medidas pelo *benchmark* Yahoo Cloud Serving Benchmark (YCSB) para a comparação entre os dois dispositivos. A finalidade deste trabalho é determinar a diferença entre os dois dispositivos no quesito poder de processamento, determinando qual se encaixa melhor na posição de nó de baixo consumo de uma nuvem *OpenStack*.

O presente artigo está organizado da seguinte forma: na Seção 2 são apresentados os dispositivos utilizados e suas características físicas, bem como o *benchmark* e a me-

todologia dos testes. Na Seção 3, são expostos os resultados obtidos após a execução do *benchmark*. Concluindo-se o trabalho na Seção 4

2. Metodologia

Nesta seção aborda-se a metodologia aplicada na realização do trabalho, o *hardware* utilizado na comparação, o *benchmark* escolhido para medição e o método de análise estatística dos resultados.

2.1. Hardware Utilizado

Os computadores Raspberry PI foram originalmente concebidos para inspirar jovens programadores a aprimorar seu talento em codificação, ganhando um lugar no curso de informática na Universidade de Cambridge [Pi 2013]. Esse dispositivo pode ser adquirido por um valor razoavelmente baixo e possui utilização em automações de diversos meios. As caracteríscas de *hardware* dos dispositivos são apresentadas na Tabela 1.

Tabela 1. Especifícações do *hardware* dos dispositivos, segundo o fabricante [Fundation 2015].

Especificação	Raspberry PI B+	Raspberry PI 3
Processador	BCM2835 Single Core 700MHz	BCM2837 64Bit Quad Core 1.2GHz
Arquitetura	ARM11	ARMv8
RAM	512MB SDRAM 400MHz	1GB SDRAM 400MHz
Amazenamento	MicroSD	MicroSD
USB 2.0	4 Portas USB	4 Portas USB
Máxima Corrente/ Tensão	700mAh / 5V	2,4A / 5V
GPIO	40 pins	40 pins

2.2. Benchmark

O *benchmark* escolhido para a utilização nesse trabalho foi o *Yahoo! Cloud Serving Benchmark* (YCSB) [Cooper et al. 2010]. Esse *benchmark* mede o desempenho de um banco de dados aplicada a uma série de cargas de trabalho padronizadas que representam casos de uso reais, são obtidas informações como vazão de operações e atraso de processamento das requisições. O teste é dividido em duas partes, o carregamento (*load*) e a execução (*run*).

O processo de *load* executa inserções no banco de dados, o preparando para a etapa seguinte. O processo de execução, *run*, consiste em inserções, atualizações (*updates*) e remoções de chaves no banco de dados. Ao término de ambas as etapas o *software* gera vários resultados, entre eles o tempo de execução (*runtime*) e o número médio de operações por segundo (*Througthput*), entre outros.

O YCSB contém seis diferentes cargas de trabalho, chamadas de *Workload* A à *Workload* F. Foram utilizados 4 delas para realização desse trabalho, as especificações dos testes podem ser vistas na Tabela 2.

Tabela 2. Características dos *workloads* utilizados nesse trabalho [YCSB (2016)]

Workload	Numero de Ops	Leitura	*Update*	Detalhes
A	1000	50%	50%	Simula salvar uma sessão
B	1000	95%	5%	Simula adição de *tags* a foto
C	1000	100%	0%	Simula *cache* de perfil do usuário
D	1000	95%	0%	5% de inserção não ordenadas

2.3. Metodologia dos Testes

Em ambos os modelos do *hardware* utilizado, foi instalado o YCSB e o banco de dados MongoDB, seguindo as indicações na documentação da ferramenta [YCSB (2016)]. Os testes foram rodados por meio de linhas de comando, juntamente com um Script de execução, que realizava os seguintes passos para cada *workload*:

1. Executa o carregamento de chaves através do comando ycsb *load*.
2. Realiza o teste utilizando o comando ycsb *run*, redirecionando a saída para um arquivo txt.
3. Limpa as chaves do banco de dados.

O *script* foi executado 30 vezes para a obtenção dos resultados. Utilizou-se a métrica de vazão de operações (*vazão*) como medida de desempenho do dispositivo. Realizando o cálculo da potência média de consumo do dispositivo utilizando a fórmula: *Potência = Tensão * Corrente*, levando em consideração as informações de consumo de corrente fornecidas pelo fabricante [Fundation 2015], juntamente com dados anteriores [Oliveira and Ataides 2017].

3. Resultados

Os resultados estão sumarizados na Tabela 3, separados por *workloads*. Devido à necessidade de processamento perante as requisições do banco de dados, é estimado que os principais responsáveis pela baixa vazão de operações quanto pela uniformidade dos resultados, sejam seus processadores ARM. A Raspberry PI B+ e Raspberry PI 3 alcançaram, respectivamente, 2 e 4 operações por Watt. Logo a Raspberry PI 3 apresenta uma eficiência energética superior à sua predecessora em teste, na execução do *benchmark*.

Tabela 3. Média da vazão, em operações por segundo, dos *workloads* dos dispositivos

Workload	*Hardware*	Média de Vazão	Potência(W)	Operações por Watt
A	Raspberry PI B+	8,2284	3,5	2,3509
A	Raspberry PI 3 B	29,3186	6,7	4,3759
B	Raspberry PI B+	8,5988	3,5	2,4568
B	Raspberry PI 3 B	31.4668	6,7	4,6965
C	RaspberryPI B+	9,0206	3,5	2,5773
C	Raspberry PI 3 B	32,3902	6,7	4,8343
D	Raspberry PI B+	8,1066	3,5	2,2161
D	Raspberry PI 3 B	30,6513	6,7	4,5748

4. Conclusão

Nesse trabalho foi realizado um estudo comparativo de eficiência energética entre os dispositivos Raspberry PI B+ e Raspberry PI 3. Para a avalização de consumo energético foi utlizado o *benchmark* YCSB.

A partir dos resultados obtidos, é possível concluir que a Raspberry PI 3 apresenta eficiência energética superior a Raspberry PI B+, no *benchmark* aplicado neste estudo, e no quesito medido, operações por Watt.

Como trabalhos futuros, pretende-se prosseguir na comparação energética entre placas SBC (*Small Board Computers*) de fabricantes variados. Como segundo trabalho futuro considerando os resultados obtidos pela avaliação dos dispositivos desse estudo, pretende-se agregar a Raspberry PI 3 a uma nuvem *OpenStack*, assumindo o papel de um nó computacional de baixo consumo. A partir deste ambiente, será realizada uma avalição minusciosa do desempenho e comportamento na nuvem, principalmente o consumo energético, de forma à averiguar a minimização de gastos de energia da nuvem como um todo.

Referências

A. Uchechukwu, K. L. and Shen, Y. (2014). Energy consumption in cloud computing data centers. *International Journal of Cloud Computing and Services Science*, 3(3):145–162.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA. ACM.

Fundation, R. P. (2015). Raspberry pi documentation.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

Oliveira, J. and Ataides, V. (2017). Analize de desempenho de um no computacional de baixo consumo utilizando benchmark ycsb. In *XVII Escola Regional de Alto Desempenho*, pages 207–210, Ijui/RS, Brasil.

Pi, R. (2013). Raspberry pi. *Raspberry Pi*, 1:1.

YCSB (2016). Ycsb github. Disponível em:`https://github.com/brianfrankcooper/ycsb`. Acessado: 2017-11-22.

Comparação do Desempenho de Diferentes Compiladores em Ambientes Virtualizados através de Aplicações Paralelas

Jonatha P. Silveira[1], **Arthur C. Silveira**[1], **Arthur F. Lorenzon**[2]

[1]Faculdade São Francisco de Assis – Porto Alegre – RS – Brasil

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{jonatha.psilveira, arth.csil}@gmail.com, aflorenzon@inf.ufrgs.br

Resumo. *A subutilização de recursos computacionais pode ser evitada com a virtualização, onde diferentes versões de sistemas operacionais e aplicações podem ser executadas em paralelo. Assim, um número maior de aplicações paralelas serão executadas em ambientes virtualizados, as quais podem ser compiladas com diferentes compiladores. Através da execução de aplicações paralelas compiladas com diferentes compiladores em dois monitores de máquinas virtuais, este trabalho mostra a influência da escolha do compilador e do monitor de máquina virtual no desempenho da aplicação paralela.*

1. Introdução

Com o aumento do poder computacional e a possível subutilização dos recursos disponíveis, a virtualização está se tornando cada vez mais comum [Lorenzon et al. 2009]. Através dela, é possível aproveitar o hardware disponível de um único computador para a execução de sistemas operacionais com objetivos diferentes, como por exemplo, aplicações paralelas [Barham et al. 2003]. No entanto, cada monitor de máquina virtual (MMV) possui características distintas, e que impactam de maneira diferente no comportamento do desempenho de aplicações paralelas. Adicionalmente, as aplicações podem ter seu código de máquina gerado por diferentes compiladores. Historicamente, o sistema operacional Linux usa como compilador padrão o *GNU Compiler Collection* (GCC), devido a sua capacidade em gerar código eficiente. Porém recentemente, o compilador *Low Level Virtual Machine* (LLVM) vem ganhando momento e tornando-se mais eficaz em gerar código eficiente.

Desta forma, este trabalho apresenta um estudo experimental com relação ao desempenho dos principais compiladores utilizados hoje em dia (GCC e LLVM), em ambientes virtualizados. Através da execução de quatro aplicações do NAS Parallel Benchmark em dois diferentes monitores de máquinas virtuais (Oracle VirtualBox e VMWare), este artigo mostra que o GCC apresenta desempenho superior na maioria dos casos, enquanto que LLVM possui melhor escalabilidade.

O restante do artigo está organizado da seguinte maneira. A Seção 2 discute os trabalhos relacionados. Os compiladores utilizados e o ambiente de execução são brevemente apresentados na Seção 3. A Seção 4 discute os principais resultados obtidos, enquanto as conclusões e oportunidades de trabalhos futuros são discutidos na Seção 5.

2. Trabalhos Relacionados

O impacto de diferentes compiladores na execução de aplicações paralelas em ambientes virtualizados é pouco estudado hoje em dia. Os autores em [Younge et al. 2011] comparam o desempenho entre monitores de máquinas virtuais para sistemas de alto desempenho. Os resultados mostram que, para a execução do benchmark SPEComp, VirtualBox

possui desempenho similar a execução em uma máquina com hardware real. Por outro lado, [Li 2010] comparam VirtualBox e VMWare de maneira qualitativa, sem a execução de aplicações paralelas.

Considerando o uso de processadores reais, o trabalho desenvolvido em [Park et al. 2014] mostra que o LLVM possui bons resultados em aplicações que são CPU-intensiva, enquanto que o GCC é até 18% mais rápido em aplicações que usam bastante a memória (alocação de estruturas de dados), devido a otimizações de saltos e alocação de registradores. Por outro lado, [Kim et al. 2010] destaca a capacidade do LLVM obter melhor desempenho na execução de chamadas de funções frequentes. Recentemente, [Krause et al. 2017] avaliam o comportamento de aplicações paralelas compiladas com diferentes compiladores, entre eles, LLVM e GCC. Os resultados mostram que o compilador GCC em sua versão 6.2.0 obteve melhor desempenho que o LLVM (Clang 3.9.0) e o ICC 16.0.4. No entanto, o trabalho não considera o uso de ambientes virtualizados.

Considerando os trabalhos destacados, observa-se um espaço de exploração de projeto com relação ao uso de diferentes compiladores para compilar aplicações paralelas que serão executadas em ambientes virtualizados. Desta maneira, este artigo colabora com o estado da arte, comparando o desempenho de aplicações paralelas compiladas com compiladores amplamente utilizados em ambientes virtualizados.

3. Metodologia

3.1. Compiladores

Um compilador pode ser definido como um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador [Aho et al. 2006]. Em geral, um compilador não produz diretamente o código de máquina, mas sim um programa em linguagem simbólica (*assembly*), que é então traduzido para o programa em linguagem de máquina através de montadores. Para desempenhar suas tarefas, um compilador deve executar dois tipos de atividade. A primeira atividade é a análise do código fonte, onde a estrutura e o significado do programa de alto nível são reconhecidos. A segunda atividade é a síntese do programa equivalente em *assembly*.

O GCC é um conjunto de compiladores de linguagens de programação produzido pelo projeto GNU para construir um sistema operacional. Ele tem sido adotado como compilador preferencial para o desenvolvimento de softwares que necessitam ser executados em vários tipos de hardware. Ao utilizar os compiladores do projeto GCC, o mesmo analisador gramatical é usado em todas as plataformas, fazendo com que se o código compila numa, muito provavelmente compilará em todas [Stallman 1988].

LLVM é um compilador amplamente modular que favorece a implementação e a experimentação com diversas análises e transformações de programas. Ele é uma infraestrutura de compilação escrita em C++, criada na Universidade de Illinóis, no início de 2000. Quando se usa LLVM, aplicações são compiladas para programas na linguagem intermediária de LLVM, chamada LLVM-IR. A infraestrutura de compilação LLVM possui também back-ends para diversos processadores, sendo capaz de gerar código para diversas arquiteturas distintas. Com LLVM pode-se, por exemplo, traduzir um programa C para LLVM-IR e após gerar código objeto para executar em um processador MIPS, X86, X86-64, entre outros [Lattner and Adve 2004].

3.2. Ambiente de Execução

Quatro aplicações do conjunto de *benchmarks* paralelos NAS foram utilizados: *conjugated gradient* (CG), *discrete 3D fast Fourier Transform* (FT), *integer sort* (IS), e *Unstructured Adaptative mesh* (UA); todos eles na classe de entrada C. Eles foram executados com diferentes números de *threads* (1, 2, 4 e 8) em cada configuração (MMV e compilador). A versão do GCC utilizada foi a 7.2.0 enquanto que a versão do LLVM (clang) foi a 6.0.0.

Cada MMV (Virtual Box 5.2.0 e VMWare 14.0.0) foi configurado com as seguintes características: processador de 8 núcleos, 8 GB RAM, e sistema operacional Linux Ubuntu 16.04, *kernel* v. 4.4.0. Adicionalmente, o sistema hospedeiro consiste de um processador Intel Core i7 com suporte à execução simultânea de 8 *threads* e 16 GB de memória RAM. Os resultados apresentados na próxima seção consideram a média aritmética da execução de 10 vezes de cada configuração (aplicação, número de *threads*, compilador e MMV). Em todos os casos, o desvio padrão foi inferior a 1% do tempo total de execução.

4. Resultados Experimentais

A Figura 1 apresenta os resultados obtidos para cada aplicação. O eixo *x* de cada gráfico representa a execução com diferentes números de *threads*, enquanto que o eixo *y*, o tempo de execução, em segundos para cada configuração (compilador + MMV).

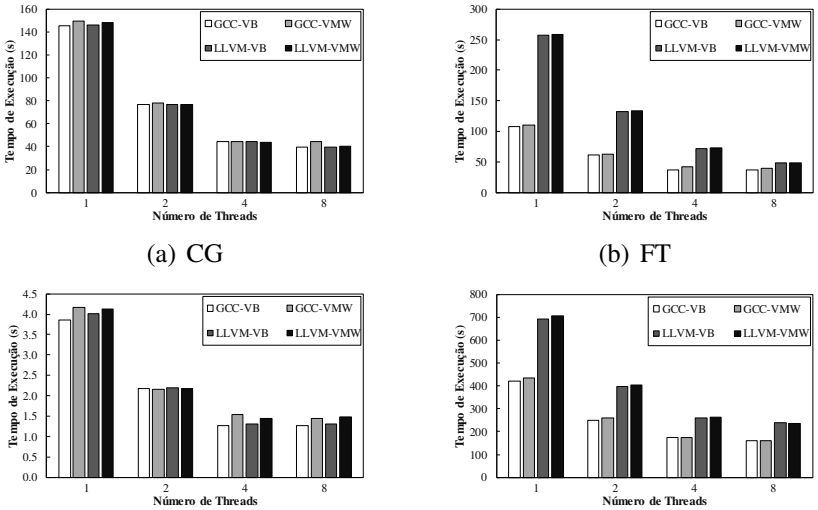
Comparando o desempenho dos monitores de máquinas virtuais, na maioria dos casos, o VirtualBox apresentou desempenho superior (i.e., tempo de execução menor) ao obtido pelo VMWare. Considerando a média geométrica de todas as aplicações, o VirtualBox executou as aplicações compiladas com o GCC 6% mais rápido que o VMWare. Por outro lado, esta diferença é reduzida para apenas 2% quando o compilador LLVM é utilizado.

Quando comparamos o desempenho dos dois compiladores, na maioria dos casos o GCC apresentou menor tempo de execução. Por outro lado, pode-se observar na Figura 1, que quanto maior o número de *threads*, menor é a diferença entre GCC e LLVM. Por exemplo, na execução com apenas 1 *thread* da aplicação FT (Figura 1b, GCC é 58% mais rápido que LLVM. No entanto, na execução com 8 *threads*, a diferença diminui para 20%, quando considerado o MMV VMWare. Isto mostra que, embora GCC possua melhores resultados, o LLVM apresenta melhor escalabilidade quando ocorre o aumento do número de *threads*.

Por fim, os resultados mostram que, enquanto LLVM fornece desempenho similar independente do MMV utilizado, o GCC apresenta melhores resultados no VirtualBox. Portanto, se o programador estiver utilizando o VirtualBox para virtualizar um sistema operacional para execução de aplicações paralelas, o indicado é o uso do compilador GCC.

5. Conclusão

Este trabalho realizou uma comparação de desempenho entre dois compiladores amplamente utilizados pela comunidade acadêmica, em ambientes virtualizados. A partir da execução de um conjunto de benchmarks paralelos, mostrou-se que o GCC apresenta melhor resultado que LLVM na maioria dos casos, independente do MMV utilizado. No entanto, observou-se que o LLVM apresenta melhor escalabilidade que o GCC. Como trabalhos futuros, pretende-se estudar as diferentes otimizações de compiladores e seus efeitos em ambientes virtualizados. Também será expandido o ambiente de execução para compreender outros compiladores (i.e., Intel *compiler*) e outros monitores de máquinas virtuais, tais como o Xen.



(a) CG (b) FT

(c) IS (d) UA

Figura 1. Resultados obtidos (VB: VirtualBox; VMW: VMWare)

Referências

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools (2Nd Edition)*.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *ACM Symp. on Operating Systems Principles*, pages 164–177.

Kim, J. J., Lee, S. Y., Moon, S. M., and Kim, S. (2010). Comparison of llvm and gcc on the arm platform. In *Int. Conf. on Embedded and Multimedia Computing*, pages 1–6.

Krause, A. M., Moro, G. B., and Schnorr, L. M. (2017). Análise do consumo energético de aplicações paralelas com diferentes versões de compiladores. In *Escola Regional de Alto Desempenho - RS*.

Lattner, C. and Adve, V. (2004). Llvm: A compilation framework for lifelong program analysis & transformation. In *Int. Symp. on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04.

Li, P. (2010). Selecting and using virtualization solutions: Our experiences with vmware and virtualbox. *J. Comput. Sci. Coll.*, 25(3):11–17.

Lorenzon, A., P., D., and Rossi, F. D. (2009). Um estudo sobre o xen e a portabilidade de seu escalonamento para arquiteturas paralelas. In *Escola Regional de Alto Desempenho*.

Park, C., Han, M., Lee, H., and Kim, S. W. (2014). Performance comparison of gcc and llvm on the eisc processor. In *Int. Conf. on Electronics, Information and Communications*, pages 1–2.

Stallman, R. M. (1988). Using the gnu compiler collection.

Younge, A. J., Henschel, R., Brown, J. T., von Laszewski, G., Qiu, J., and Fox, G. C. (2011). Analysis of virtualization technologies for high performance computing environments. In *IEEE Int. Conf. on Cloud Computing*, pages 9–16.

Computação distribuída: Desafios do uso do Dropbox como suporte ao espaço de tuplas*

Lucas Eduardo Bretana, Alana Schwendler, Gerson Geraldo H. Cavalheiro

[1]Laboratory of Ubiquitous and Parallel Systems – UFPEL
Pelotas, RS - Brasil

{lebretana, aschwendler, gerson.cavalheiro}
@inf.ufpel.edu.br

Resumo. *A biblioteca ILUCTUS é uma alternativa para o processamento largamente distribuído. Sua solução explora um sistema de arquivos provido em uma nuvem computacional e oferece uma interface de acesso baseada no sistema de Espaço de Tuplas. Neste artigo, questões envolvendo a implementação de ILUCTUS sobre o Dropbox são apresentadas, bem como os tratamentos realizados na biblioteca.*

1. Introdução

As tecnologias de comunicação promovidas pela Internet fomentaram a criação de novos mecanismos de colaboração entre as pessoas. Podemos destacar a grande disponibilidade de serviços de armazenamento em nuvem como um estímulo ao desenvolvimento de projetos para colaboração distribuídas. Com base nisso se deu o desenvolvimento da biblioteca ILUCTUS [Bretana et al. 2017]. A biblioteca apresenta a implementação de uma abstração de comunicação para aplicações distribuídas bem como um modelo de negócios para o desenvolvimento de **Projetos Colaborativos**.

A biblioteca desenvolvida se propõe a servir de meio de compartilhamento do processo de criação do conhecimento. Sendo assim, ILUCTUS permite o compartilhamento de bases de dados entre os participantes do projeto, denominados **Colaboradores**. Além das ferramentas de compartilhamento são também apresentadas primitivas para a manipulação destes dados em um ambiente de endereçamento compartilhado. Para isso a ILUCTUS utiliza do apoio de nuvens computacionais e uma API de comunicação com esta nuvem. Dentre as nuvens que atendiam aos requisitos para o desenvolvimento da biblioteca se optou pela nuvem do Dropbox [Drago et al. 2012] pois, na época da implementação, apresentava melhor documentação da API e um processo robusto de autentificação.

A adoção do Dropbox como suporte de infraestrutura de nuvem para o Espaço de Tuplas implica em decisões de implementações específicas para a ferramenta proposta. Cita-se ainda que existem diferenças entre a versão gratuita, que foi utilizada na presente implementação, da paga. Um destes limites está relacionado ao limite de armazenamento, sendo de 16 GiB na versão não paga e variável nas versões pagas, conforme o plano contratado. O uso de uma API ainda em desenvolvimento também traz alguns desafios

*O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

que devem ser tratados pelas políticas de funcionamento da biblioteca. Neste trabalho são apresentadas as limitações encontradas no desenvolvimento da ILUCTUS e como estas foram tratadas.

2. Arquitetura da Biblioteca

O desenvolvimento da biblioteca ILUCTUS considera um modelo de negócio onde são desenvolvidos **Projetos Colaborativos**. A proposta deste modelo é ilustrada na Figura 1. A instanciação de um novo projeto é realizada por um **Administrador**, com a criação uma base de dados inicial, com os dados brutos a serem evoluídos. Em seguida, um **Colaborador** interessado no projeto requisita acesso ao projeto para o Administrador, que concede o acesso, fazendo o compartilhamento da base de dados com o Colaborador. Após o acesso ser concedido, a base de dados é copiada para o ambiente de nuvem do Colaborador. A evolução dos dados agora se dá pela execução, por parte do Colaborador, de uma aplicação que é dada pelo Administrador. Esta aplicação faz uso da ILUCTUS para realizar a evolução dos dados na nuvem deste Colaborador, segundo as heurísticas implementadas nesta aplicação. As soluções obtidas pela aplicação são salvas na nuvem em uma nova base de dados até que o processamento chegue ao fim. Os resultados obtidos são então compartilhados com o Administrador. Por fim as cópias dos dados presentes na nuvem do Colaborador são apagadas, de forma a não onerar custos de armazenamento.

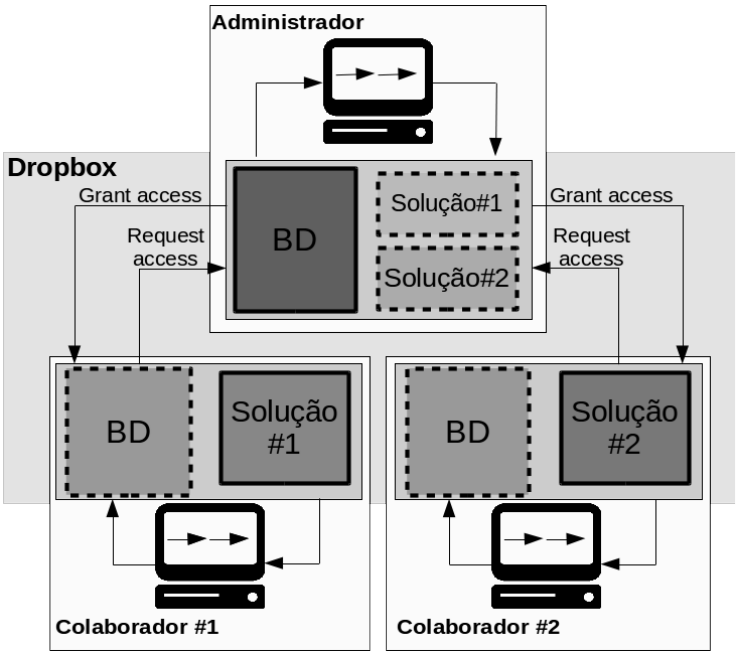


Figura 1. Ilustração da arquitetura da biblioteca.

O modelo de negócio proposto pela biblioteca ILUCTUS abstrai as dificuldades do uso da nuvem, independente da plataforma utilizada para implementação da biblioteca. As soluções desenvolvidas para execução dos processos citados na seção anterior foram tratadas caso-a-caso. Duas situações ocorram nestes casos, uma onde os recursos da API permitiram realizar a implementação da solução, outra em que, por inexistência de suporte, as soluções foram propostas de forma externa à biblioteca. Limitações provenientes da arquitetura oferecida são tratadas conforme as indicações do suporte ou então modificando o desenho da biblioteca de forma à não inviabilizar o projeto.

3. Aspectos da Implementação

O desenvolvimento da ILUCTUS se deu utilizando a linguagem de coordenação Linda [Ahuja et al. 1986] e a nuvem do Dropbox como substrato para armazenamento de dados. Sendo assim, é importante destacar que, para o Dropbox, todo arquivo e diretório compartilhado irá onerar custos de armazenamento para cada um dos usuários que tem acesso. Além disso, internamente um arquivo compartilhado é tratado como pertencente a um mesmo *namespace*. Todos os demais arquivos não compartilhados de um usuário pertencem ao seu *namespace root*. A API do Dropbox se encontra ainda em desenvolvimento e por isso ainda não apresenta todas as funcionalidades esperadas pela comunidade de interesse. Tem-se ainda algumas restrições por parte da API que são reflexos das políticas de funcionamento e arquitetura do Dropbox. A seguir temos os principais desafios encontrados no desenvolvimento da biblioteca ILUCTUS.

Autenticação no Projeto Colaborativo: No Dropbox, o compartilhamento de um diretório requer o envio explícito da concessão de compartilhamento. Junto à esta concessão, segue as permissões de acesso concedidas. A API SDK Java do Dropbox não oferece primitivas de requisição/concessão de acesso a diretórios, embora seja de desejo da comunidade de desenvolvedores sua existência. Na atual implementação da ILUCTUS, estas operações são realizadas de forma externa à aplicação. Para fazer autentificação em um **Projeto Colaborativo** é necessária a troca de *tokens* de identificação entre **Administrador** e **Colaborador**. Da parte do Administrador, esses *tokens*, ou chaves, identificam o projeto e autorizam uma aplicação a ter acesso aos dados compartilhados na nuvem. *Tokens* são únicos para cada Colaborador em um Projeto Colaborativo. Pelo lado do Colaborador, o *token* identifica-o dentro do projeto, oferecendo, desta forma, acesso aos seus resultados na nuvem. Nesta troca inicial de informações entre o Administrador e Colaborador, além dos *tokens*, também é feita a troca de quaisquer outras informações necessárias para dar inicio ao processamento dos dados, tais como identificação da base de dados primária, lista de bases já processadas por Colaboradores pré-existentes, permissões sobre estas bases etc.

Compartilhamento de um mesmo aquivo/diretório: as políticas de compartilhamento não preveem um limite de usuários com acesso ao mesmo arquivo. Porém, como forma de segurança contra abuso do serviço oferecido, na versão não paga do Dropbox, é limitado o número de compartilhamentos que podem ser realizados em um período de 24 horas. Neste caso a API responde com uma exceção única e, então, o usuário é informado que deve fazer nova tentativa a posteriori. A documentação disponível não apresenta o número de compartilhamentos permitidos na versão gratuita.

Número excessivo de requisições: as requisições são feitas em relação a um *namespace*. Essas requisições são tratadas internamente utilizando *locks* por *namespace*, i.e., cada operação deve obter o *lock* correspondente, fazer sua tarefa e devolver o *lock*. Sendo assim, é de se esperar que requisições provenientes de diferentes processamentos colidam e gerem o erro de *too many requests* ou *too many write operations*. A solução dada pela API é manter a natureza síncrona da operação, resubmetendo a requisição após um um tempo de espera (*back off time*). Este tempo de espera é determinado em função da carga à qual a nuvem (Dropbox) está submetida e é informado também pela própria API. Em caso de nova colisão, o processo é repetido

Limite de colaboradores: o desenvolvimento de **Projetos Colaborativos** en-

tende que um maior número de **Colaboradores** representa uma maior evolução dos dados, seja de forma mais rápida ou de maneiras mais diversas. No entanto, no Dropbox, o número de usuários compartilhando uma aplicação é limitado em 10. É viável, mesmo na versão não paga, estender o número de colaboradores até 500, considerando a aprovação pela Dropbox, Inc. de uma solicitação expressa pelo proprietário dos dados.

Abordagem da primitiva Eval: A nuvem do Dropbox funciona em um modelo de PaaS oferecendo o serviço de armazenamento de dados. Para a implementação canônica da primitiva **Eval** é necessário que a nuvem ofereça algum recurso de processamento de dados. Portanto esta primitiva teve sua semântica modificada para possibilitar sua existência. Na biblioteca ILUCTUS o funcionamento da **Eval** funciona como um agendamento de tarefa, onde é colocada uma função com parâmetros de entrada que será computada futuramente e seu resultado será escrito novamente na nuvem. Esta computação será realizada, oportunamente, por algum dos processos colaboradores envolvidos.

4. Conclusões e discussão

O emprego de serviços gratuitos de nuvem na aplicação é um método positivo para a biblioteca pois facilita a participação de vários **Colaboradores** ao se conectarem a um mesmo **Projeto Colaborativo**. Contudo, o uso das ferramentas gratuitas faz com que existam também algumas limitações quanto a plataforma e sua API. Estas limitações devem ser resolvidas, o quanto for possível, na implementação da biblioteca ou então se estender como uma limitação do funcionamento da própria ferramenta. Ainda assim, é possível usufruir das funcionalidades que a ILUCTUS oferece, utilizando-a como apoio para o desenvolvimento de estudos e pesquisas. ILUCTUS já foi atestada em quesitos de desempenho e funcionamento [Schwendler et al. 2017].

Em trabalhos futuros pretende-se dar manutenção ao nosso software, tornando-o compatível com novas versões de API do Dropbox e possibilitando o gerenciamento do compartilhamento de arquivos de dentro da biblioteca. O processo de requisição e concessão de compartilhamento de dados no Dropbox atualmente é manual, contudo é um desejo da comunidade usuária da API Dropbox que este processo esteja disponível por meio de requisições à API. Para a ILUCTUS isso possibilitaria que a autentificação em Projetos Colaborativos possa ser feito internamente à biblioteca, tornando a autentificação mais robusta e segura. Também serão projetadas novas implementações das primitivas de manipulação da biblioteca, bem como, realização de novos testes para aferir a ILUCTUS.

Referências

Ahuja, S., Curriero, N., and Gelernter, D. (1986). Linda and friends. *Computer;(United States)*, 19(8).

Bretana, L. E., Schwendler, A., and Cavalheiro, G. G. H. (2017). ILUCTUS: Uma biblioteca para o apoio ao processamento colaborativo de dados. *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho-WSCAD*.

Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. (2012). Inside Dropbox: Understanding personal cloud storage services. pages 481–494.

Schwendler, A., Bretana, L. E., Pernas, A. M. S., and Cavalheiro, G. G. H. (2017). Um estudo de caso da ferramenta ILUCTUS utilizando o cálculo de Fibonacci. *ERAD Escola Regional de Alto Desempenho*.

Desempenho em Instâncias LXC e KVM de Nuvem Privada usando Aplicações Científicas

Anderson M. Maliszewski[1], **Dalvan Griebler**[1], **Claudio Schepke**[3]

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC)
Faculdade Três de Maio (SETREM) – Três de Maio – RS – Brasil

[2]Universidade Federal do Pampa (UNIPAMPA),
Laboratório de Estudos Avançados (LEA), Alegrete – RS – Brasil

Email: andersonmaliszewski@gmail.com

Resumo. *As nuvens privadas IaaS oferecem um ambiente atraente para aplicações científicas. Como este ambiente possui camadas adicionais de abstração, alcançar um bom desempenho é um desafio. O objetivo é realizar uma avaliação de desempenho das tecnologias de virtualização baseadas em KVM e LXC gerenciadas pelo CloudStack, usando benchmarks da suite NPB-OMP. Os resultados revelaram que LXC supera KVM em 93,75% dos experimentos.*

1. Introdução

A arquitetura de computação em nuvem pode ser representada por uma pilha de camadas, onde a virtualização fica acima dos recursos de hardware, oferecendo suporte as camadas de alto nível, como IaaS, PaaS e SaaS [Chang et al. 2013]. Embora as tecnologias de computação em nuvem tenham evoluído, ainda tem-se perdas de desempenho na camada de virtualização. Outro desafio é apontado por [Iosup et al. 2011], no qual aplicações científicas normalmente requerem recursos da computação de alto desempenho (HPC).

A análise de desempenho de aplicações HPC em ambientes de computação em nuvem tem sido um problema de pesquisa atual. Trabalhos anteriores [Maron et al. 2016, Vogel et al. 2016a] e relacionados (Seção 2) abordam o problema em diferentes aspectos, objetivos, ambientes e condições. A literatura necessita de estudos empíricos que abordem a comparação de tecnologias de virtualização baseadas em contêiner e baseadas em kernel sob condições de ambiente de nuvem privada. Consequentemente, o objetivo é realizar uma avaliação de desempenho das tecnologias de virtualização baseadas em KVM e LXC gerenciadas pelo CloudStack. O foco da avaliação está no paralelismo multithreading usando NPB OpenMP, que pode representar uma ampla gama de aplicações científicas. Portanto, para esse artigo será realizada uma análise de desempenho e comparação entre duas tecnologias de virtualização. Esse trabalho é dividido em 4 seções. A Seção 2 apresenta os estudos relacionados. Na Seção 3 são mostrados os resultados dos experimentos. Por fim, na Seção 4, apresenta-se a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

A utilização da computação em nuvem para HPC está em ascensão na literatura. O trabalho de [Roloff et al. 2012] realiza uma comparação detalhada de aplicações HPC executando em três provedores de nuvem (Amazon EC2, Microsoft Azure e Rackspace). As características de desempenho e eficiência de custo, foram listadas e comparadas em

clusters. Os experimentos foram realizados usando a versão OpenMP e MPI do NPB em um ambiente de nuvem pública. Os resultados mostraram que HPC pode funcionar de forma eficiente na nuvem, mas os autores enfatizam grandes diferenças entre provedores de nuvem. Em contraste, este artigo está focado em nuvem privada gerenciada pelo CloudStack e fornece uma análise de desempenho entre as tecnologias KVM e LXC. Para avaliar o desempenho de comunicação das aplicações HPC, o estudo desenvolvido por [Okada et al. 2016] usou os benchmarks da suite NPB-MPI. Foram comparados o comportamento da execução no Google Compute Engine, com OpenStack usando a virtualização KVM e um sistema multiprocessador NUMA usando o LXC. Os autores concluíram que os usuários de HPC devem usar o número apropriado de vCPUs em cada VM. Diferentemente, o foco neste artigo está no paralelismo multithreading usando benchmarks da suite NPB-OMP para comparar a virtualização LXC e KVM em condições de nuvem privada gerenciadas pelo CloudStack.

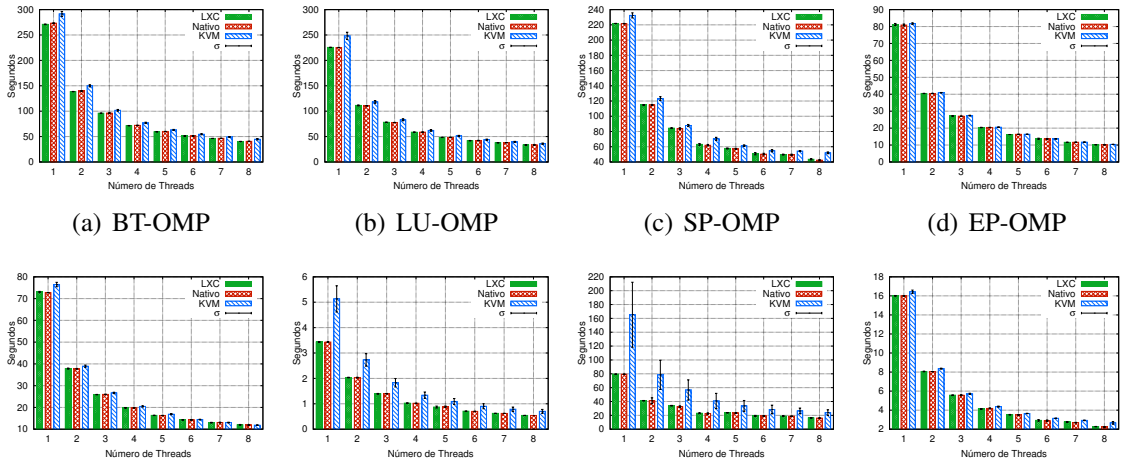
Estudos realizados por [Vogel et al. 2016b] avaliaram diferentes ferramentas IaaS com a virtualização KVM para identificar possíveis impactos de desempenho causados pelas ferramentas de gerenciamento de nuvem. Os testes foram feitos usando os benchmarks da suite NPB com OpenMP e MPI em OpenStack, OpenNebula e CloudStack. Os experimentos revelaram que não há diferença de desempenho significativa entre as ferramentas da nuvem. Neste artigo, uma implementação do CloudStack combinada com as tecnologias de virtualização LXC e KVM foi realizada.

3. Resultados

Testes foram realizados no LARCC [1] da SETREM em uma nuvem composta por três servidores com a mesma configuração. O nó front-end é responsável pela administração da nuvem e dois nós de computação são responsáveis pela execução das experimentos. Os nós executam o S.O. Ubuntu 14.04 (kernel 3.19.0) em um processador Intel Xeon X5560, 24GB de RAM (1333MHz), disco de armazenamento Sata II e a rede gigabit em cada nó. A ferramenta CloudStack 4.8 foi utilizada como plataforma de nuvem. Também foi usado GNU Fortran (GCC) versão 4.8.5 20150623 (Red Hat 4.8.5-11). O armazenamento primário e secundário são montados no nó front-end e usam o protocolo NFS para se comunicar e compartilhar recursos entre nós. Estes são responsáveis pelo armazenamento das imagens da VM, modelos e imagens do sistema operacional. A oferta de serviços nas instâncias da nuvem utilizaram a capacidade total das máquinas (24 GB de RAM, 8 vCPUs). Para os experimentos, instalamos KVM v.2.0.0 e LXC v.1.0.8.

Para os testes de desempenho NPB-OMP 3.3.1 foi utilizado e compilado com a classe B. Executou-se para cada instância de 1 a 8 threads, resultando em oito execuções separadas. Cada experimento foi repetido 10 vezes. Além disso, implementou-se uma nuvem baseada em CloudStack LXC (virtualização de contêiner) e outra com a nuvem CloudStack KVM (virtualização completa) sobre o mesmo hardware. Finalmente, o ambiente nativo foi comparado para criar uma *baseline* para os resultados. Na Figura 1 são mostrados os tempos de execução dos três ambientes distintos, os quais foram identificados com uma cor específica, vermelha para o Nativo, verde para LXC e azul para KVM. Primeiramente, pode-se observar que o ambiente de nuvem baseado em KVM apresentou os piores resultados, indicando que esse tipo de virtualização impõe um *overhead*

[1] http://larcc.setrem.com.br/



(a) BT-OMP (b) LU-OMP (c) SP-OMP (d) EP-OMP

(e) FT-OMP. (f) IS-OMP (g) CG-OMP (h) MG-OMP

Figura 1. Resultados das aplicações da suite NPB-OMP.

considerável em relação aos ambientes LXC e Nativo.

As aplicações BT (Figura 1(a)), LU (Figura 1(b)) e SP (Figura 1(c)) efetuam soluções de sistemas lineares e tem impacto semelhante em KVM. O ganho de desempenho do *kernel* EP (Figura 1(d)) é perceptível a medida que as threads aumentam. Esse resultado também é enfatizado pelos autores [Hashimoto and Aida 2012] que explicam que a perda de desempenho no EP é mínimo porque este requer menos capacidade de memória. No FT (Figura 1(e)) algumas perdas de desempenho são apresentadas, principalmente na primeira thread em KVM. Essa aplicação usa uma grande quantidade de memória e, por isso, sofre com a adição da camada de virtualização. Da mesma forma, as aplicações CG (Figura 1(g)) e IS (Figura 1(f)) apresentaram resultados diferentes entre os ambientes de nuvem. Os autores [Regola and Ducom 2010] enfatizaram que os *overheads* criados pela virtualização são mais significativos em benchmarks que usam uma grande quantidade de comunicação ou acesso a memória. IS tem características específicas como o menor conjunto de trabalho e a mais rápida execução entre os *kernels* da suíte NPB. Percebe-se que IS executando em KVM (cor azul) tem resultado ruim mesmo sem a utilização de paralelismo. O problema de escalabilidade desta aplicação já foi observada pelos autores [Strazdins et al. 2012]. Por sua vez, MG (Figura 1(h)) sofre *overhead* em KVM. Embora MG tenha utilização intensiva de memória, o acesso a memória não teve grande impacto.

4. Conclusões

Este artigo apresentou uma avaliação de desempenho de experimentos realizados em condições de nuvem privada implantadas com a plataforma CloudStack. O ambiente testado suportou as tecnologias de virtualização KVM e LXC, onde os benchmarks da suite NPB OpenMP foram experimentados. Concluímos que LXC fornece uma sobrecarga menor para este tipo de aplicações. Ao comparar os tipos de instância, a virtualização baseada em contêiner supera a virtualização baseada em kernel em 93,75 %. Somente para um caso excepcional (FT com 8 threads) que KVM supera LXC com uma diferença mínima.

Descobriu-se que o alto uso de memória e o acesso nesses aplicativos impactam significativamente no desempenho de instâncias KVM. Isso ocorre porque a virtualização

completa adiciona mais instruções que precisam ser gerenciadas pela CPU. Consequentemente, é preciso tratar mais informações que causam degradação do desempenho em comparação com o desempenho do ambiente nativo. Como estudos futuros, planeja-se avaliar diferentes domínios de aplicação para descobrir diferentes comportamentos; incluir diferentes ferramentas de gerenciamento do IaaS (i.e., OpenStack, OpenNebula); e realizar experiências com *overprovision* ou até *multi-tenancy*.

Referências

[Chang et al. 2013] Chang, V., Walters, R. J., and Wills, G. (2013). The Development that Leads to the Cloud Computing Business Framework. *International Journal of Information Management*, 33(3):524 – 538.

[Hashimoto and Aida 2012] Hashimoto, Y. and Aida, K. (2012). Evaluation of performance degradation in hpc applications with vm consolidation. *Third International Conference on Networking and Computing*, pages 1–5.

[Iosup et al. 2011] Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. (2011). Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Trans. on Par. and Dist. Sys.*, 22(6):931–945.

[Maron et al. 2016] Maron, C. A. F., Griebler, D., Schepke, C., and Fernandes, L. G. (2016). Desempenho de OpenStack e OpenNebula em Estações de Trabalho: Uma Avaliação com Microbenchmarks e NPB. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação (REABTIC)*, 6(1):15.

[Okada et al. 2016] Okada, T. K., Goldman, A., and Cavalheiro, G. G. H. (2016). Using NAS Parallel Benchmarks to Evaluate HPC Performance in Clouds. *International Symposium on Network Computing and Applications*, pages 1–4.

[Regola and Ducom 2010] Regola, N. and Ducom, J. C. (2010). Recommendations for virtualization technologies in high performance computing. In *CloudCom*, page 8.

[Roloff et al. 2012] Roloff, E., Diener, M., Carissimi, A., and Navaux, P. O. (2012). High Performance Computing in the cloud: Deployment, performance and cost efficiency. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 371–378. IEEE.

[Strazdins et al. 2012] Strazdins, P. E., Cai, J., Atif, M., and Antony, J. (2012). Scientific application performance on hpc, private and public cloud resources: A case study using climate, cardiac model codes and the npb benchmark suite. In *PhD Forum (IPDPSW)*, pages 1416–1424. IEEE.

[Vogel et al. 2016a] Vogel, A., Griebler, D., Maron, C. A. F., Schepke, C., and Fernandes, L. G. (2016a). Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 672–679, Crete, Greece. IEEE.

[Vogel et al. 2016b] Vogel, A., Maron, C. A. F., Griebler, D., and Schepke, C. (2016b). Medindo o Desempenho de Implantações de OpenStack, CloudStack e OpenNebula em Aplicações Científicas. In *16th Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul (ERAD/RS)*, pages 279–282, São Leopoldo, RS, Brazil.

Exploração de Computação Híbrida com OpenACC em um Algoritmo Friends-of-Friends para Classificação de Objetos Astronômicos

Ana Luísa V. Solórzano[1], **Andrea S. Charão**[1]
Renata S. R. Ruiz[2], **Haroldo F. de Campos Velho**[2]

[1] Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria

[2]Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais

Resumo. *A utilização de dispositivos aceleradores para processar porções de um código mostra-se promissora na área de computação híbrida. Neste trabalho, utilizou-se a ferramenta de paralelização com aceleradores OpenACC em um sistema composto por CPU e GPU para o algoritmo Friends-of-Friends, que lida com grandes quantidades de dados. Os resultados indicam que o algoritmo pode ser explorado neste ambiente, porém com algumas limitações.*

1. Introdução

A utilização de computação híbrida vem demonstrando ser uma opção vantajosa para resolver problemas que lidam com amplas quantidades de dados. Esse fato pode ser bem explorado para códigos em que certas porções se comportam melhor em CPU, enquanto outras têm maior potencial se executarem em dispositivos aceleradores.

Atualmente, uma plataforma de aceleração bastante utilizada é a GPU, mesmo existindo outras como FPGAs e Intel® Xeon Phi®. Dentre as diversas ferramentas para paralelização de programas com aceleradores estão CUDA, OpenCL e OpenACC. Neste trabalho, optou-se pelo padrão OpenACC, baseado no uso de diretivas para distribuir a computação em um ambiente híbrido.

Um cenário que lida com problemas de grande envergadura é a análise computacional de dados astronômicos provenientes de simulações cosmológicas e de observatórios virtuais. Considerando sua importância para, por exemplo, prever comportamentos similares na formação de grandes estruturas observadas no Universo, como galáxias e aglomerados de galáxias [Ruiz 2011], o desenvolvimento de programas eficientes e com um baixo tempo de execução é primordial.

Um dos algoritmos mais utilizados em simulações de N-corpos para classificar galáxias e aglomerados de galáxias é o Friends-of-Friends (FoF) [Huchra e Geller 1982, Caretta et al. 2008]. Em trabalhos anteriores, um algoritmo FoF sequencial foi implementado e paralelizado utilizando MPI para execução em um *cluster* [Ruiz et al. 2009], e posteriormente foi paralelizado com OpenMP [Berwian et al. 2017]. Entretanto, o seu desempenho em um ambiente de computação híbrida ainda não havia sido explorado.

2. Algoritmo Friends-of-Friends (FoF)

O algoritmo FoF é utilizado para identificar estruturas do Universo através do agrupamento de partículas com uma dada proximidade física [Huchra e Geller 1982, Caretta et al. 2008]. Para isso, ele considera que se dentro de uma esfera de raio de percolação R, ao redor de uma partícula, existirem outras partículas, elas são consideradas todas pertencentes ao mesmo grupo e chamadas de amigas. Após, ele realiza o mesmo procedimento para uma esfera ao redor de cada amiga, até que nenhuma nova amiga possa ser adicionada ao grupo.

A execução do algoritmo recebe como parâmetro um arquivo com um grande volume de dados, em que cada linha contém informações relativas a uma partícula e a primeira linha define o número de partículas a serem classificadas, e um valor para o raio de percolação. O tamanho do raio de percolação varia de acordo com os objetos astronômicos a serem identificados, utilizando-se 0.1 para identificar galáxias, 0.18 para aglomerados de galáxias e 0.92 para super aglomerados de galáxias [Caretta et al. 2008].

3. Paralelização do FoF com OpenACC

Optou-se por utilizar a ferramenta OpenACC, pois ela possibilita a geração de códigos paralelizados para placas gráficas aceleradoras sem a necessidade de maior conhecimento quanto a sua arquitetura, usando diretivas de compilação. Isso permite gerar um código com poucas modificações em comparação ao original.

A paralelização foi feita no laço mais interno do algoritmo (Algoritmo 1), visto que ele não possui dependência de dados. Para isso, foi utilizada a diretiva `#pragma acc parallel`, que descreve uma região do código a ser acelerada com os *kernels* da GPU. Notou-se que a variável `dist`, que armazena o valor da distância entre duas partículas, deve ser privada para evitar valores inconsistentes.

```
for (i = 0 ; i < N ; i++){
  k++;
  while (igru[i] != 0 )i++;
  igru[i] = k;
  for (j = i ; j < N ; j++){
    if(igru[j] == k) {
      for (l = (i + 1) ; l < N ; l++){
        if (igru[l] == 0){
          dist = sqrt((x[j] - x[l])*(x[j] - x[l]) +
          (y[j] - y[l])*(y[j] - y[l]) +
          (z[j] - z[l])*(z[j] - z[l]));
          if (dist <= rperc){
            igru[l] = k;
          }
        }
      }
    }
  }
}
```

Algoritmo 1. Versão sequencial

Visto que o laço paralelizado lê os vetores `x`, `y`, `z` e `igru`, e também pode alterar o último, utilizou-se a diretiva `#pragma acc data copyin` antes do laço mais externo, de modo a copiar uma única vez para a GPU os valores dos vetores apenas lidos. O `igru` precisou de maior atenção, pois é modificado antes de ser lido no laço paralelizado e pode ser modificado dentro dele. Esse tratamento gerou duas abordagens semelhantes:

1. A primeira consistiu em utilizar a diretiva `#pragma acc copy`, que copia o vetor da CPU para a GPU antes de entrar na região paralela e da GPU para a CPU antes de sair, garantindo a consistência dos dados do vetor durante toda a execução.
2. A segunda utilizou a diretiva `#pragma acc update device`, que copia os valores atualizados no `igru` da memória local para o dispositivo após sua alteração no laço mais externo, e `#pragma acc update self`, que atualiza os dados modificados no dispositivo para a memória local após sair da região paralela. Para isso, precisou-se alocar o vetor no acelerador usando a diretiva `#pragma acc data create` antes do laço mais externo.

4. Experimentos e Resultados

Os experimentos foram realizados em um servidor com processador Intel® Xeon® E5620 de quatro cores físicos e oito virtuais, com 32KB de cache L1, 256KB de cache L2 e 12MB de cache L3, e 12 GB de memória, e com uma GPU NVIDIA GeForce GTZ Titan X. O sistema operacional é Debian 9, versão do Linux 4.9.0-2, e o compilador utilizado foi o da Portland Group (PGI) edição Community [NVIDIA] versão 17.4-0.

A compilação do código original apresentou alguns erros, acusando sobrecarga de funções. Possivelmente, isso ocorreu porque a `libc` nativa define as funções das bibliotecas declaradas no mesmo momento em que a `libstdc++` também as carrega, assumindo que a libc não o faça. Assim, a solução encontrada foi não utilizar as bibliotecas conflitantes `math` e `stdlib`, adaptando um trecho do código que utilizava a função `sqrt()` para cálculo da raiz quadrada.

A partir de três arquivos de entrada com 65536 partículas (Arquivo 1), 174761 partículas (Arquivo 2) e 1048576 (Arquivo 3), foram realizadas 30 execuções dos arquivos 1 e 2 e 10 execuções do Arquivo 3, todas utilizando raio de percolação 0.1. Os resultados obtidos são apresentados na Tabela 1.

Arquivo	Média (s) serial	Média (s) versão 1	Speedup versão 1	Média (s) versão 2	Speedup versão 2
1	25.25	21.07	1.20	19	1.33
2	162	126.31	1.28	112.40	1.44
3	5120.30	4015.29	1.27	3582.66	1.43

Tabela 1. Análise de desempenho para as duas abordagens com OpenACC

Utilizando o *profiler* nvprof, investigou-se a diferença entre os *speedups* das duas abordagens. Na segunda, o *pragma* de atualização de dados da CPU para a GPU apenas sincroniza os valores alterados, enquanto que na primeira, o *pragma* de cópia escreve todo o vetor na GPU a cada iteração do laço, resultando em um maior processamento. Com essa análise, também se constatou que o maior tempo gasto em ambas abordagens,

cerca de 60% do tempo total, está no bloqueio da CPU a espera dos *kernels* lançados pelo acelerador, ou seja, tempo em que a GPU está fazendo o seu trabalho.

Os resultados de desempenho demonstraram que, apesar de fazer uso do processamento paralelo em GPU, o programa ficou distante de explorar todo o potencial oferecido pela placa. Esse fato, sustenta-se na lógica utilizada no algoritmo FoF, que contém várias estruturas condicionais e dependências de dados, o que dificultou a sua paralelização no dispositivo utilizado.

Também, o laço mais interno possui um desbalanceamento de carga, o que levou à utilização de diretivas de escalonamento para determinar a quantidade de dados a serem processados por *kernel* lançado. Entretanto, não obteve-se êxito, visto que a computação no laço paralelizado não depende apenas do número de partículas a serem processadas, mas também das informações quanto as suas posições, o que implicaria em cargas distintas para cada arquivo de entrada.

5. Considerações Finais

A partir dos resultados, observou-se que o algoritmo Friends-of-Friends pode explorar o uso de uma GPU como acelerador, porém não faz uso de todo o seu potencial. Por outro lado, embora OpenACC não proporcione ao desenvolvedor acesso a detalhes sobre o dispositivo, ela é uma ferramenta simples de ser utilizada e de alta portabilidade, sendo vantajosa para investigações iniciais. Assim, para trabalhos futuros, poderiam ser investigadas outras alternativas envolvendo a reescrita do algoritmo FoF, visando o seu processamento paralelo em um sistema de computação híbrida composto por CPU e GPU.

Referências

Berwian, L., Zancanaro, E. T., Cardoso, D. J., Charão, A. S., da Rocha Ruiz, R. S., e de Campos Velho, H. F. (2017). Comparação de estratégias de paralelização de um algoritmo friends-of-friends com openmp. In *Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, pages 279 – 252.

Caretta, C. A., Rosa, R. R., de Campos Velho, H. F., Ramos, F. M., e Makler, M. (2008). Evidence of turbulence-like universality in the formation of galaxy-sized dark matter haloes. *Astronomy & Astrophysics*, 487(2):445–451.

Huchra, J. P. e Geller, M. J. (1982). Groups of galaxies. I - Nearby groups. *Astrophysical Journal*, 257:423–437.

NVIDIA. PGI community edition. Available: https://developer.nvidia.com/openacc-toolkit.

Ruiz, R. S. d. R. (2011). *Turbulência em cosmologia: análise de dados simulados e observacionais usando computação de alto desempenho*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

Ruiz, R. S. R., Campos Velho, H. F., e Caretta, C. A. (2009). Parallel algorithm friends-of-friends to identify galaxies and cluster of galaxies for dark matter halos. In *Proceedings...* Workshop dos Cursos de Computação Aplicada do INPE, 9. (WORCAP)., Instituto Nacional de Pesquisas Espaciais (INPE).

Explorando o Paralelismo de Stream em CPU e de Dados em GPU na Aplicação de Filtro Sobel

Charles Michael Stein, Dalvan Griebler

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC)
Faculdade Três de Maio (SETREM) – Três de Maio – RS – Brasil

charlesmst@gmail.com,dalvan.griebler@acad.pucrs.br

Resumo. *O objetivo deste estudo é a paralelização combinada do stream em CPU e dos dados em GPU usando uma aplicação de filtro sobel. Foi realizada uma avaliação do desempenho de OpenCL, OpenACC e CUDA com o algorítimo de multiplicação de matrizes para escolha da ferramenta a ser usada com a SPar. Concluiu-se que apesar da GPU apresentar um speedup de 11.81x com CUDA, o uso exclusivo da CPU com a SPar é mais vantajoso nesta aplicação.*

1. Introdução

As aplicações de processamento de *Stream* estão presentes em diversas áreas. Por exemplo, no monitoramento de eventos sísmicos, nas análises de mercado de bolsa de valores, tratamento de imagem, áudio e vídeo. Uma das características destas aplicações é o processamento de um fluxo contínuo de dados e a estrutura bem definida em uma sequencia de operações, com o formato de uma linha de produção. Muitas destas aplicações possuem requisitos de desempenho associados ao processamento de tempo real e a alta vazão. Assim, torna-se necessário a introdução do paralelismo a fim de atender estas demandas.

Existem diversas bibliotecas ou *frameworks* de programação paralela para diferentes arquiteturas atualmente disponíveis. Somado a isso, existem também padrões paralelos que auxiliam o programador a decompor um problema [McCool et al. 2012]. Para explorar o paralelismo de *stream*, os mais usados são os padrões *Farm* e *Pipeline*, que são muitas vezes combinados [Griebler and Fernandes 2013]. No paralelismo de dados, os mais usados são o *Map* e o *Reduce*, que podem ser combinados [Danelutto et al. 2017]. Além disso, é conhecido que problemas que podem ser decompostos independentemente sobre um conjunto de dados são mais recomendados para GPU, pois trata-se de uma arquitetura criada para este cenário. Enquanto isso, a CPU possui uma unidade de controle mais robusta e suporta eficientemente o paralelismo de dados, tarefas e *stream*.

Desta forma, o objetivo deste artigo é realizar a exploração do paralelismo nestes dois níveis arquiteturais em uma aplicação de filtro sobel. A proposta inicial é avaliar quais das interfaces de programação paralela (OpenCL, OpenACC e CUDA) oferecem o melhor desempenho através da paralelização do algorítimo de multiplicação de matrizes. Uma vez definida a melhor, o problema de pesquisa é descobrir se o desempenho nesta aplicação aumenta ou diminui fazendo o uso combinado do paralelismo de *stream* na CPU com a SPar e paralelismo de dados na GPU com a interface escolhida. O desempenho desta aplicação com a SPar [Griebler et al. 2017] pode ser observado em trabalho anteriores, aplicando diferentes estratégias [Griebler et al. 2015]. Consequentemente, a principal contribuição está no suporte ao paralelismo para GPU e sua combinado nesta aplicação de filtro sobel. O artigo apresenta primeiramente as avaliações e o desenvolvimento na Seção 2 e depois discute os resultados dos experimentos realizados na Seção 3.

2. Avaliação e Desenvolvimento

Primeiramente, iremos fazer uma avaliação do desempenho das interfaces de programação paralelas mais utilizadas para explorar o paralelismo nas GPUs. Diante disso, foram implementadas diversas versões do algoritmo de multiplicação de matrizes com CUDA, OpenCL e OpenACC. Dentre estas, foram escolhidas as versões que ofereceram o melhor desempenho para plotar no gráfico da Figura 1. Estas versões foram desenvolvidas da seguinte forma. **CUDA**: Dentro do *kernel*, a multiplicação é feita em blocos de 8x8. As *threads* copiam de forma conjunta matrizes 8x8 para a memória compartilhada e fazem o cálculo de forma cumulativa. **OpenCL**: Semelhante à versão em CUDA, cada grupo de trabalho faz cache de matrizes 8x8, e os *threads* realizam o cálculo de forma cumulativa. **OpenACC**: A implementação possui anotações nos dois laços mais externos e a soma dos valores utilizou a redução do OpenACC. As transferências de memória foram feitas de forma explicita.

Como pode ser visto, o CUDA foi a biblioteca que conseguiu obter o melhor desempenho, alcançando 36x de *speedup* em matrizes 2000x2000. Para estes testes, foi utilizado o mesmo *hardware* e configuração descrito na Seção 3. O melhor desempenho do CUDA ocorre devido ao OpenACC e OpenCL suportarem outras arquiteturas, gerando assim uma implementação mais genérica.

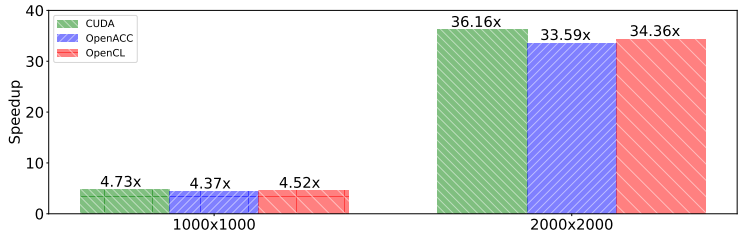


Figura 1. *Speedup* das versões otimizadas para multiplicação de matrizes.

A aplicação de filtro sobel le todas as imagens de um diretório e destaca as bordas das imagens. O operador sobel é aplicado em cada pixel da imagem, que leva em consideração os pixeis vizinhos para obter um valor que representa a diferença de corres. O Código 1 apresenta a implementação desta aplicação usando a SPar. Para cada arquivo, um trabalhador lê, aplica o filtro e salva o resultado.

O Código 2 apresenta o operador/função sobel em formato de um kernel CUDA. Como pode ser observado, a execução ocorre em blocos que são calculados baseado no tamanho da imagem. O *kernel* aplica o filtro no índice atual da imagem e escreve o resultado em uma variável global, que ao final é transferida para a CPU. Note que ambos os códigos são versões totalmente voltados para a sua arquitetura alvo. Para realizar a combinação, foi mantida as anotações do Código 1, porém, ao invés de chamar o operador sobel sequencial na linha 10, inseriu-se todo o código necessário para inicializar e chamar o *kernel* do Código 2, usando a biblioteca CUDA.

A transferência de dados entre CPU e GPU foi implementada de três formas diferentes (resultados estão na Figura 3). A primeira faz a gestão do dados de forma explicita com a API do CUDA. A outra utilizou o recurso *stream*, permitindo que a comunicação CPU e GPU e o processamento dos *kernels* ocorram de forma simultânea. Por último, foi utilizado um recurso *Unified Memory* que faz as transferências automaticamente.

```
DIR *dptr = opendir(...);
struct dirent *dfptr;
[[spar::ToStream, spar::Input(dptr, dfptr,
     tot_img, tot_not), spar::Output(tot_img,
      tot_not)]]
while ((dfptr = readdir(dptr)) != NULL){
 // preprocessing
 if (file_extension == "bmp"){
  tot_img++;
  im = read(name, h, w);
  [[spar::Stage, spar::Input(h, w, im,
    newname), spar::Output(new_im), spar::
    Replicate(2)]]{
   new_im = sobel(im, h, w);
   write(newname, new_im, h, w);
  } //end stage
 }else{ tot_not++; }
}
```

Código 1. Aplicação com SPar.

```
__global__ void sobel_k(unsigned char *fi,
     unsigned char *im, int w, int h){
     int y = blockIdx.y * blockDim.y +
     threadIdx.y + 1;
     int x = blockIdx.x * blockDim.x +
     threadIdx.x + 1;
     unsigned char b[3][3];
     if (x < (w-1) && y < (h-1)){
          for (int v = 0; v < 3; v++){
               for (int u = 0; u < 3; u++){
                    b[v][u] = im[(((y + v - 1) *
      w) + (x + u - 1))];
                    fi[((y*w)+x)]=Sobel(b);
               }
          }
     }
}
```

Código 2. Sobel em CUDA

3. Experimentos

Para analisar o desempenho das versões implementadas, foram usadas sempre 100 imagens com dimensões de 800x600, 3000x2250, 5000x5000 e 10000x10000. O *hardware* utilizado foi um computador com CPU Intel Xeon E5-2620 v3 @2.40GHz de 12 núcleos, possuindo uma GPU Titan X(pascal) de 12GB e 3584 CUDA cores.

O primeiro teste foi a comparação do desempenho da implementação CUDA com a sequencial. A Figura 2 apresenta o desempenho nas diferentes cargas de dados, destacando o tempo total da execução e o tempo da região paralela (operador sobel em CUDA). Como pode ser visto na Figura 2, a região paralela obteve ganho de desempenho a partir de imagens 3000x2250, com um *speedup* de 1.86x a 11.81x, porém, isso não refletiu em um ganho de desempenho no tempo total de mesma escala. O gargalo na leitura e escrita de arquivos não permite a utilização eficiente da GPU.

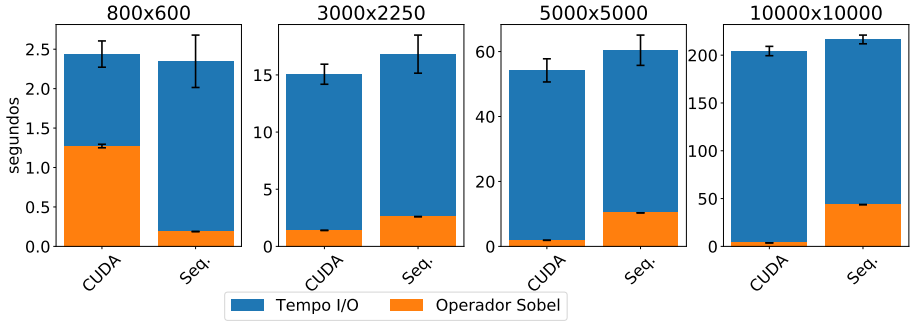


Figura 2. CUDA vs sequencial

A SPar permitiu o paralelismo de *Stream* para CPU de forma simples e ao mesmo tempo eficiente conforme podemos ver na Figura 3, onde o tempo total de processamento das diferentes versões desenvolvidas foi plotado. Nota-se que a execução em imagens de baixa resolução na GPU apresentou menor desempenho que as versões paralelizadas apenas em CPU (versão SPar). Em imagens a partir de 5000x5000, o desempenho entre versões que utilizam GPU se assemelha ao uso exclusivo de CPU. Embora existe o suporte ao paralelismo nos dois níveis arquiteturais, a frequente cópia dos dados e as operações de

I/O afetaram a escalabilidade nesta aplicação. Na carga de imagens 5000x5000, nota-se claramente um problema de balanceamento de carga com a SPar, ocorrendo também nas outras cargas quando testado com um número de réplicas em específico. O desempenho poderia ser melhorado usando o escalonador sob-demanda da SPar (`spar_ondemand`).

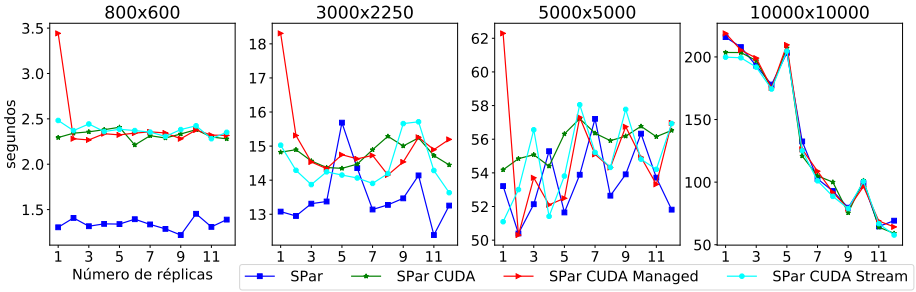


Figura 3. Resultados com o uso combinado do paralelismo em CPU e GPU.

4. Conclusões

Este artigo fez uma análise da aplicação de filtro sobel para GPU em CUDA e também o seu uso combinado com o paralelismo da CPU com a SPar. Apesar do operador sobel apresentar um *speedup* de até 11.81x em CUDA, este desempenho não refletiu em ganhos no tempo total, já que o grande limitador da aplicação foi o tempo de I/O. Neste sentido, observou-se que é melhor utilizar apenas o paralelismo em CPU, já que o processamento realizado na GPU não justifica o *overhead* da transferência de dados. Como trabalhos futuros, almeja-se avaliar se outras aplicações de *stream* conseguem aumentar o desempenho explorando o paralelismo combinado de CPU e GPU.

Referências

[Danelutto et al. 2017] Danelutto, M., de Matteis, T., de Sensi, D., Mencagli, G., and Torquati, M. (2017). P3ARSEC: Towards Parallel Patterns Benchmarking. In *32nd Annual ACM Symposium on Applied Computing*, SAC '17, Marrakech, Morocco. ACM.

[Griebler et al. 2015] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2015). An Embedded C++ Domain-Specific Language for Stream Parallelism. In *International Conference on Parallel Computing*, ParCo'15, pages 317–326, Edinburgh, UK. IOS Press.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):20.

[Griebler and Fernandes 2013] Griebler, D. and Fernandes, L. G. (2013). Towards a Domain-Specific Language for Patterns-Oriented Parallel Programming. In *Programming Languages - 17th Brazilian Symposium - SBLP*, volume 8129 of *Lecture Notes in Computer Science*, pages 105–119, Brasilia, Brazil. Springer Berlin Heidelberg.

[McCool et al. 2012] McCool, M., Robison, A. D., and Reinders, J. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. MKF, MA, USA.

Explorando Paralelismo em Python para Múltiplas Execuções de Modelos de Culturas Agrícolas

Lucas F. da Silva[1], **Andrea S. Charão**[1], **Romulo P. Benedetti**[3], **Nereu A. Streck**[3]

[1] Curso de Bacharelado em Ciência da Computação

[2]Departamento de Linguagens e Sistemas de Computação

[3]Departamento de Fitotecnia

Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

{lferreira, andrea, rbenedetti}@inf.ufsm.br

Resumo. *Neste trabalho, explorou-se um dos recursos de paralelismo disponíveis para a linguagem Python, o multiprocessing, no contexto da paralelização de um script que automatiza execuções de modelos agrícolas. Como resultado, obteve-se uma melhoria de desempenho em relação ao script original, reduzindo o tempo necessário para obter os dados de saída desejados.*

1. Introdução

A linguagem Python tornou-se popular em uma ampla gama de aplicações, prestando-se desde à iniciação à programação até à computação científica. Trata-se de uma linguagem multi-paradigma, interpretada, projetada para favorecer a legibilidade do código, priorizando-a em relação à velocidade de execução. Embora inicialmente distante da computação de alto desempenho, onde predominam linguagens como Fortran e C/C++, a linguagem Python tem despertado o interesse da comunidade dessa área. Exemplos típicos nesta linha são pacotes para multiprocessamento ou ferramentas voltadas à programação em Python com aceleração em GPUs.

Na Universidade Federal de Santa Maria, uma colaboração entre o Departamento de Fitotecnia e o Departamento de Linguagens e Sistemas de Computação têm propiciado o desenvolvimento de diferentes tipos de software visando a modelagem de culturas agrícolas. Nesta colaboração, modelos que efetuam cálculos em Fortran são acoplados a interfaces gráficas em Java ou Javascript, produzindo ferramentas disponíveis ao público, como por exemplo SimulArroz, Simanihot e Phenoglad, respectivamente para culturas de arroz, mandioca e gladíolo [CropModelsUFSM 2017].

Em pesquisas que utilizam esses modelos agrícolas, é comum a necessidade de coleta de grandes quantidades de dados sobre as interações genótipo-ambiente de cada cultura. Assim, faz-se necessária a execução de modelos diversas vezes, alternando-se seus parâmetros, para que todos os fatores a serem analisados sejam atendidos. Neste trabalho, explora-se uma alternativa de programação paralela na linguagem Python para acelerar conjuntos de execuções desses modelos.

2. Fundamentação

Existem distintos pacotes para multiprocessamento disponíveis para Python, porém o uso da linguagem para computação de alto desempenho ainda é pouco frequente. Uma razão

para isso pode estar na forma de como o multiprocessamento é realizado em CPython, implementação principal e mais usada da linguagem, pois o interpretador do CPython conta com o *Global Interpreter Lock* (GIL) em cada um de seus processos, o que faz com que as subrotinas concorrentes sejam literalmente desativadas, segundo [PythonDocs 2017].

Mesmo com as limitações relacionadas ao processamento paralelo em Python, opções como o módulo *threading* e o pacote multiprocessing podem trazer bons resultados dependendo da finalidade do algoritmo em questão. O módulo *threading*, baseado na interface de *threads* da linguagem Java, oferece uma solução de mais alto nível para manipulação de *threads*, abstraindo a complexidade da implementação de funcionalidades mais refinadas no módulo Thread, seu antecessor. Já o pacote multiprocessing oferece simultaneidade local e remota, pois são utilizados subprocessos em vez de *threads*, evitando-se assim o bloqueio gerado pelo GIL no interpretador. Isso possibilita uma melhor utilização dos processadores de uma mesma máquina, pois passa a ser possível a distribuição dos vários subprocessos para os processadores de forma mais homogênea, o que melhora significativamente o desempenho do algoritmo que utiliza multiprocessing.

2.1. Modelos agrícolas

Os modelos agrícolas são softwares que simulam o crescimento e o desenvolvimento das plantas. Os dados sobre o clima, o solo e o manejo das culturas são processados para prever o rendimento, data de maturidade, eficiência dos fertilizantes e outros elementos da produção agrícola. Os cálculos nos modelos de culturas baseiam-se no conhecimento existente da física, fisiologia e ecologia das respostas das culturas ao meio ambiente ao qual estão inseridas [Dourado-Neto et al. 1998].

Para sua execução, um modelo necessita da entradas de alguns dados que servirão de parâmetros para as equações matemáticas descritas no algoritmo e que representam algum fator de influência no processo de desenvolvimento da planta. Nos modelos desenvolvidos na UFSM, mais especificamente no modelo PhenoGlad, é necessário que se informe como entrada a localidade (onde se desenvolverá a cultura) e seus respectivos dados de temperatura (valores máximos e mínimos). Além disso, em tempo de execução, o modelo requer informações referentes ao dia e ano que se deseja realizar a simulação e também algumas características sobre a cultivar e seu ciclo de desenvolvimento. Ao final da execução, o modelo entrega como saída um arquivo de texto com dados tabulados representando o desenvolvimento diário da planta durante todo período simulado.

Para execuções do modelo PhenoGlad com uma única configuração de dados de entrada, o tempo de execução não é um fator problemático. Contudo quando se deseja realizar um processamento que consiga abranger várias datas, cultivares e regiões, passam a ser necessários o uso de *scripts* para a automação desse processo, os quais acabam por demandar um considerável tempo de execução, pois realizam diversas simulações intercalando diversas opções de entrada, alcançando um tempo total de processamento na unidade de horas.

3. Paralelização

A solução proposta baseia-se na melhoria do desempenho do *script* Python que faz a automação das múltiplas execuções dos modelos agrícolas, alternando os dados de entrada referentes ao local de plantio, dia de início da simulação e ano de início da simulação. Os

dados resultantes das múltiplas simulações são utilizados para realizar o zoneamento da cultivar de gladíolo para as inúmeras regiões aptas ao seu cultivo no Rio Grande do Sul. Sendo assim, o número de execuções do modelo Phenoglad segue o seguinte padrão: para cada um dos 55 anos de dados meteorológicos, executa-se uma simulação para cada um dos 365 deste ano, multiplicado pelo número total de regiões.

Ao observar o tempo gasto, na escala de horas, para a realização das múltiplas execuções do modelo PhenoGlad, pode-se perceber a necessidade da otimização de tal processo. Com a implementação do *script* utilizada até então, não era possível o aproveitamento de todo o recurso de processamento da máquina, pois o *script* era executado em um único processo, saturando apenas um dos *cores* do processador, deixando os demais ociosos. Pensando nisso, encontrou-se no pacote multiprocessing do Python uma alternativa para a melhoria do desempenho do *script* por meio do paralelismo de processos.

O *script* pode ser dividido em 3 partes, são elas: chamada de execução de simulações, filtragem dos dados dos arquivos de saída de cada simulação e contagem de danos sofridos pela planta. Ao realizar uma análise do comportamento do *script*, percebe-se que a maior carga de trabalho concentra-se da parte que faz as chamadas de execução do modelo, consumindo em média 98,23% do tempo de execução total do *script*. A porcentagem de tempo gasto com a filtragem e a contagem é de 1,76% e 0,0033% respectivamente.

A implementação de uma versão paralela do *script* Python, utilizando o pacote multiprocessing, somente foi possível graças à independência existente entre cada execução do modelo agrícola. O processo de paralelização baseou-se na distribuição das múltiplas execuções entre processos distintos, por meio do objeto Pool do multiprocessing, processando assim mais de uma simulação simultaneamente, não havendo memória a ser compartilhada e nem regiões críticas entre os processos, pois cada execução do modelo escreve os resultados em um arquivo diferente.

4. Experimentos e Resultados

Foram realizados testes variando os dados de entrada referentes ao dia e ano de início da simulação e limitando a execução para apenas uma das regiões produtoras de gladíolo. Assim, o *script* automatizou execuções do modelo para cada um dos 365 dias de cada ano dos 55 anos de dados meteorológicos disponíveis, totalizando 20.075 (55 x 365) execuções. Para validar os tempos de execução dos testes com diferentes números de processos, foram realizadas 10 execuções do *script* para cada uma das configurações distintas e, posteriormente, feita a média dos tempos obtidos.

O *hardware* utilizado para a execução dos testes é um computador *desktop*, onde geralmente as simulações e execuções de *scripts* que envolvem o modelo são realizadas. A máquina é equipada com um processador Intel Core i5-2410M de 2.30GHz com 2 núcleos e 4 *threads*, 6 GB de RAM DDR3 e sistema operacional Debian GNU/Linux 9.

Os resultados para os testes com 2, 4, 6 e 8 processos por *pool* do multiprocessing, podem ser vistos de forma sintetizada na Figura 1. Com os tempos de saída para cada um dos testes pode se observar que a implementação paralela se mostrou bastante eficiente e, desde a execução do *script* paralelo com 2 processos no pool do multiprocessing, pode ser notada uma redução de mais da metade do tempo de execução. Porém, é possível

observar também que a diminuição dos tempos de execução alcança seu limite ao passo que o número de processos se aproxima do número de *cores* do processador.

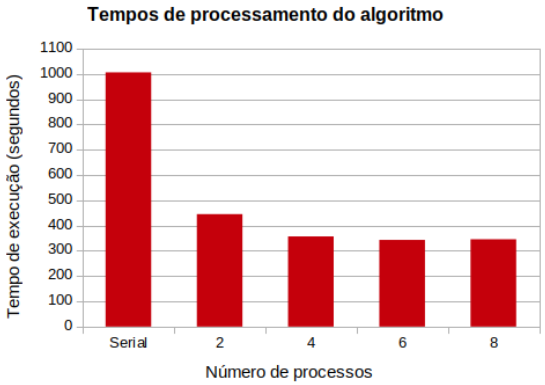


Figura 1. Tempos de execução com o número de processos por *pool*

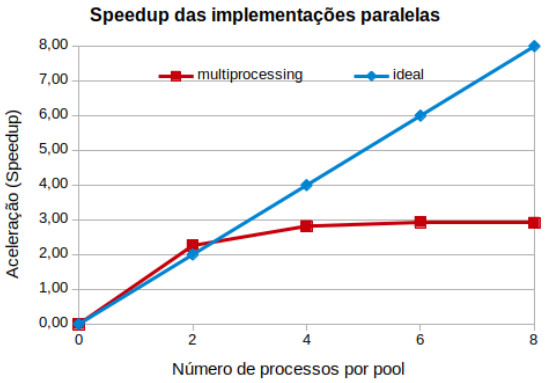


Figura 2. *Speedup* obtido com as implementações paralelas

Com os testes, é possível notar também uma das limitações do paralelismo com processos, que é a sobrecarga que pode ser gerada na fila de escalonamento do processador quando muitos processos, mais que o números de *cores*, são demandados para execução. Esse fator pode ser observado na Figura 2, que apresenta o gráfico do *speedup* atingido. Quando o número de processos chega a 8 (dobro do número de *cores* disponíveis), há uma ligeira queda no *speedup* com relação aos testes em que o número de processos não extrapola o número de cores do processador.

5. Considerações Finais

Os resultados obtidos se mostraram satisfatórios e o paralelismo com processos, utilizando o pacote multiprocessing do Python, se revelou uma solução viável para a otimização dos *scripts* que automatizam as múltiplas execuções dos modelos agrícolas. Algumas limitações do paralelismo com processos foram notadas. Entretanto, para a utilização do *script* para a coleta de dados, elas podem ser contornadas com uma administração do número de processos de cada *pool*. Como trabalhos futuros, pretende-se avaliar a escalabilidade do *script* paralelizado, realizando testes mais aprofundados em uma máquina de processamento de alto desempenho, bem como explorar outras aplicações as quais o paralelismo de processos possa ser implementado.

Referências

CropModelsUFSM (2017). Modelos matemáticos de culturas agrícolas da ufsm. `http://www.cropmodels.ufsm.br/sobre/`. Acesso em: 10 Ago. 2017.

Dourado-Neto, D., Teruel, D. A., Reichardt, K., Nielsen, D., Frizzone, J. A., e Bacchi, O. (1998). Principles of crop modeling and simulation: I. uses of mathematical models in agricultural science. *Scientia Agricola*, 55:46 – 50.

PythonDocs (2017). Python 3 documentation. `https://docs.python.org/3/glossary.html#term-global-interpreter-lock`. Acesso em: 10 Ago. 2017.

Implementação de uma Aplicação de Simulação Geofísica em OpenCL

Arthur Mittmann Krause[1], **Matheus da Silva Serpa**[1], **Philippe Olivier Alexandre Navaux**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{amkrause,msserpa,navaux}@inf.ufrgs.br

Resumo. *A indústria energética recorre a aplicações de simulação que são normalmente implementadas em CUDA ou OpenMP, o que reduz a portabilidade do programa e prende a empresa a um fabricante específico de hardware. Neste trabalho é apresentada uma implementação no padrão aberto OpenCL de um software de simulação geofísica que apresenta desempenho comparável à versões equivalentes em CUDA e OpenMP.*

1. Introdução

Há quase duas décadas, as GPUs são cada vez mais usadas para aceleração de aplicações de propósito geral massivamente paralelas (GPGPU), graças ao grande número de unidades de processamento e largura de banda da memória desses dispositivos [Nickolls and Dally 2010]. Um grande facilitador da popularização de GPGPU são as APIs que permitem aos programadores não precisarem abstrair seus problemas em operações gráficas para que executem em GPUs, entre elas a mais conhecida é CUDA.

Uma outra alternativa é o OpenCL, um *framework* para desenvolvimento de programas não apenas para CPUs e GPUs, mas também para DSPs, FPGAs e demais dispositivos [Stone et al. 2010]. OpenCL especifica uma linguagem baseada em C, um modelo de hierarquia de memória e APIs para controlar esses dispositivos, e principalmente uma interface de programação paralela. A principal vantagem do OpenCL é que o código é portável, permitindo que um mesmo código possa ser executado tanto em uma GPU quanto em uma CPU sem qualquer alteração. Outra vantagem é que ele é um padrão aberto, ou seja, um software feito em OpenCL não estará atrelado a um determinado fabricante de hardware como no caso do CUDA.

Uma classe de algoritmos muito apropriada para paralelização em GPUs são os códigos de estêncil. Nesse tipo de código, uma operação é realizada em cada um dos elementos de um arranjo de forma independente, podendo ser realizadas em paralelo, obtendo assim grande eficiência quando executados em GPUs.

Neste trabalho, uma paralelização em OpenCL de uma aplicação de modelagem geofísica baseada em estêncil é apresentada, assim como o seu desempenho em comparação a versões paralelizadas em CUDA para GPUs e em OpenMP para CPUs.

2. Aplicação e Paralelização com OpenCL

Enquanto os combustíveis fósseis ainda são necessários para atender a demanda energética da sociedade, as empresas de energia precisam executar escavações com um custo que chega a centenas de milhões de dólares, e possuem uma precisão menor do que

50%. Essas empresas recorrem a softwares de simulação baseados em computação de alto desempenho como uma forma de reduzir custos e riscos.

$$\frac{1}{V^2}.\frac{\partial^2 p}{\partial t^2} = \nabla^2 p \tag{1}$$

Foi fornecida pela Petrobras uma aplicação geofísica que simula a propagação de uma *wavelet* ao longo do tempo através da resolução da equação isotrópica de propagação da onda (Equação 1). O laço principal do programa executa por um número de passos de tempo parametrizável, e tem duas partes: a primeira é a inserção da *wavelet* fonte através de um ponto no espaço definido como parâmetro, e a segunda é a propagação das ondas no meio. Através de uma implementação em OpenMP já existente, foi construída uma versão em OpenCL capaz de executar tanto em GPUs quanto em CPUs. A segunda parte do laço principal é um código estêncil, e foi escrita como um kernel OpenCL. A primeira parte consiste em breves operações não paralelizáveis, e foram estudadas três maneiras de implementá-la, cada uma com suas vantagens e desvantagens:

Como um kernel específico
: Com um kernel apenas para inserir a onda no meio evita-se a transferência de dados do host para o dispositivo, mas cria-se um overhead de compilação e chamada desse kernel.

No kernel da segunda parte
: Desta forma, continua-se com um único kernel e evita-se a transferência de dados do host para o dispositivo, mas no início do kernel é necessário verificar se é o *work-item* responsável por introduzir a onda fonte, o que adiciona um overhead.

Como parte do código do host
: Insere-se a onda fonte no código do host antes de chamar o kernel para a segunda parte. Isso adiciona a latência de leitura e escrita de uma posição de memória que reside no dispositivo, mas simplifica o código do kernel.

O método escolhido foi o terceiro, pois apresentou o melhor desempenho na maioria dos casos de teste para ambas as plataformas. Desta forma, o único kernel da aplicação executa o código estêncil de propagação da onda, calculando todas as posições de z para uma determinada posição no plano xy, para um intervalo de tempo. O cabeçalho do kernel ficou da seguinte forma:

```
__kernel void kernel_cte(global float *U0, global float *U1,
global float *VP0, uint stride, uint nnoi, constant float *g_W,
uint k0, uint k1, float FATMDFX, float FATMDFY, float FATMDFZ)
```

Os dados que precisam ser enviados ao dispositivo podem ser vistos pelo cabeçalho. A maior parte dos dados pertence aos três vetores de float que representam o espaço tridimensional. Dois deles (U0 e U1) representam a amplitude da onda em cada ponto para passos de tempo subsequentes, e o outro (VP0), contém a velocidade de propagação da onda no ponto, totalizando três floats para cada ponto do espaço. Foi utilizada apenas a memória global do dispositivo. Após o término da execução, apenas um desses vetores precisa ser lido de volta pelo host.

3. Desempenho

O desempenho da aplicação foi testado tanto com GPU quanto com CPU como plataforma de execução. Os resultados foram comparados à implementações equivalentes em CUDA e OpenMP. Como demonstrado em Serpa et al. [Serpa et al. 2017], o desempenho desta aplicação com OpenMP depende fortemente da ordem dos laços no laço principal, por alterar o padrão de acesso à memória e consequentemente a eficiência da memória cache. Portanto, o desempenho foi comparado tanto com uma versão simples, com a ordem dos laços inalterada, quanto com a versão demonstrada como mais eficiente.

3.1. Metodologia

O ambiente de execução consistiu em uma máquina que possui uma GPU NVidia Tesla P100 e dois processadores Intel Xeon E5-2699 v4, cada um com 22 núcleos operando em 2.2GHz e com *Hyper-threading* habilitado. A máquina possui o sistema operacional Ubuntu com o kernel do Linux 4.4.0-104 instalado. A versão do OpenCL utilizada foi a 1.2, a do CUDA foi a 9.0 e a do GCC foi a 5.4.0. A metodologia utilizada nesse trabalho se baseou na definição de um Projeto Experimental (*Design of Experiments*), tendo como fatores a API e a dimensão do espaço tridimensional de simulação. O número de passos de tempo foi fixado em 500. O experimento consiste em um projeto de fatorial completo, onde todas as combinações de fatores são executadas 15 vezes de forma aleatória, fazendo com que anomalias afetem estatisticamente de forma homogênea as diferentes combinações.

3.2. Resultados

A Figura 1 detalha os tempos de execução da implementação em OpenCL executada na GPU comparada à versão em CUDA, para os quatro tamanhos de entrada. A Figura 2 é semelhante, mas compara o desempenho na CPU com as versões em OpenMP.

A implementação em OpenCL apresentou um desempenho levemente inferior à versão em CUDA. Para um tamanho pequeno de entrada, o tempo de execução com OpenCL foi 80% maior, mas com a dimensão x aumentada em quatro vezes, o tempo com OpenCL chega a ser menor do que o com CUDA. Para tamanhos maiores, o desempenho com CUDA é superior, e torna-se cada vez melhor conforme aumenta-se o tamanho da simulação. Era esperado que o desempenho com CUDA fosse superior, pois diversos trabalhos como Fang et al. [Fang et al. 2011] mostram que a performance do CUDA pode ser até 30% melhor do que o OpenCL para implementações equivalentes da mesma aplicação.

Utilizando as CPUs como dispositivo de execução, o desempenho depende fortemente das dimensões da simulação. Para um tamanho pequeno, o tempo de execução é semelhante à melhor versão com OpenMP e 75% melhor que a versão padrão. Com os tamanhos médios, o desempenho é levemente pior do que a versão com OpenMP otimizada, mas aumenta a diferença para a versão padrão. Com uma entrada muito grande, a implementação em OpenCL apresenta um tempo de execução muito semelhante à OpenMP padrão, enquanto a OpenMP otimizada é 8.3x melhor que ambas. Este comportamento é esperado porque o desempenho dessa aplicação em CPUs é diretamente relacionado com o aproveitamento da memória cache, e é sabido que a versão OpenMP padrão faz um uso ineficiente da mesma, enquanto a OpenMP otimizada utiliza-a da melhor maneira possível. A implementação em OpenCL, como foi realizada de forma genérica tanto

para GPUs como CPUs, não toma nenhuma providência para melhor utilizar a cache, e acaba por depender totalmente do escalonamento dos *work-items* feito pelo *runtime* do OpenCL.

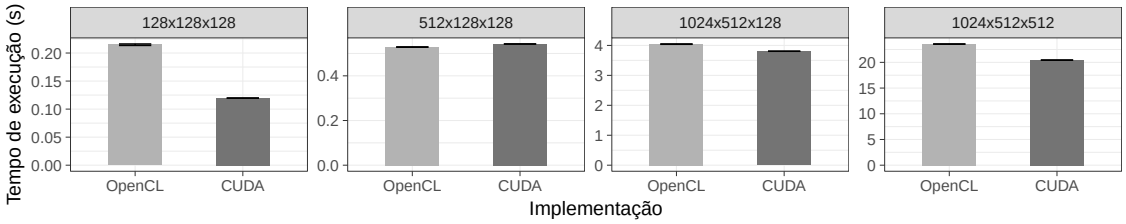


Figura 1. Tempos de execução das implementações baseadas em GPU

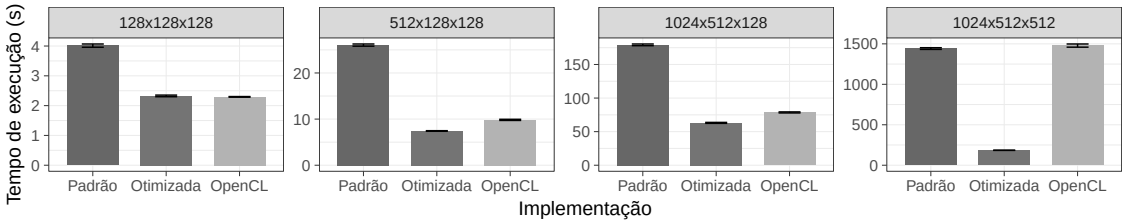


Figura 2. Tempos de execução das implementações baseadas em CPU

4. Conclusão

Este trabalho relata a implementação de uma versão em OpenCL de uma aplicação geofísica e analisa seu desempenho com uma GPU e CPU como plataformas, comparando à versões equivalentes em CUDA e OpenMP. O desempenho pode ser considerado satisfatório pois em muitos casos foi semelhante às outras versões, possuindo uma portabilidade ausente nas demais.

Como trabalho futuro, pretende-se adaptar a aplicação para utilizar as memórias específicas e mais rápidas das GPUs, e estudar como garantir um padrão de acesso à memória que forneça um melhor uso da cache quando executada em uma CPU.

Referências

Fang, J., Varbanescu, A. L., and Sips, H. (2011). A comprehensive performance comparison of cuda and opencl. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 216–225. IEEE.

Nickolls, J. and Dally, W. J. (2010). The gpu computing era. *IEEE micro*, 30(2).

Serpa, M. S., Cruz, E. H., Diener, M., Krause, A. M., Farres, A., Rosas, C., Panetta, J., Hanzich, M., and Navaux, P. O. (2017). Strategies to improve the performance of a geophysics model for different manycore systems. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 49–54. IEEE.

Stone, J. E., Gohara, D., and Shi, G. (2010). Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73.

Implementação do Modelo Eta com CUDA

Alex Lima de Mello[1]**,Henrique Gavioli Flores**[2]**, Marcelo Trindade Rebonatto**[12]**, Carlos amaral Holbig**[12]

[1]Instituto de Ciências Exatas e Geociências – Universidade de Passo Fundo (UPF)
Passo Fundo – RS – Brasil

[2]Programa de Pós-Graduação em Computação Aplicada
Universidade de Passo Fundo (UPF) – Passo Fundo, RS, Brasil

{122596,119694,rebonatto,holbig}@upf.br

Resumo. *Eta é um modelo de previsão meteorológica e climática criado nos anos 70 e utilizado até hoje. Modelos de previsão necessitam de uma grande quantidade de dados de entrada e tendem a possuir um tempo de execução consideravelmente grande. O intuito deste trabalho é avaliar a viabilidade de CUDA como aprimoramento ao modelo Eta, reduzindo o tempo de processamento e mantendo a integridade dos resultados do modelo.*

1. Introdução

Modelos atmosféricos são modelos matemáticos construídos com base no conjunto de equações dinâmicas que governam os movimentos atmosféricos [Flores 2016]. Tais modelos podem ser usados para prever o clima e a meteorologia de uma determinada área

O Eta é um modelo de previsão atmosférica voltado para o uso de pesquisa e decisões operacionais. Este modelo é descendente do modelo Hydrometeorological Institute and Belgrade University (HIBU), após várias atualizações foi reescrito em 1988 para usar a coordenada vertical Eta. [INPE/CPTEC]

Devido à quantidade de informações e a complexidade matemática do modelo, a execução do Eta implica em um alto custo, tanto em tempo quanto em recursos computacionais. Uma das formas de diminuir o tempo de execução do modelo é explorar outros recursos computacionais, como por exemplo GPUs.

2. Materiais e Métodos

O modelo Eta é atualmente escrito em Fortran 90, contando com alguns trechos de bibliotecas escritos em Fortran 77. A paralelização do modelo é realizada utilizando MPICH, uma implementação da tecnologia Message Passing Interface (MPI).

Os processos criados para a execução do modelo são definidos como tarefas de previsão ou servidores de entrada e saída (servidores de I/O), a quantidade de cada uma pode ser definida individualmente pelo usuário. Os servidores de I/O serão responsáveis por armazenar os resultados obtidos pela execução do modelo, enquanto as tarefas de previsão são responsáveis pelo cômputo do modelo.

A área a ser processada pelo Eta é definida por uma matriz, com a quantidade de elementos no eixo vertical e horizontal definidos nos parêametros do experimento sendo realizado. A matriz é sobreposta no mapa terrestre, tendo como ponto central a latitude

e longitude, também definidas no experimento. O tamanho de cada elemento da matriz é definido pela resolução do modelo em quilômetros. A área total a ser processada é dividida em subáreas, das quais cada tarefa de previsão é responsável por uma delas.

Após cada fase de cálculos, os dados são enviados para o servidor de I/O, que ficará responsável por gravar as informações em disco enquanto as tarefas de previsão prosseguem para a próxima iteração, se disponível, na próxima vez será utilizado um servidor de I/O diferente. Caso seja definido zero servidores de I/O, a gravação será efetuada pelas próprias tarefas de previsão, todas gravam os dados em um mesmo arquivo.

Foi realizada uma análise do código, buscando identificar atividades computacionalmente intensivas, com o intuito de identificar pontos de paralelismo. Foi escolhida a tecnologia CUDA para tentar obter um ganho de desempenho sem alterar a lógica utilizada pelo modelo. Apesar de acreditarmos que existam melhorias em relação as operações matemáticas e meteorológicas, neste trabalho nos concentramos apenas no aspecto computacional.

Dos pontos de paralelismo identificados, foram escolhidos 3 pontos para a implementação com CUDA, os pontos escolhidos realizam multiplicação entre valores de três diferentes matrizes, tarefa favorável à execução paralela pela GPU.

Com a utilização de CUDA, a paralelização no Eta agora é composta por 2 níveis: O primeiro MPI, dividindo a área a ser computada entre diferentes processos; o segundo CUDA, paralelizando os cálculos de complexidade elevada.

3. Implementação

Dentro dos blocos, as threads possuem identificadores únicos (ID) para cada dimensão, são eles threadIdx%x, threadIdx%y e threadIdx%z, para as dimensões X, Y e Z respectivamente. Utilizando a ID da thread, a ID do bloco (blockIdx%x, blockIdx%y e blockIdx%z) a que pertence e conhecendo o tamanho do bloco, definido por blockDim em cada direção, é possível acessar as threads de forma contínua, independentemente do bloco a que pertencem.

Os índices da matriz são calculados com base na ID de cada thread. As threads que possuem valor do índice menor que o início, ou maior que o fim de cada dimensão da matriz são finalizadas, restando uma para cada elemento a ser trabalhado.

Dentro do arquivo fonte VTADV foram criados 3 kernels, responsáveis por conter os códigos a serem executados no device (GPU). As threads criadas pela mesma CPU competem pelo uso da GPU, sendo que, em cada loop da fase de cálculos, dois dos kernels são executados apenas uma vez, enquanto o outro é chamado duas vezes.

Para efetuar os cálculos, cada kernel é chamado, tomando como parêametros uma cópia das matrizes a serem trabalhadas e as coordenadas referentes aos índices a serem calculados pelo processo. Para cada dimensão da matriz existem duas variáveis, uma correspondendo ao índice inicial da dimensão e outra correspondendo ao índice final.

Para os testes da implementação do Eta utilizando CUDA foram utilizados 20 computadores com CPU Intel Core i7-3770 de 64 bits, com um clock de 3,40 GHz, com 4 nécleos físicos e 4 lógicos; 8GiB de RAM (Random-access memory); Sistema operacional Ubuntu versão 16.04 LTS para processadores de 64 bits; GPU GeForce GT 630 da

Nvidia, possuindo 384 CUDA cores, 2 GiB de memória, interface 64 bit-DDR3, com 14.4 Gigabytes de largura de banda por segundo [NVIDIA a]. Como servidor de I/O foi utilizado um computador com CPU Intel Core i7 920 de 64 bits, com um clock de 2,66 GHz, 4 cores e 8 threads; 8 GiB de RAM; Sistema operacional Linux, distribuição Ubuntu versão 16.04 LTS para processadores de 64 bits; GPU GeForce GTX 770 da Nvidia, possuindo 1536 CUDA cores e 2 GiB de memória, interface 256-bits GDDR5 e largura de banda de 224.3 Gigabytes por segundo [NVIDIA c]. As máquinas se comunicam usando uma rede FastEthernet.

A GPU utilizada possui limite de 1024 threads por bloco, dimensões máximas de 1024,1024,64 para x, y e z respectivamente, e warp size de 32, significando que independentemente do tamanho do bloco, o némero de threads utilizadas será sempre méltiplo de 32. [NVIDIA d]

Foi utilizada a versão 17.10 do compilador PGI edição comunitária, que pode ser obtida de forma gratuita no pacote disponibilizado pela NVIDIA , com licença para 90 dias, ou por um ano para estudantes de áreas relacionadas. [NVIDIA b]

Para a compilação do modelo Eta utilizando CUDA, é importante verificar que esteja instalado o driver CUDA, disponibilizado pela Nvidia. Deve-se acrescentar a flag "-Mcuda" ao arquivo responsável por controlar as flags de compilação (makefile).

4. Resultados

Com o objetivo de analisar o ganho de desempenho da implementação realizada, foram realizados testes com três áreas diferentes, denominadas: GRANDE(251x581), MÉDIA(181x349) e PEQUENA(101x159), mantendo a resolução de 10Km em todas. Para os testes foram utilizados até 80 processos e um servidor de I/O. O período realizado foi de 12 horas de previsão, compreendido a partir de zero horas do dia 01/01/2009. Cada um dos testes foi repetido 20 vezes, o tempo médio e o coeficiente de variação foram então calculados. Os coeficientes de variação variam de aproximadamente 0,72% à 3.9%, mostrando que não ocorreu interferência significativa de valores externos durante a execução dos testes. Como se pode esperar, o coeficiente de variação tende a aumentar para os menores tempos de execução.

Os testes se dividem em dois grupos: A implementação original do modelo Eta e a implementação com CUDA. A Tabela 1 compara o tempo de execução em segundos dos testes realizados.

Tabela 1. Tempo médio de execução em função do número de Processos

Processos/horas de previsão	PEQUENA	MÉDIA	GRANDE
80 processos	191,67	357,67	582,87
80 processos com CUDA	194,65	361,07	588,05
60 processos	169,56	332,20	577,09
60 processos com CUDA	166,32	330,77	573,35
40 processos	144,49	285,24	544,04
40 processos com CUDA	143,15	283,60	543,23

A Tabela 1 apresenta o resultado dos testes preliminares. é possível observar que a

integração de CUDA em relação a execução do modelo sem o uso dessa tecnologia afetou de forma pouco significativa os resultados, tanto positivamente quanto negativamente.

é possível analisar que com a diminuição do número de processos obteve-se um tempo de execução menor que em relação a execução com 80 processos, este resultado se dá pelo overhead dos processos ser maior que o ganho de desempenho obtido.

5. Considerações finais

Os resultados obtidos mostram que a integração de CUDA na implementação atual do modelo ETA resulta em uma mudança pouco significativa nos tempos de execução, em alguns casos resultando em um tempo maior que o da implementação com MPI.

Outro possível ponto de estudo é utilização de outras tecnologias de paralelismo utilizando a GPU, como por exemplo OpenACC, que possui uma forma de utilização similar ao OpenMP. Futuramente, pode-se desenvolver um trabalho com objetivo de integrar a tecnologia CUDA em todos os pontos do modelo Eta que se identifique o benefício do uso da ferramenta, ou até mesmo a criação de um modelo paralelizado apenas com recursos de GPU.

Referências

Flores, H. G. (2016). Estudo e aprimoramento do modelo eta utilizando recursos de computação paralela e distribuída.

INPE/CPTEC. Model | Eta Model. `http://etamodel.cptec.inpe.br/history`.

NVIDIA. NVIDIA GeForce GT 630. `http://www.nvidia.com.br/object/geforce-gt-630-br.html#pdpContent=2`.

NVIDIA. OpenACC: More Science Less Programming. `https://developer.nvidia.com/openacc`.

NVIDIA. Placa de vídeo GTX 770 com GPU Boost 2.0. `http://www.nvidia.com.br/object/geforce-gtx-770-br.html#pdpContent=2`.

NVIDIA. Programming Guide :: CUDA Toolkit Documentation. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities`.

Investigando Técnicas de Otimização e Paralelização Para Simulação de Camada de Mistura de Fluidos Binários

Sherlon Almeida da Silva[1], **Matheus Castro Nicolau da Silva** [2], **Claudio Schepke**[1]

[1]Laboratório de Estudos Avançados – Universidade Federal do Pampa (UNIPAMPA)
97546-550, Alegrete – RS – Brasil

sherlonalmeida@alunos.unipampa.edu.br, claudioschepke@unipampa.edu.br

[2]Grupo de Fenômenos de Transporte Avançado – Universidade Federal do Pampa (UNIPAMPA)
97546-550, Alegrete – RS – Brasil

mathe1s.castro@gmail.com

Resumo. *Simulações numéricas possibilitam aos pesquisadores reproduzir computacionalmente o comportamento de fenômenos naturais. Elas permitem a realização de experimentos muitas vezes inviáveis, porém demandam um alto custo computacional com a realização dos cálculos. Com o objetivo de agilizar a execução de uma simulação de uma camada de mistura de fluidos binários, este trabalho investiga alternativas de otimização e paralelização.*

1. Introdução

Áreas de estudo da matemática e da física deduzem equações diferenciais para representar o comportamento de fenômenos naturais. A partir disso, tais fenômenos podem ser reproduzidos em um computador. As simulações reproduzidas no computador são representações precisas dos fenômenos, e utilizam dados numéricos de entrada para a partir deles poder acompanhar a evolução temporal do problema tratado.

Nos dias atuais a ciência necessita cada vez mais realizar cálculos complexos e simulações que são computadas por máquinas rápidas. Porém, um dos fatores limitantes hoje é a velocidade em que um resultado pode ser retornado. Como exemplo, existe a previsão do tempo, a qual simula o comportamento do clima para descobrir fenômenos como a chuva, furacões, entre outros. Mas esta previsão deve retornar seu resultado antes do fenômeno acontecer, caso contrário seria inútil. Por isso é necessário que os sistemas retornem os resultados em tempo hábil, contornando os problemas de desempenho.

É possível acelerar a execução de uma simulação através de técnicas de programação e utilização de arquiteturas *multi-core* e *many-core*, as quais fornecem grande poder de processamento devido às dezenas, centenas, até mesmo milhares de núcleos de processamento que podem realizar cálculos simultaneamente em paralelo. Uma das alternativas são as unidades de processamento gráfico (GPUs), que estão cada vez mais populares devido ao seu alto desempenho e menor consumo de energia comparado às unidades centrais de processamento (CPUs), suportando alto paralelismo a custos relativamente baixos [Breder et al. 2016]. Ao definir uma arquitetura para executar o programa, também é possível melhorar o desempenho a partir da otimização para memória *cache*, que é uma memória rápida que armazena os dados mais utilizados pela CPU, reduzindo a alta latência de busca por dados e instruções na memória principal.

O objetivo deste trabalho foi investigar o acoplamento das subrotinas mais custosas de um software de simulação de camada de mistura de fluidos binários, e a maneira que o software foi estruturado. A partir disso, foram identificados os principais trechos que podem ser explorados por técnicas de otimização e paralelização, e por fim foi definida a arquitetura alvo e quais técnicas utilizar para o ganho de desempenho.

Este trabalho está organizado da seguinte forma: A Seção 2 apresenta a metodologia deste trabalho. Na Seção 3 são apresentados e discutidos os resultados parciais. Por fim, é exposta na Seção 4 a conclusão deste artigo.

2. Metodologia

Para explorar o potencial paralelo de um programa é necessário identificar as subrotinas que são mais custosas. Neste sentido existem as ferramentas de análise de execução como o GPROF, que coletam dados em tempo de execução, como o tempo, as funções mais requisitadas, entre outros. A partir disso, é possível identificar quais trechos de código são possíveis candidatos a serem otimizados e paralelizados.

O software utilizado é uma continuação do trabalho de [Quirino and Mendonça 2007] e de [Manco 2014] e calcula as equações de Euler utilizando uma matriz como estrutura de dados para representação em 2 dimensões (2D) da camada de mistura de fluidos, representando o domínio físico através de uma malha computacional. Fluidos inseridos com diferentes velocidades criam um ponto de inflexão no perfil vertical de velocidade, o que somado com a perturbação feita por um pulso acústico, propicia a criação de vórtices de Kelvin-Helmholtz no escoamento. Os vórtices aumentam a área de contato entre os fluidos, acelerando o processo de difusão entre os fluidos. Sendo estes um oxidante e um combustível, pode-se realizar experimentos numéricos de combustão. Uma aplicação pode ser verificada no sistema de propulsão Scramjet, da NASA, em que um veículo não tripulado atingiu a velocidade de $12.144 km/h$ por 300 segundos. Cada imagem da simulação é uma representação adimensional da vorticidade no instante de tempo em que se encontra. Por exemplo, na Figura 1 o eixo y representa a vorticidade adimensional, e o eixo x representa o domínio físico da câmara de mistura. O método numérico utilizado no software é denominado

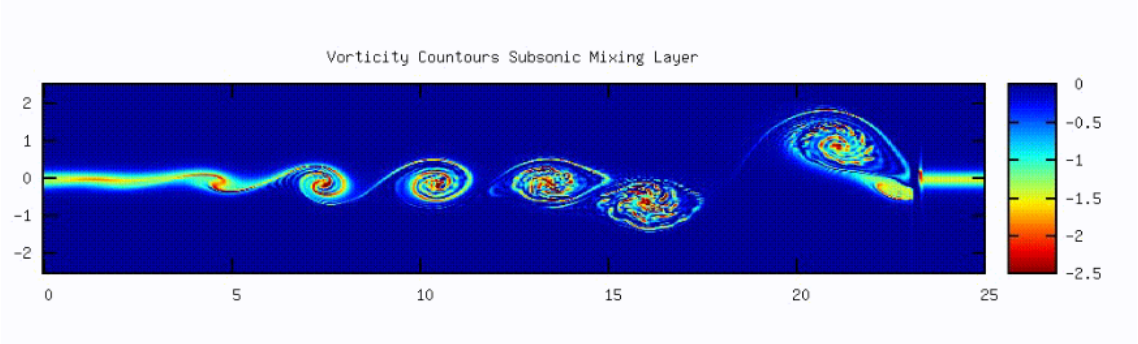


Figura 1. Simulação de Mistura de Fluidos - Vórtices.

Euler Explícito, o que significa que os valores futuros são calculados diretamente a partir dos valores atuais. São gerados arquivos de dados contendo os valores das propriedades físicas medidas na simulação. Com estes dados são geradas imagens para interpretar

estes resultados. A otimização do desempenho para esta aplicação é necessária uma vez que os dados levam horas para serem obtidos.

Outro fator a ser salientado é que o software ainda está em desenvolvimento. Novas equações serão adicionadas em forma de subrotinas, como as equações para reações químicas e combustão. A demora na execução atrasa os testes das novas subrotinas, além do programa ficar mais lento conforme novas funcionalidades forem inseridas, e para obter dados ainda mais precisos, o custo computacional se torna ainda maior.

3. Resultados Parciais e Discussão

A análise detalhada do programa juntamente com a análise de execução do GPROF forneceram informações relevantes para futuras otimizações e paralelizações. Como é possível ver na Tabela 1, foram identificadas as 12 subrotinas mais custosas de um total de 78 subrotinas. Juntas, estas 12 subrotinas somam aproximandamente 86% da execução do programa. Com a coleta das subrotinas mais custosas, tornou-se necessário

Tabela 1. Custo de execução por Subrotina coletados com GPROF.

Subrotina	Código	Perc(%)
eulernonlinear(...)	eqeuler.f90	11.13
lddfiltery(...)	filtering.f90	11.02
strechingx(...)	eqeuler.f90	10.57
lddfilterx(...)	filtering.f90	8.55
dudx(...)	diff.f90	8.36
dudytau(...)	diff.f90	7.68
dudy(...)	diff.f90	7.33
dudxtau(...)	diff.f90	7.27
rhs_euler(...)	rhs.f90	6.18
rk_steep(...)	rk.f90	2.96
dy4(...)	diff.f90	2.46
dx4(...)	diff.f90	2.35

compreender o fluxo de execução do programa de forma detalhada, como por exemplo a comunicação entre as subrotinas, a passagem das variáveis e parâmetros, e o que cada uma computa. Todo o fluxo de execução das subrotinas mais custosas estão na Figura 2. A Figura 2 contém um diagrama que apresenta o início da execução no programa euler.f90, o qual chama as demais subrotinas do programa. Cada nó do diagrama possui o nome do código e a subrotina chamada, bem como a porcentagem de execução. Após esta análise foi possível observar o comportamento do programa e os possíveis trechos de código a serem otimizados e paralelizados. O programa conta com um grande número de laços de repetição que operam sobre matrizes com cálculos independentes, viabilizando a paralelização. Estes laços de repetição também podem ser otimizados para varredura contígua na memória, para aproveitar a localidade espacial e temporal da memória *cache* a partir de técnicas como *Loop Interchange*, que reordena os laços aninhados.

4. Conclusão e Trabalhos Futuros

Simulações computacionais demandam um alto poder de processamento. Tal recurso é encontrado em arquiteturas *multi-core* e *many-core*, uma vez que estas máquinas

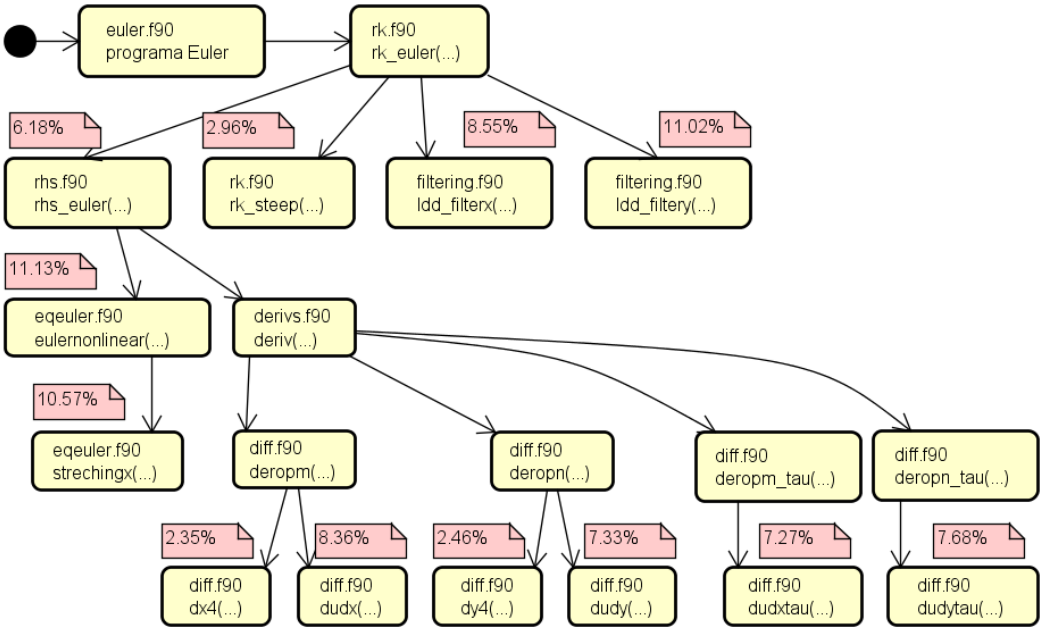


Figura 2. Fluxo de execução do programa.

dispõem de diversos núcleos de processamento. Uma das principais etapas para o aumento do desempenho de um programa é identificar o foco de maior custo computacional e aplicar técnicas de otimização e paralelização. Esta identificação de trechos de código custosos pode ser feita com ferramentas de análise de execução, como o GPROF.

Devido à análise detalhada do código, como trabalhos futuros serão implementadas versões paralelas com a utilização de programação híbrida com as interfaces de programação OpenMP (*Open Multi-Processing*) para CPUs (*Central Processing Units*) e CUDA (*Compute Unified Device Architecture*) para GPUs (*Graphics Processing Units*). Também serão implementadas otimizações para memória *cache* a partir da reordenação da varredura de laços de repetição, como a técnica *Loop Interchange*.

Agradecimentos

Este trabalho foi realizado com recursos do PROBIC/FAPERGS 2017 e pela agência de fomento CNPq via Edital Universal Processo N. 457684/2014-3.

Referências

Breder, B., Charles, E., Cruz, R., Clua, E., Bentes, C., and Drummond, L. (2016). Maximizando o uso dos recursos de gpu através da reordenação da submissão de kernels concorrentes. In *Anais do WSCAD 2016 (XVII Simpósio em Sistemas Computacionais de Alto Desempenho)*, pages 98–109, Aracajú. SBC.

Manco, J. A. A. (2014). Condições de contorno não reflexivas para simulação numérica de alta ordem de instabilidade de kelvin-helmholtz em escoamento compressível. Master's thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos.

Quirino, S. F. and Mendonça, M. T. (2007). Direct numerical simulation of compressible shear layer with heat source. In *19th International Congress of Mechanical Engineering*, Brasília.

MOBI: Um Módulo para Monitorar Aplicações Big Data em Ambientes de Nuvem Heterogêneos

Jobe D. Santos[1], **Kassiano J. Matteussi**[1], **Paulo R. R. de Souza Junior**[1], **Claudio R. Geyer** [1]

[1]Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de informática - GPPD
Caixa Postal 15.064 - 91.501-970 - Porto Alegre, RS, Brasil
{jobe.dylbas, kjmatteussi, prrsjunior, geyer}@inf.ufrgs.br

Resumo. *Aplicações Big Data são amplamente utilizadas por organizações que buscam obter informações em meio a bases de dados complexas. Neste artigo, apresenta-se o MOBI: um módulo para monitorar aplicações Big Data em ambientes de nuvem heterogêneos. MOBI possui comportamento não-intrusivo e permite minimizar ou anteceder problemas de interferência durante o processamento das aplicações, levando a melhor utilização dos recursos computacionais.*

1. Introdução

O número de dispositivos conectados à internet aumenta consideravelmente a cada ano, e os dados produzidos por estes sensores, *smartphones*, entre outros, têm sido considerados cada vez mais valiosos pelas organizações. Estes dados na forma crua podem não possuir significado, mas analisados podem potencializar a tomada de decisões estratégicas e gerar informações significativas nos mais variados segmentos.

Monitorar as estruturas que processam tais informações se tornou crucial para manter a eficiência e integridade dos ambientes de processamento. Porém, o monitoramento possui desafios, como por exemplo analisar se os *frameworks* Big Data estão processando o dado com a miníma interferência possível. Assim, este artigo apresenta o MOBI, um módulo de monitoramento de aplicações *Big Data*, que considera as diferentes necessidades dos ambientes heterogêneos de larga escala.

2. Estado da Arte

Ambientes de nuvem fornecem segurança, eficiência, flexibilidade e escalabilidade para suportar o processamento de grande volumes de dados requisitados por aplicações e *frameworks* Big Data [Assunção et al. 2015]. Entretanto, em ambientes não controlados podemos ter uma sobreutilização de recursos gerando gargalos de processamento, ou uma subutilização, isto é, existem recursos disponíveis que não estão sendo utilizados.

Para suprir tais problemas mecanismos de monitoramento podem ser adotados, permitindo por exemplo, a avaliação sobre a condição da infraestrutura. No mercado existem diversos monitores, tais como: Nagios [Luchian et al. 2016], DARGOS [Povedano-Molina et al. 2013] e Ganglia [Massie et al. 2004] - cujo foco são *clusters* e *grids*. Esses monitores propiciam diversas métricas para o melhor controle sobre o ambiente e, por meio de uma interface *web* com suporte *near real-time* fornecem a visualização dos dados. Contudo, muitos desses serviços e aplicações de monitoramento não são adaptativos a multi-ambientes [Fatema et al. 2014] nem a ambientes heterogêneos, ou seja, compostos por diversos tipos de dispositivos.

O monitoramento deve ser feito de forma não-intrusiva, de modo que não interfira na execução das aplicações, garantindo a qualidade do Serviço (QoS) para com o

ambiente. Para que isso seja possível, o *trade-off* entre qualidade de monitoramento e interferência nos recursos deve ser considerado, ou seja, mecanismos devem ser adotados para minimizar tal interferência, caso ela exista. Além disto, sistemas de monitoramento devem ser escaláveis quanto a sua capacidade de monitorar. Em [Andreolini et al. 2013] estes critérios são analisados e métricas são propostas para avaliar o monitoramento de recursos em alta escala.

3. Arquitetura Smart: Uma Visão Geral

O objetivo da arquitetura SMART [dos Anjos et al. 2015] é simplificar o processamento Big Data. A SMART é formada por seis grandes módulos, conforme representado na Figura 1, são eles: *Global Collector, Global Dispatcher, Storage, Core Engine, Global Aggregator e Central Monitoring*.

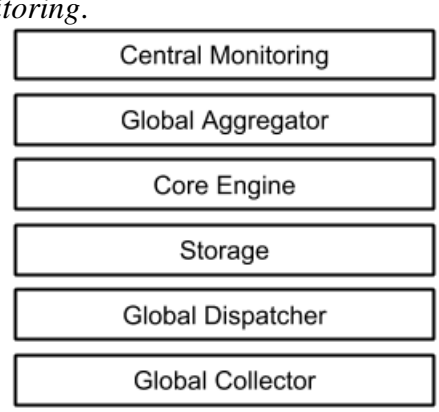


Figura 1. Smart Architecture

(i) O *Global Collector* é responsável por coletar e manter a integridade dos dados fornecidos pelos mais diversos dispositivos; (ii) No *Global Dispatcher* é onde os dados coletados são divididos em diferentes filas para assim serem distribuídos nos servidores de acordo com a disponibilidade dos mesmos e então armazenados no (iii) *Storage*; Os dados são processados pelo *Core Engine* gerando as informações relevantes que são levadas pelo (v) *Global Aggregator* ao usuários finais que acessam estes através do (vi) *Central Monitoring*.

A arquitetura SMART precisa ser constantemente monitorada para minimizar possíveis problemas com interferência ligadas a utilização de recursos que, podem implicar em perda de desempenho por parte das aplicações que estiverem em execução. Então, o MOBI é apresentado como componente desta arquitetura fazendo parte do módulo *Central Monitoring*, visando monitorar os recursos do módulo de processamento de dados (i.e. *Core Engine*).

4. MOBI: Arquitetura e Implementação

O MOBI é composto por cinco módulos: (i) O módulo de Plugins que fica encarregado em adaptar os diversos coletores de dados, como dstat[1], htop[2] e outros ao MOBI; (ii) O módulo Engine é responsável pela captura, inserção no banco de dados e ainda coletar e enviar os dados requisitados pela (iii) REST API, que por sua vez, funciona como um meio de comunicação entre a GUI e a Engine, traduzindo as requisições do usuário; Por

[1]http://dag.wiee.rs/home-made/dstat/
[2]http://hisham.hm/htop/

fim, (iv) GUI é uma interface de usuário onde pode-se requisitar quais recursos e máquinas deseja-se monitorar tanto de forma automática quanto customizada. Sua arquitetura do MOBI pode ser observada na na Figura 2.

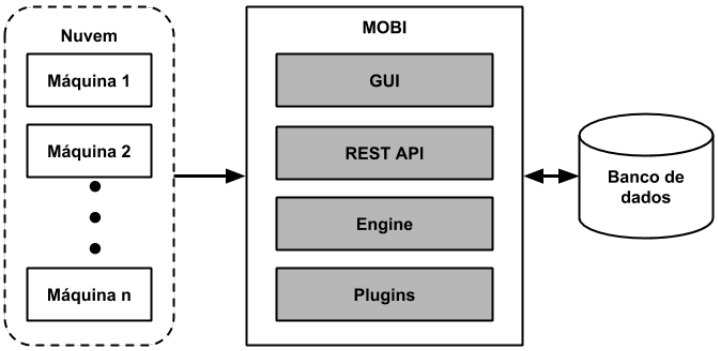


Figura 2. Arquitetura MOBI

As tecnologias empregadas em seu desenvolvimento são abertas e largamente utilizadas por diversos serviços, como Paypal e Uber. Entre as escolhas estão o MongoDB como banco de dados, devido a fácil utilização e do formato JSON por ser compacto, nativamente reconhecido pela linguagem JavaScript e de clara compreensão [Povedano-Molina et al. 2013]. Os serviços do servidor são desenvolvidos com Node.js, escolhido por ser baseado em eventos e por ser *I/O* não bloqueante, que permite otimizar o uso dos recursos do servidor [Chitra and Satapathy 2017].

5. Resultados Preliminares

Atualmente, o usuário ou administradores do sistema podem monitorar e analisar em tempo real a utilização dos recursos computacionais de uma nuvem. Além disso, o usuário pode configurar a ferramenta de monitoramento, setando quais recursos deseja monitorar por meio de uma GUI (Figura 3). Afora isso, é possível exportar os gráficos referente ao uso de recursos de forma instantânea, bem como é possível obter os *traces* contendo tais dados. Isso possibilita que ações sejam tomadas para minimizar questões ligadas aos problemas interferência como o congestionamento de rede e contenção de disco, por exemplo.

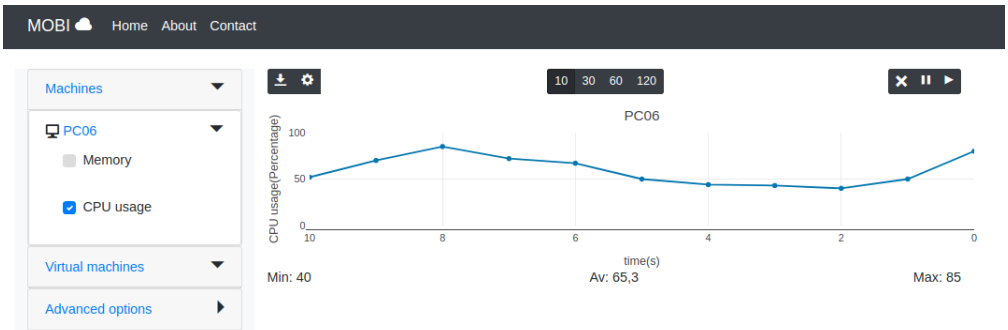


Figura 3. GUI - MOBI

Por fim, futuramente pretende-se avaliar outros servidores de banco de dados e protocolos de comunicação (UDP, TCP e etc.), visando equilibrar a integridade dos dados e sua disponibilidade em tempo real. Além de que para melhor avaliação dos recursos

estuda-se quais serviços são interessantes ou necessários, tais como formato de gráficos, granularidade, exportação de dados e avisos.

6. Considerações Finais

Este artigo apresentou um módulo de monitoramento, o MOBI, o qual utiliza técnicas e ferramentas presentes na literatura com objetivo de minimizar o impacto no desempenho de ambientes heterogêneos de processamento Big Data.

7. Agradecimentos

O presente trabalho foi realizado com o apoio da Pró-Reitoria de Pesquisa - UFRGS - Brasil.

Referências

Andreolini, M., Colajanni, M., Pietri, M., and Tosi, S. (2013). Real-time adaptive algorithm for resource monitoring. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 67–74.

Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., and Buyya, R. (2015). Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79-80(Supplement C):3 – 15. Special Issue on Scalable Systems for Big Data Management and Analytics.

Chitra, L. P. and Satapathy, R. (2017). Performance comparison and evaluation of node.js and traditional web server (iis). In *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*, pages 1–4.

dos Anjos, J. C. S., Assunção, M. D., Bez, J., Carissimi, A., Costa, J. P. C. L., Freitag, F., Markl, V., Fergus, P., Pereira, R., de Freitas, E. P., Fedak, G., and Geyer, C. F. R. (2015). Smart: An application framework for real time big data analysis on heterogeneous cloud environments. In *In proceedings of 15th IEEE International Conference on Computer and Information Technology (CIT-2015), Liverpool, England, UK, October 2015*. IEEE Computer Society.

Fatema, K., Emeakaroha, V. C., Healy, P. D., Morrison, J. P., and Lynn, T. (2014). A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918 – 2933.

Luchian, E., Docolin, P., and Dobrota, V. (2016). Advanced monitoring of the openstack nfv infrastructure: A nagios approach using snmp. In *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, pages 51–54.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 840.

Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., and Foschini, L. (2013). Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *Future Generation Computer Systems*, 29(8):2041 – 2056. Including Special sections: Advanced Cloud Monitoring Systems & The fourth IEEE International Conference on e-Science 2011 — e-Science Applications and Tools & Cluster, Grid, and Cloud Computing.

Otimização da Comunicação em Aplicações Estêncil Paralelas Implementadas com o PSkel no Processador MPPA-256

Bruno Marques do Nascimento[1], **Emmanuel Podestá Jr.**[1], **Márcio Castro**[1]

[1] Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Universidade Federal de Santa Catarina (UFSC) – SC, Brasil

{bruno.mn, emmanuel.podesta}@grad.ufsc.br, marcio.castro@ufsc.br

Resumo. *Neste artigo é proposta uma versão otimizada do framework PSkel para o processador MPPA-256 com base no uso de uma nova biblioteca de comunicação desenvolvida especificamente para esse processador. Ela permite um melhor desempenho na comunicação além de oferecer uma maior abstração para o desenvolvedor. Os resultados mostraram que a implementação proposta possui um desempenho superior em até 8x em relação à implementação anterior.*

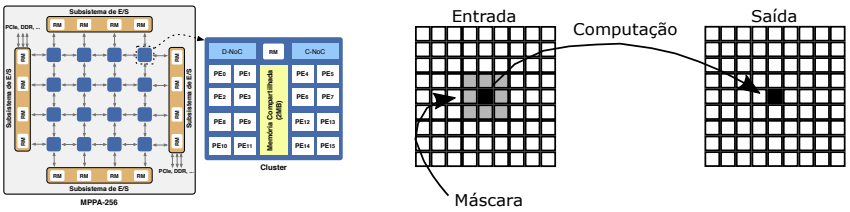
1. Introdução

Processadores *manycore* de baixa potência (*low-power manycore processors*) desenvolvidos recentemente, tais como o Sunway SW26010 [Fu et al. 2016] e Kalray MPPA-256 [Francesquini et al. 2014], apresentam melhor eficiência energética em comparação com processadores de propósito geral atuais. O supercomputador atualmente em primeiro lugar no Top500[1] (*Sunway TaihuLight*) foi o primeiro a empregar este tipo de processador, possuindo 40.960 processadores Sunway SW26010 de 260 núcleos cada. Contudo, desenvolver aplicações paralelas otimizadas para esses processadores é bastante desafiador [Francesquini et al. 2014]. Os núcleos de processamento possuem memórias *cache* não coerentes e são distribuídos em *clusters*, os quais possuem uma memória local de tamanho limitado, e se comunicam por meio de uma *Network-on-Chip* (NoC), caracterizando assim um modelo com espaço de endereçamento compartilhado e distribuído.

Nesse contexto, Podestá *et al.* [Podestá Jr. et al. 2017b] propuseram o uso de esqueletos paralelos para simplificar o desenvolvimento de aplicações nesses processadores. Mais precisamente, foi proposta uma adaptação do *framework* PSkel, o qual implementa o padrão paralelo estêncil, para o processador MPPA-256. O padrão estêncil é um dos padrões mais utilizados em áreas importantes, como física quântica, processamento de imagens e previsão do tempo. Apesar da adaptação proposta ter apresentado um bom potencial, foi identificado que a mesma apresentava gargalos na comunicação via NoC.

O presente artigo apresenta uma versão otimizada da proposta descrita anteriormente através da exploração de uma nova *Application Programming Interface* (API) de comunicação desenvolvida para o MPPA-256, permitindo assim otimizações na comunicação via NoC. As otimizações implementadas permitem ganhos significativos no desempenho e no consumo de energia. Este artigo está organizado da seguinte forma. A Seção 2 apresenta os principais conceitos do processador MPPA-256 e do *framework* PSkel. A Seção 3 discute as otimizações propostas. Os resultados obtidos são apresentados na Seção 4. Por fim, Seção 5 apresenta as conclusões deste trabalho.

[1] http://top500.org



(a) MPPA-256. (b) O padrão estêncil.

Figura 1. Visão geral do MPPA-256 e do padrão estêncil [Podestá Jr. et al. 2017a].

2. Fundamentação Teórica

2.1. MPPA-256

O MPPA-256 é um processador *manycore* desenvolvido pela empresa francesa Kalray. Esse processador possui 256 núcleos de processamento de 400 MHz denominados *Processing Element*s (PEs). Além dos PEs, o MPPA-256 apresenta 32 núcleos dedicados à gerência de recursos denominados *Resource Manager*s (RMs). Os RMs são responsáveis por gerenciar a Entrada e Saída (E/S) e controlar comunicações entre *clusters* e/ou subsistemas. Os PEs são distribuídos em 16 *clusters*, onde cada *cluster* possui 16 PEs e 1 RM. Além disso, o MPPA-256 possui 4 subsistemas de E/S, contendo 4 RMs cada um. Por fim, toda a comunicação no MPPA-256 é realizada por meio de uma NoC *torus* 2D. A arquitetura do MPPA-256 é ilustrada na Figura 1(a).

Trabalhos anteriores mostraram que desenvolver aplicações paralelas otimizadas para o MPPA-256 é um grande desafio [Francesquini et al. 2014] devido a alguns fatores importantes, tais como: o modelo de memória distribuída presente no MPPA-256, a capacidade de memória dentro do *chip* e a comunicação explícita através da NoC. Mais detalhes sobre esses desafios são apresentados em [Podestá Jr. et al. 2017a].

2.2. PSkel

O PSkel é um *framework* de programação em alto nível para aplicações baseadas no padrão estêncil, oferecendo suporte para a execução dessas aplicações em ambientes heterogêneos, incluindo *Central Processing Unit* (CPU) e *Graphics Processing Unit* (GPU). O usuário do *framework* será responsável por definir o *kernel* principal da computação, enquanto o *framework* irá realizar a distribuição de dados e escalonamento de tarefas de forma transparente [Pereira et al. 2015]. O padrão estêncil, ilustrado pela Figura 1(b), funciona da seguinte forma. Para cada elemento de uma estrutura n-dimensional é computado um novo valor relativo aos vizinhos do elemento atual. Os vizinhos de um elemento são determinados pela máscara da computação. Por fim, cada novo valor computado é atribuído à sua célula respectiva em uma estrutura n-dimensional de saída. Em aplicações estêncil iterativas, a estrutura de saída é utilizada como estrutura de entrada de uma nova iteração da aplicação.

3. Versão Otimizada do PSkel para o MPPA-256

A adaptação anterior do PSkel proposta por Podestá *et al.* [Podestá Jr. et al. 2017b], denominada **IPC** neste artigo, utilizava uma API similar à POSIX *Inter-Process Communication* (IPC) para comunicação. Nela, eram utilizados **portais de comunicação** para o envio de dados e o método de ***strides*** para gerenciar explicitamente o envio e recebimento de *tiles*. Porém, o desempenho desta implementação mostrou-se prejudicado pelo elevado tempo despendido nas operações de comunicação. Com o advento da nova API de

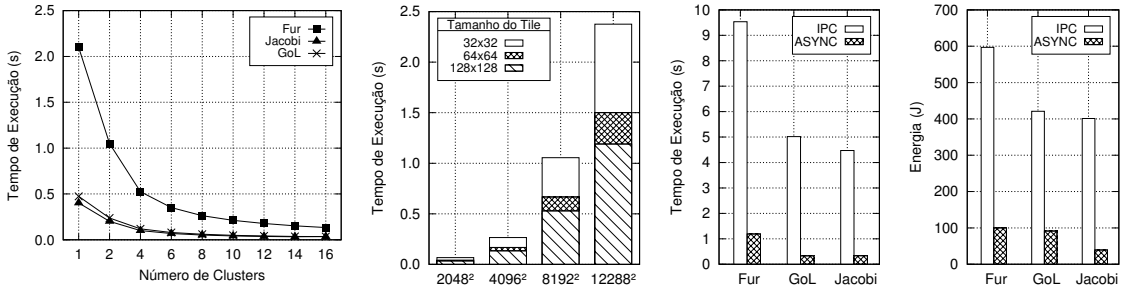
comunicação para o MPPA-256 foi possível mitigar o impacto provocado ao desempenho por estas operações. Esta API trabalha com o conceito de **segmentos**, que possibilita o endereçamento de segmentos de memória externos à memória local do *cluster*. A função `mppa_async_segment_create()` permite criar um segmento de memória no subsistema de E/S ligado à *Low Power Double Data Rate 3* (LPDDR3) ao passo que a função `mppa_async_segment_clone()` permite clonar este segmento remoto na memória local de um *cluster*, resultando em um segmento local que referencia o segmento criado remotamente na LPDDR3. Este segmento criado é posteriormente utilizado no âmbito das funções de transferência de dados disponibilizadas pela API, para endereçar o segmento de memória remoto. Estas funções são capazes de copiar dados locais para o segmento de memória criado, através dos métodos `put` e também copiar dados de um segmento remoto e armazená-los localmente através dos métodos `get`.

Na proposta apresentada neste artigo, denominada **ASYNC**, a responsabilidade pela divisão dos *tiles* foi transferida aos *clusters* e não mais realizada no subsistema de E/S. Além disso, os métodos `put` e `get` são executados apenas pelos *clusters*. Mais precisamente, os métodos utilizados foram `mppa_async_sget_block2d()` e `mppa_async_sput_block2d()`, que são métodos próprios para a manipulação de matrizes e submatrizes, presentes na operação de divisão do *grid* em *tiles* na computação estêncil. Estas facilidades agregadas ao elevado nível de abstração da API demonstram com nitidez as vantagens proporcionadas pelo uso desta nova API no desenvolvimento de aplicações estêncil. Os benefícios em termos de desempenho serão analisados na Seção 4.

4. Resultados Experimentais

Esta seção apresenta os resultados obtidos com a solução proposta, comparando-a com a versão apresentada em [Podestá Jr. et al. 2017b]. Todas as métricas foram obtidas com auxílio de ferramentas disponíveis no MPPA-256. Os dados se referem a execução de uma única iteração das aplicações Fur, GoL e Jacobi, as quais são descritas em mais detalhes em [Podestá Jr. et al. 2017b]. Foram realizadas 5 repetições de cada experimento, computando-se a média aritmética dos valores. A variabilidade dos valores obtidos foi extremamente pequena (desvio-padrão inferior à 1%), pois as *threads* da aplicação são executadas de maneira ininterrupta no MPPA-256.

Na análise da escalabilidade da versão ASYNC, apresentada pela Figura 2(a), foi utilizada uma matriz de tamanho 4.096x4.096 e *tiles* de tamanho 128x128. Foi obtido um ganho de desempenho de até $15,6$x com 16 *clusters* em relação à execução com um único *cluster* (aplicação Fur). Por outro lado, a Figura 2(b) apresenta o impacto do tamanho dos *tiles* sobre o desempenho nessa aplicação. Para obter uma maior precisão nos resultados em relação ao tempo de execução foram utilizadas matrizes de tamanho maior que 2.048x2.048. Além disso, devido à memória limitada, não foram utilizadas matrizes de entrada e *tiles* maiores que 12.288x12.288 e 128x128, respectivamente. Os resultados mostram que o tempo de execução da aplicação diminui à medida em que se aumenta o tamanho dos *tiles*, obtendo-se ganhos de até 2x. Este comportamento é decorrente do melhor aproveitamento da vazão da NoC, realizando menos transferências com maior quantidade de dados. As aplicações GoL e Jacobi também apresentaram comportamentos similares. As Figuras 2(c) e 2(d) apresentam a comparação de tempo e consumo de energia entre as versões ASYNC e IPC. Nesse experimento, foi utilizada uma matriz de entrada de tamanho 12.288x12.288, *tiles* de tamanho 128x128 e 16 *clusters*. Como pode ser observado, os resultados mostram que o tempo de execução da versão ASYNC é, aproximadamente, 8x menor em relação ao tempo da versão IPC. O resultado da ener-



(a) Escalabilidade. (b) *Tiles* vs. tempo (Fur). (c) Tempo. (d) Energia.

Figura 2. Escalabilidade (a) e impacto do tamanho dos *tiles* (b) na versão ASYNC. Comparação do tempo (c) e consumo de energia (d) do ASYNC com a versão IPC.

gia consumida segue um comportamento similar, apresentando uma eficiência energética superior em até 6x a favor da versão ASYNC. A versão IPC possui funções que manipulam dados (no mínimo, $7,1\%$ do tempo total) e distribuições de tarefas (no mínimo, 72% do tempo total), trazendo um impacto negativo sobre o tempo de execução.

5. Conclusão

O desenvolvimento de aplicações otimizadas para processadores *manycore* de baixa potência é bastante desafiador devido a fatores importantes tais como a existência de um modelo de programação híbrido, capacidade limitada de memória no *chip*, ausência de coerência de *cache*, entre outros. Neste artigo foi apresentada uma nova versão otimizada do *framework* PSkel para o processador MPPA-256. Os resultados mostraram que a nova versão obteve ganhos de desempenho de até 8x e uma redução no consumo de energia de até 6x, em comparação com a solução inicial proposta em [Podestá Jr. et al. 2017b]. Como trabalhos futuros, pretende-se implementar o suporte para a execução de aplicações estêncil iterativas, comparar o desempenho e consumo de energia com outros processadores e implementar suporte a matrizes tridimensionais.

Referências

Francesquini, E., Castro, M., Penna, P. H., Dupros, F., de Freitas, H. C., Navaux, P. O. A., and Méhaut, J.-F. (2014). On the Energy Efficiency and Performance of Irregular Applications on Multicore, NUMA and Manycore Platforms. *J. Parallel Distrib. Comput.*, 76:32–48.

Fu, H. et al. (2016). The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences*, 59(7):1–16.

Pereira, A. D., Ramos, L., and Góes, L. F. W. (2015). PSkel: A Stencil Programming Framework for CPU-GPU Systems. *Concurrency and Computation: Practice and Experience*, 27(17):4938–4953.

Podestá Jr., E., Pereira, A. D., Rocha, R. C., Castro, M., and Góes, L. F. W. (2017a). Uma Implementação do Framework PSkel com Suporte a Aplicações Estêncil Iterativas para o Processador MPPA-256. In *ERAD/RS*, pages 395–398, Ijuí, Brazil. SBC.

Podestá Jr., E., Pereira, A. D., Rocha, R. C. d. O., Castro, M., and Góes, L. F. W. (2017b). Execução Energeticamente Eficiente de Aplicações Estêncil com o Processador Manycore MPPA-256. In *Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)*, Campinas, SP.

Otimização de desempenho de software para análise filogenética (ExaML) utilizando a arquitetura CUDA

Marcelo Gomes Martins[1], **Timoteo Alberto Peters Lange**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul
Rua Santos Dumont, 2127 – Bairro Albatroz – 95.520-000 – Osório – RS – Brasil

marcelogm94@hotmail.com, timoteo.lange@osorio.ifrs.edu.br

Resumo. *O aumento das bases genéticas de dados e da complexidade das análises filogenéticas requer maior poder computacional: este artigo apresenta o processo de reimplementação das principais funções da aplicação ExaML para a arquitetura CUDA, demonstrando aspectos relevantes no processo de otimização de uma aplicação de alto desempenho para GPU e apresentando bons resultados na manipulação de grandes entradas de dados.*

1. Introdução

O processo de análise filogenética baseia-se na construção de árvores filogenéticas que, por definição, são diagramas utilizados para ilustrar de forma elegante quais organismos possuem uma estrutura genética semelhante [Vandamme 2009]. Essa estrutura é utilizada para representar um possível histórico evolutivo dos organismos envolvidos em uma análise.

Em 2004, Alexandros Stamatakis apresentou a implementação de um algoritmo para pesquisa de árvores filogenéticas utilizando o método estatístico de máxima verossimilhança: *Randomized Axecelerated Maximum Likelihood* (RAxML) [Stamatakis 2004], programa desenvolvido em linguagem C para operar em ambiente paralelo e distribuído. RAxML teve seu desenvolvimento em um período anterior à popularização[1] do uso das *Graphics Processing Units* (GPUs) como unidades de processamento auxiliares e do surgimento das plataformas CUDA e OpenCL.

Posteriormente, foram apresentados trabalhos relacionados ao RAxML, como a *Phylogenetic Likelihood Library* [Izquierdo-Carrasco et al. 2013] e o ExaML [Kozlov et al. 2015], algoritmo desenvolvido para operar em supercomputadores de forma paralela e distribuída. O objetivo deste artigo é apresentar os principais aspectos envolvidos no processo de paralelização da aplicação ExaML para a arquitetura CUDA, acompanhados dos resultados obtidos.

2. Configuração de ambiente

Para compor o ambiente de desenvolvimento foi utilizado o processador Intel Core I7-7700K (4,5 GHz), 8 GB DDR4, placa gráfica Nvidia GeForce GTX 1060 (1.280 CUDA *cores*), sistema operacional Ubuntu 16.04.3 LTS e CUDA *Toolkit* 9.0.176.

Como teste de validação do modelo, foi utilizado o arquivo **49**[2] (*patterns* de DNA de 48 *taxa* variados), possuindo 628 *patterns*. Para os testes de desempenho da aplicação,

[1]Ainda que em 2005 já existam trabalhos utilizando BrookGPU na paralelização de funções da aplicação [Charalambous et al. 2005], não há uma versão oficial para processamento em GPU.

[2]Disponível no repositório do ExaML no Github: `https://github.com/stamatak/ExaML`.

utilizou-se a ferramenta INDELible [Fletcher and Yang 2009] para gerar um arquivo de alinhamento de DNA. O arquivo foi dividido em *taxa* distintos com a ajuda de um *script* desenvolvido em *Python* 3.

3. Desenvolvimento

Utilizando *Fixed-Position Logging Profiling* [Hegner 1999], comandos para medição de tempo foram distribuídos no inicio e no final das principais funções da aplicação. Após a análise das medições, conclui-se que a função **newview** representa cerca de 58% do tempo total de execução; sendo assim, essa função foi escolhida como escopo inicial do trabalho.

3.1. Implementação inicial

A função **newview** consiste em um *switch statement* com três caminhos lógicos (TIP_TIP, TIP_INNER e INNER_INNER), obteve-se então cinco *kernels*, dois dedicados a pré-computação dos dados para TIP_TIP e TIP_INNER e três para computação de cada caso. Originalmente os *kernels* foram implementada utilizando *buffers* na memória global da GPU, o que representa que, para a cada execução de uma função *kernel*, é necessário transferir matrizes entre o *host* e o *device*. Isto obviamente torna a aplicação mais lenta, uma vez que o desempenho da GPU fica condicionado a velocidade de transferências de dados realizada pelo barramento PCI-E [Cook 2012]; todavia, essa implementação inicial foi suficiente para validar, logo no início, os tempos de execução dos *kernels* (considerando a eliminação dessa transferência de memória).

3.2. Otimizações

A maneira usual de diminuir a movimentação de memória entre *device* e *host* é mantendo as variáveis que serão acessadas por determinado *kernel* na memória global do próprio *device* [Nvidia 2012]. Subentende-se que todos os cálculos que serão realizados sobre determinadas variáveis devem ser realizados na GPU, ocasionando a necessidade de implementação outras sub-rotinas da aplicação.

O transporte dos vetores para a memória da GPU foi possível em praticamente todos os casos, com exceção dos vetores recalculados a cada execução de **newview**. Dessa forma, amplia-se o número de *kernels* indispensáveis para tornar a utilização da arquitetura CUDA viável na aplicação, sendo agora necessária a reimplementarão das funções **core**, **evaluate** e **sum**, refletindo no desenvolvimento de seis novos *kernels*.

No desenvolvimento das funções, foram utilizados comandos de *loop unrolling* (permitindo a eliminação instruções de controle dos laços de repetição e suas penalidades naturais[3] [Davidson and Jinturkar 1996]) e redução paralela [Harris 2007] na soma de valores dos vetores.

3.3. Novo layout

Após o desenvolvimento inicial, um novo *layout* de *kernel* foi concebido a partir da ideia de redução paralela e da utilização da memória compartilhada: o objetivo do novo *layout* é

[3]As penalidades de desvios condicionais ocorrem em arquiteturas que utilizam *pileline*. Em uma *pipeline* com *prefetching* sequencial, o processador descarta instruções antecipadas quando deparado com um desvio condicional, esse descarte reduz consideravelmente o desempenho do processamento [Lalja 1988].

transferir a responsabilidade de indexação dos vetores para uma estrutura de *thread blocks* que facilite seu acesso e permita que múltiplas *threads* trabalhem de forma cooperativa utilizando a memória compartilhada. Nesse novo *layout* cada *thread* trabalha com uma pequena fração do cálculo, refletindo em um aumento considerável de utilização da placa gráfica.

4. Resultados

Em um primeiro momento, foram executados testes empregando o arquivo de entrada **49** (Seção 2), com o objetivo de comprovar a coerência das arvores filogenéticas geradas pelas novas implementações a partir de dados biológicos empíricos. Após validarmos os resultados, as aplicações foram submetidas para coleta dos tempos de execução.

Tabela 1. Tempos de execução (em segundos) para até 10.000 padrões.

	Número de padrões							
Versão da aplicação	**1.000**	**2.000**	**3.000**	**4.000**	**5.000**	**6.000**	**8.000**	**10.000**
SSE3	6,38	11,01	23,94	-	-	-	-	-
AVX	2,07	3,56	7,61	12,11	8,98	16,81	11,70	24,14
OMP-AVX	1,15	1,81	3,74	5,83	4,29	7,90	6,49	11,43
OLD-CUDA	5,15	5,38	8,98	11,77	8,25	14,64	9,27	17,76
NEW-CUDA	5,00	4,83	7,65	10,04	7,54	13,04	7,92	15,45

Em uma primeira análise da Tabela 1, os resultados parecem desanimadores, uma vez que os tempos de execução das aplicações utilizando CUDA só conseguem ser melhores que os tempos da versão SSE3; todavia, ao analisamos considerando os tempos de execução para conjuntos de dados maiores, é possível perceber duas tendências: (1) a redução da diferença entre a versão mais rápida (OMP-AVX) e as implementações em CUDA e (2) o distanciamento dos tempos de execução dos dois *layouts* desenvolvidos. Também é possível perceber que as implementações em CUDA, a partir de 4.000 *patterns*, já superam os tempos de execução da versão AVX.

Tabela 2. Tempos de execução (em segundos) para até 100.000 padrões.

	Número de padrões					
Versão da aplicação	**10.000**	**20.000**	**40.000**	**60.000**	**80.000**	**100.000**
SSE3	74,90	140,33	356,21	616,57	-	-
AVX	24,14	48,39	129,57	231,26	339,49	482,00
OMP-AVX	11,43	25,12	80,37	154,02	222,32	322,38
OLD-CUDA	17,76	34,87	79,30	130,92	179,77	250,17
NEW-CUDA	15,45	25,89	56,74	92,35	125,15	171,27

Quando submetemos conjuntos de dados maiores, entre 10.000 e 100.000 *patterns* (Tabela 2), as duas implementações desenvolvidas nesse trabalho ganham destaque: com 40.000 padrões, superam o tempo de execução da versão OMP-AVX. Com 100.000 padrões, CUDA-NEW obtém um ganho de desempenho de 181% sobre a versão AVX. Em nenhum dos testes NEW-CUDA obteve desempenho inferior à OLD-CUDA.

5. Conclusão

As implementações desenvolvidas em CUDA apresentam desempenho inferior quando submetidas a sequências de entrada menores, porém apresentam um ganho de desempenho considerável quando submetidas a um número maior de *patterns*, atingindo tempos

de processamento que podem chegar a duas ou três vezes o tempo de execução em CPU. Em análises usuais, o número de *patterns* utilizados por partição não costuma passar de 2.000, entretanto existem casos específicos onde são utilizadas partições com um número maior de padrões. É possível concluir que, no processamento com GPU, trabalhar com uma grande quantidade de dados não é somente desejável, mas necessário. Atualmente o processamento por inferência apresenta melhores resultados em processadores x86[4], entretanto há uma potencial obtenção de desempenho a partir do desenvolvimento de uma aplicação heterogênea, capaz de aproveitar o poder de processamento das GPUs atuais. Os códigos e utilizados neste artigo estão disponíveis no repositório do Github[5].

Referências

Charalambous, M., Trancoso, P., and Stamatakis, A. (2005). Initial experiences porting a bioinformatics application to a graphics processor. *Advances in Informatics*, pages 415–425.

Cook, S. (2012). *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes.

Davidson, J. W. and Jinturkar, S. (1996). Aggressive loop unrolling in a retargetable, optimizing compiler. In *International Conference on Compiler Construction*, pages 59–73. Springer.

Fletcher, W. and Yang, Z. (2009). Indelible: a flexible simulator of biological sequence evolution. *Molecular biology and evolution*, 26(8):1879–1888.

Harris, M. (2007). Optimizing cuda. *SC07: High Performance Computing With CUDA*.

Hegner, S. J. (1999). Analysis of algorithm fall 1999: Profiling of algorithms. Datavetenskap - Umeå universitet. Acesso em: 13 de outubro de 2017.

Izquierdo-Carrasco, F., Alachiotis, N., Berger, S., Flouri, T., Pissis, S. P., and Stamatakis, A. (2013). A generic vectorization scheme and a gpu kernel for the phylogenetic likelihood library. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 530–538. IEEE.

Kozlov, A. (2017). amkozlov/raxml-ng: Raxml-ng v0.2.0 beta.

Kozlov, A. M., Aberer, A. J., and Stamatakis, A. (2015). Examl version 3: a tool for phylogenomic analyses on supercomputers. *Bioinformatics*, page btv184.

Lalja, D. J. (1988). Reducing the branch penalty in pipelined processors. *Computer*, 21(7):47–55.

Nvidia (2012). Cuda programming guide.

Stamatakis, A. (2004). *Distributed and parallel algorithms and systems for inference of huge phylogenetic trees based on the maximum likelihood method.* PhD thesis, Technical University Munich.

Vandamme, A. (2009). *The phylogenetic handbook*. Cambridge, UK: Cambridge University Press.

[4]A aplicação RAxML *Next Generation* [Kozlov 2017] representa o estado da arte em relação a inferência por máxima verossimilhança em processadores x86.

[5]Repositório no Github: `https://github.com/marcelogm/CUDAExaML`

Paralelização de uma Aplicação de Detecção e Eliminação de Ruídos em Streaming de Vídeo com a DSL SPar

Renato B. Hoffmann, Dalvan Griebler, Luiz G. Fernandes

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Grupo de Modelagem de Aplicações Paralelas (GMAP), Porto Alegre – RS – Brasil

{renato.hoffmann,dalvan.griebler}@acad.pucrs.br

Resumo. *Restauração de imagem é uma importante etapa de qualquer sistema de computação gráfica. Este trabalho tem como objetivo apresentar e avaliar o paralelismo de Denoiser, uma aplicação para detecção e eliminação de ruído em streaming de vídeo. Foram avaliados o speed-up e programabilidade das interfaces SPar, Thread Building Blocks e FastFlow. Os resultados mostram que a SPar obteve bons resultados de programabilidade e desempenho.*

1. Introdução

Aplicações caracterizadas por um fluxo de dados vêm progressivamente crescendo em importância e difusão. De fato, aplicações de *streaming* como vídeo, áudio e imagem são recorrentes cargas de trabalho em computadores pessoais. Como são computacionalmente intensivas, essas aplicações tipicamente necessitam de paralelismo para obter resultados eficientemente. Nesse âmbito, a DSL (Linguagem Específica de Domínio) SPar [Griebler et al. 2017] foi proposta com o intuito de fornecer abstrações de paralelismo para o domínio de *stream*. Através de anotações no código fonte, a SPar propõe habilitar o paralelismo de *stream* sem a necessidade de refatorar o código fonte. Contudo, por ser uma interface emergente, ainda é necessário testar ela em aplicações reais.

Por outro lado, com a massiva produção de imagens e filmes digitais, muitas vezes gerados em más condições, filtros de restauração tornaram-se importantes etapas de sistemas de computação gráfica. Nesse contexto, destacamos Denoiser [Aldinucci et al. 2012], uma robusta aplicação para restauração de imagens corrompidas por ruído *Salt and Pepper*. Esse ruído se caracteriza por alterar o valor de variação do píxel para o máximo ou mínimo (0 ou 255 para uma imagem de 8 bits), gerando sobre a imagem um chuviscado preto e branco. Entretanto, por garantir alta qualidade de restauração, Denoiser possui uma vazão baixa, demandando paralelismo para fornecer resultados mais eficientemente.

Esse trabalho contribui com o desenvolvimento do paralelismo da aplicação Denoiser em duas novas interfaces: SPar e TBB [Reinders 2007]. Além disso, é fornecida uma análise comparativa de desempenho e programabilidade para as versões paralelas implementadas, incluindo uma própria versão com a interface FastFlow (seção 2.3 de [Aldinucci et al. 2014]). O artigo está estruturado da seguinte forma. Na Seção 2, são comparados e discutidos os trabalhos relacionados. A Seção 3 descreve a estrutura e estratégia de paralelismo da aplicação Denoiser. Subsequentemente, são apresentados e discutidos os resultados e experimentos na Seção 4. Por fim, a Seção 5 conclui o trabalho.

2. Trabalhos Relacionados

Os estudos de [Wan et al. 2010] abordam uma implementação paralela de um algoritmo de restauração de imagem em arquiteturas com memória distribuída. Em sua implementação, os autores utilizam um padrão mestre-escravo para processar diferentes segmentos de uma imagem em paralelo. Além disso, os autores fornecem uma avaliação do *speed-up* com até 6 processadores. Em comparação, nosso trabalho aborda o paralelismo de stream em arquiteturas com memória compartilhada e também apresenta uma avaliação de programabilidade. Outro fator de diferenciação é que nosso trabalho não prioriza a descrição das técnicas algorítmicas de restauração de imagem.

Em [Aldinucci et al. 2012], Denoiser é apresentado como uma aplicação para restauração de imagens corrompidas por *Salt and Pepper*. Os pesquisadores apresentam uma detalhada explicação algorítmica e o estado da arte das técnicas de restauração de imagem. Além disso, também é realizada uma avaliação do desempenho e breve descrição da implementação de paralelismo de dados e GPGPU. Mais tarde, uma nova versão do Denoiser foi apresentada pelos mesmos autores em [Aldinucci et al. 2014]. Nesse novo trabalho, o paralelismo da aplicação Denoiser é estendido para o domínio de *stream* com o suporte da biblioteca FastFlow. Entretanto, essa implementação foca em arquiteturas heterogêneas (multi-núcleo e GPGPU) enquanto que esse artigo trabalha com arquiteturas homogêneas (multi-núcleo). Em comparação com esse trabalho, [Aldinucci et al. 2012] e [Aldinucci et al. 2014] não comparam sua implementação com outras interfaces de paralelismo e não avaliam a programabilidade.

3. Denoiser

Denoiser é uma filtro capaz de restaurar imagens poluídas por esparsos píxeis pretos ou brancos (*salt and pepper*). O algoritmo é proposto em duas etapas: 1) identificação dos píxeis corrompidos e 2) restauração da imagem. Desta forma, Denoiser garante alta qualidade de restauração de imagem/vídeo com até 90% de ruído. Este estudo aborda paralelismo apenas em uma *stream* de vídeo. Na Figura 1, pode-se visualizar os resultados da execução e o fluxo de operação da aplicação. Inicialmente, a operação $Capture$ obtêm uma imagem de um vídeo. Se não possuir ruído, a imagem é então fornecida para o estágio $Noise$, que aleatoriamente aplicará o ruído *Salt and Pepper* nos píxeis da imagem. Na sequência, a imagem é enviada para o algoritmo $Detect$, onde os píxeis corrompidos são identificados e mapeados. Subsequentemente, os píxeis corrompidos da imagem são restaurados por $Denoise$ através de sucessivas iterações até a convergência. Finalmente, a imagem resultante é escrita em arquivo de saída na operação $Write$. Esse processo se repete para cada imagem do vídeo. O suporte para as operações de leitura e escrita em vídeo é fornecido pela biblioteca OpenCV.

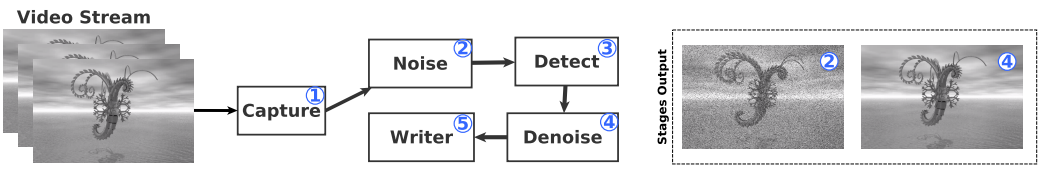


Figura 1. Fluxo de operação da aplicação Denoiser.

É possível adotar uma estratégia de paralelismo de dados, entretanto, desenvolvemos a versão paralela da aplicação Denoiser empregando elementos do paralelismo de *stream*. Dessa forma, apenas entradas de vídeo se beneficiam do paralelismo enquanto que imagens únicas são processadas sequencialmente. Essencialmente, a estrutura de paralelismo consiste em um *pipeline* de três estágios. Responsável por gerar os elementos da *stream*, o primeiro estágio é executado sequencialmente e encapsula as operações $Capture$, $Noise$ e $Detect$. Destas operações, apenas $Capture$ é estritamente sequencial. Entretanto, optamos por executar $Noise$ e $Detect$ no mesmo estágio. Isso porque, de acordo com nossos testes, essas duas operações representam uma parcela negligenciável da carga de trabalho total da aplicação. Mesmo sequencial, essas operações são capazes de gerar a vazão necessária para evitar um gargalo. Isso também foi constatado por [Aldinucci et al. 2014].

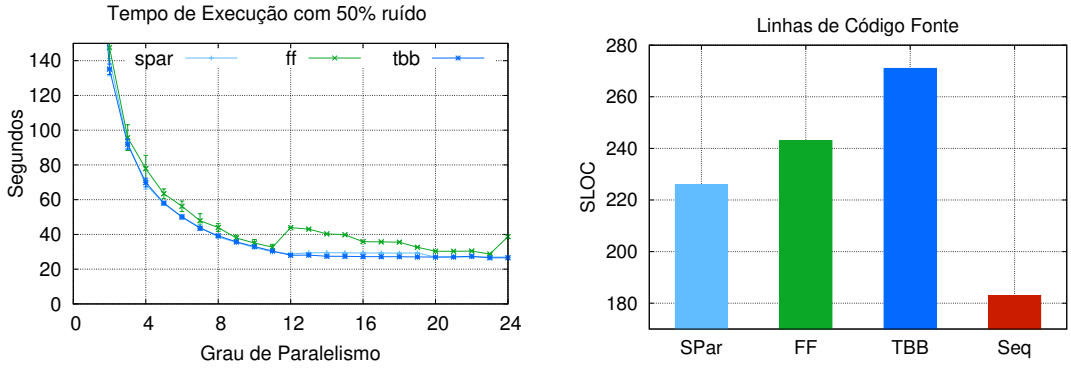
Envolto pelo segundo estágio do *pipeline*, o algoritmo $Denoise$ é a operação mais custosa da aplicação. Representando aproximadamente 95% do tempo de processamento de uma imagem, $Denoise$ é a operação onde o paralelismo é efetivado. Assim, diferentes imagens de um vídeo são processadas em paralelo de forma segura, visto que não possuem dependência de dados entre elas. Subsequentemente, o terceiro estágio do *pipeline* contém a operação $Write$, que é executada sequencialmente, uma vez que opera com apenas um canal de dados. Por padrão, em um programa paralelo, a ordem de processamento dos elementos da *stream* não é determinística. Consequentemente, foi necessário reordenar a *stream* no último estágio a fim de evitar a sobreposição no vídeo de saída.

4. Experimentos

Nesta seção, descrevemos os resultados obtidos e experimentos realizados. A máquina utilizada possui dois processadores *Intel(R) Xeon(R) CPU E5-2620 v3 2.40GHz* e 24GB de memória RAM. O sistema operacional foi Ubuntu Server 64 bits (*kernel 4.4.0-59-generic*). As ferramentas utilizadas foram: GCC (5.4.0), bibliotecas TBB (4.4 20151115), FastFlow (revisão 13) e OpenCV (2.4.9.1). Além disso, a compilação foi feita com a diretiva -O3. O vídeo de entrada é de formato AVI com tamanho de 7,4 MB (768x512 píxeis), que é carga padrão da aplicação base. Os valores foram gerados a partir da média aritmética de 10 execuções. O desvio padrão também é apresentado no gráfico. Para a avaliação de programabilidade, foi levado em consideração a métrica SLOC (número de linhas de código fonte). Todos os testes foram realizados para 3 versões: SPar (*spar*), FastFlow (*ff*) e TBB (*tbb*).

Apresentamos o Tempo de Execução de Denoiser (50% de ruído) na Figura 2(a). O tempo de execução sequencial é de 251 segundos. Aqui, observamos que a escalabilidade máxima do programa é atingida no grau de paralelismo 12 (número de *threads* físicas). Contudo, *ff* atinge esse ponto com apenas 11. Isso porque o FastFlow dedica uma *thread* exclusivamente para manejar o pipeline. Ainda, há um desbalanceamento de carga em *ff*, acentuado ao atingir o uso do recurso de *hyper-threading*. Esse desbalanceamento é solucionado em *spar* através da utilização de uma política de escalonamento sob-demanda em conjunto com um reordenamento livre de travas de sincronização. Já em *tbb*, o desbalanceamento de carga é lidado pelo *greedy-scheduling*, que impede que *threads* assumam um estado ocioso enquanto houver trabalho pendente.

O gráfico da Figura 2(b) apresenta o SLOC (Linhas de Código Fonte) das versões



(a) Tempo de Execução. (b) SLOC.

Figura 2. Experimentos Denoiser.

avaliadas. Dessa forma, esperamos obter um indicativo de programabilidade avaliando a intrusão de código necessária para expressar o paralelismo da aplicação Denoiser. *Seq* representa a versão sequencial do código. Como esperado, a SPar obteve o menor percentual de aumento. Isso porque seu sistema de anotações permite manter a estrutura do código fonte razoavelmente inalterada. Por outro lado, as bibliotecas FF e TBB possuem um aumento de SLOC maior visto que necessitam refatorar o código sequencial e adicionar novas estruturas de dados para comunicação entre os estágios.

5. Conclusões

Este artigo apresentou uma análise de desempenho, programabilidade e paralelização da aplicação de restauração de imagem Denoiser. Os resultados indicam que, das interfaces avaliadas, TBB apresenta o menor tempo de execução enquanto que a SPar fica 6% atrás no pior dos casos. Já na avaliação programabilidade, a SPar possui aproximadamente 24% menos intrusão de código que o TBB e 9% para o FastFlow. Futuramente, pretendemos melhorar a geração de código da SPar com os resultados observados, além de estender os estudos da aplicação Denoiser através de novos testes com diferentes cargas de trabalho.

Referências

[Aldinucci et al. 2014] Aldinucci, M., Peretti Pezzi, G., Drocco, M., Tordini, F., Kilpatrick, P., and Torquati, M. (2014). Parallel video denoising on heterogeneous platforms. In *2014 HLPGPU*, pages 1–8, Vienna, Austria. HiPEAC.

[Aldinucci et al. 2012] Aldinucci, M., Spampinato, C., Drocco, M., Torquati, M., and Palazzo, S. (2012). A parallel edge preserving algorithm for salt and pepper image denoising. In *2012 IPTA*, pages 97–102, Istambul, Turkey. IEEE.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):20.

[Reinders 2007] Reinders, J. (2007). *Intel Threading Building Blocks*. O'Reilly, Sebastopol, CA, USA.

[Wan et al. 2010] Wan, J., Ye, L., and Zhang, Q. (2010). Parallel multi-regions image restoration system and its implementation. In *2010 ICCSIT*, pages 216–220. IEEE.

Uso de OpenMP na Paralelização do algoritmo Artificial Bee Colony (ABC)

Natiele Lucca[1], **Claudio Shepke**[1]

[1] Ciência da Computação– Universidade Federal do Pampa (UNIPAMPA)
Campus Alegrete – RS – Brazil

{natiele,claudio} lucca@gmail.com, schepke@unipampa.edu.br

Resumo. *Este trabalho visa paralelizar o algoritmo Artificial Bee Colony (ABC) através do uso de diretivas OpenMP. Com isso será possível melhorar o desempenho da execução de um código-fonte da aplicação. Neste artigo, testes foram realizados comparando o paralelismo com limitante de threads. Os resultados demonstraram o impacto no uso de threads no tempo de execução e na qualidade da solução do algoritmo.*

1. Introdução

Artificial Bee Colony (Colônia Artificial de Abelhas) é um algoritmo bioinspirado baseado em populações e metaheurísticas que vêm sendo usado para resolver problemas de busca e otimização (SERAPIÃO, 2009). Este algoritmo é empregado em aplicações complexas, que possuem elevado custo de execução, como problemas de engenharia, economia e ciência.

A inteligência de colônias ou inteligência coletiva, é um conjunto de técnicas baseadas no comportamento coletivo de sistemas auto-organizados, distribuídos, autônomos, flexíveis e dinâmicos (SERAPIÃO, 2009). Algoritmos inspirados em formigas, bactérias, partículas e abelhas simulam o comportamentos do agente diante da colônia, em busca de soluções ótimas.

Este trabalho visa analisar o desempenho do algoritmo Artificial Bee Colony (ABC), explorando a eficiência no tempo de execução quando submetido a programação paralela de memória compartilhada.

2. Metodologia

Para a paralelização deste algoritmo foram utilizadas instruções OpenMP. A API (Application Programming Interface) e um conjunto de diretivas que permitem a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de threads (NETO; COSTA; SILVEIRA, 2010).

Inicialmente o programa possui uma thread. Quando seções paralelas são identificadas mais threads são disparadas de acordo com a especificação. A área paralelizada só é finalizada quando todas as threads completam a execução e são encerradas, em seguida a aplicação volta a possuir somente uma thread.

OpenMP é baseado em diretivas de compilação. Isso significa que são utilizadas palavras-chave (representadas por pragmas) que são interpretadas pelo compilador no momento da compilação. Essas palavras são instruções ao compilador e caso este não

tenha suporte a esse nível de paralelização, essas palavras serão ignoradas, fazendo com que o código seja serial (NETO; COSTA; SILVEIRA, 2010).

O algoritmo Artificial Bee Colony é composto por funções que representam as operações das abelhas campeiras, seguidoras e escudeiras. As abelhas campeiras são responsáveis por verificar a qualidade da resposta encontrada na equação em questão. As abelhas seguidoras são designadas quando as campeiras encontram boas fontes, ou seja, encontram melhores soluções para o problema. Já as escudeiras são abelhas campeiras que não obtiveram bons resultados e iniciam uma nova busca. O processo repete por um limite estabelecido.

Em sequência o pseudocódigo do algoritmo Artificial Bee Colony segundo SERAPIÃO:

1. Determine o tamanho da colônia de abelhas (COL), o número inicial de abelhas campeiras (BN); o número de fontes de alimento (SN), que é igual a BN; o número inicial de abelhas seguidoras (BC), que é igual à diferença entre COL e BN; o número de abelhas escudeiras (BE); e o número de tentativas de liberar uma fonte de alimento (lim).

2. Envie aleatoriamente as abelhas campeiras para as fontes de alimento iniciais.

3. Envie as abelhas seguidoras para as melhores fontes de alimento encontradas pelas campeiras e determine as quantidades de néctar coletadas por cada uma.

4. Calcule o valor de probabilidade das fontes que serão escolhidas pelas abelhas campeiras.

5. Interrompa o processo de exploração das fontes abandonadas pelas abelhas (piores fontes).

6. Envie as escudeiras, aleatoriamente, para a área de busca para descobrir novas fontes.

7. Memorize a melhor fonte de alimento encontrada até o momento.

8. Se o número de tentativas de descobrir novas fontes de alimento fracassar (nt ¿ lim), ou seja, se durante lim tentativas as fontes de alimento não melhorarem então as abelhas escudeiras devem abandonar suas fontes estagnadas e buscar aleatoriamente novas fontes de alimento.

9. Se condição de término não for alcançada, retorne ao passo 3.

Seguem os trechos de códigos que foram paralelizados:

1. O método que calcula as probabilidades de escolha de uma fonte de alimento (proporcional a qualidade da fonte) e a atualização da qualidade da mesma.

2. Função responsável por inicializar as fontes de alimento, onde acontecem os sorteios aleatórios das posições para os alimentos e as inicializações do vetor de fontes.

3. Método que compara as soluções encontradas pelas threads e em caso de um resultado melhor atualiza os valores.

4. Determinar para as fontes de baixa qualidade (soluções ruins) um novo inicio.

5. Múltiplas execuções concorrentes (runtime).

O algoritmo utilizado para a pesquisa é disponibilizado pelo Intelligent Systems Research Group (ABC, 2009), na linguagem de programação C e para a paralelização do mesmo são empregadas instruções OpenMP.

3. Resultados

Foram realizadas 30 execuções sequenciais do algoritmo ABC. Como ele possui variáveis determinadas por valores aleatórios há a necessidade de repetir o processo para que os dados não sejam equivocados. Em média o tempo gasto pela aplicação foi de 6,254s, com a solução em média de 0,36e-6 próximo ao ideal (zero).

O computador utilizado nos testes possui um processador Intel (R) Core (TM) i5-5200U, 4 cores, CPU de 2.20GHz e 4GB de memória.

Em seguida o algoritmo foi analisado e métodos paralelizáveis foram identificados e paralelizados através do pragma ?parallel for?, limitando o número máximo de threads em 2. Sendo registrado o tempo de execução e a qualidade da solução obtida para cada função exposto na Tabela 1.

Função	Tempo de Execução (s)	Solução
1 - CalculateProbabilities	6,22	$7.57e^{-08}$
2 - init	6,18	$4.80e^{-10}$
3 - SendEmployedBees	6,30	$3.98e^{-11}$
4 - SendScoutBees	6,20	$7.21e^{-9}$
5 - Main	6,21	$1.08e^{-11}$

Figure 1. Tabela 1. Tempo de execução para as funções paralelizadas com 2 threads

Como podemos observar o tempo de execução dos métodos foi semelhante ao sequencial e em apenas um caso superior, uma vez que o uso fixo de threads minimiza o tempo de espera entre os blocos de execução. Enquanto a solução apresentou melhora para todos os testes efetuados.

Um segundo teste foi realizado limitando o número de threads em 10 para cada uma das funções anteriormente citadas. Em busca de melhor desempenho como demonstrado na Tabela 2.

Como podemos observar na Tabela 2, o tempo de execução foi superior ao teste anterior (Tabela 1)e ao sequencial. Para os casos 2, 3 e 5 o algoritmo teve a qualidade da sua solução afetada.

As vezes instruções OpenMP não apresentam o resultado esperado, uma vez que o fluxo de execução de uma área paralelizada é encerrado apenas quando a sincronização das threads acontece.

Função	Tempo de Execução (s)	Solução
1 - CalculateProbabilities	9,51	$1.17e^{-8}$
2 - init	8,32	$1.04e^{+1}$
3 - SendEmployedBees	8,24	$1.43e^{+1}$
4 - SendScoutBees	9,4	$1.10e^{-12}$
5 - main	3,97	$4.52e^{+2}$

Figure 2. Tabela 2. Tempo de execução para as funções paralelizadas com limite de 10 threads

4. Considerações Finais

Os testes realizados e descritos nesse trabalho exploram a paralelização do algoritmo Artificial Bee Colony através de diretrizes OpenMp.

O algoritmo de otimização ABC apresentou melhora no tempo de execução e na solução do problema quando foram utilizadas 2 threads, mas quando submetido a 10 threads o tempo de execução e a solução tiveram um aumento.

Os resultados mostram a influência da quantidade de threads no desempenho de um algoritmo, apesar da análise que várias threads executam determinada tarefa em menos tempo, detalhes como capacidade de hardware e tempo de sincronização entre as threads, podem interferir na execução do algoritmo.

Em sequência, serão realizados outros testes com as funções que apresentaram bons tempos de execução e/ou qualidade de solução. Em busca de minimizar o tempo de execução do algoritmo Artificial Bee Colony, mantendo uma boa solução.

Referências

SERAPIÃO, A. B. S. Fundamentos de otimização por inteligência de enxames: uma visão geral. Revista Controle e Automação, v. 20, n. 3, p. 271-304, 2009.

NETO, H. P. B.; COSTA, J. A.; SILVEIRA, E. S. Processamento paralelo com openmp em um simulador dinâmico de linhas de ancoragem e risers, parte ii. vol, v. 29, p. 4, 2010.

ABC, Intelligent Systems Research Group, Department of Computer Engineering, Erciyes University, Turkiye, 2009.

Paralelização do Método Procura em Rede com OpenMP*

Pablo J. Pavan[1], Sandra B. Neuckamp[1], Edson L. Padoin[1,2], Philippe O. A. Navaux[2]

[1]Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

[2]Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil

{pablo.pavan, sandra.neuckamp, padoin}@unijui.edu.br, navaux@inf.ufrgs.br

Resumo. *A utilização eficiente dos recursos computacionais é um fator indispensável na busca por desempenho. Com essa premissa técnicas de paralelização são cada vez mais empregadas. Este trabalho apresenta os resultados da paralelização do método Procura em Rede com a API* OpenMP. *Testes foram realizados em dois ambientes e demostram uma redução de até* $25,56$ *vezes no tempo de execução.*

1. Introdução

As aplicações de simulação que são utilizadas na resolução de problemas reais necessitam de elevados tempos de processamento computacional. Assim, sistemas computacionais de alto desempenho tem sido desenvolvidos para suprir tal necessidade. O processador é um dos componentes que tem apresentado grande avanço tecnológico, passando a proporcionar a execução paralela de aplicações. Deste modo, aplicações podem ser divididas em partes e executadas em paralelo nos núcleos das unidade de processamento [Pilla e Meneses 2015]. Devido a necessidade de se implementar sistemas que utilizam de maneira eficiente a capacidade computacional das máquinas modernas, novas tecnologias estão sendo desenvolvidas. Dentre estas tecnologias, destaca-se a API OpenMP.

Em modelos matemáticos, a estimação de parâmetros desconhecidos, a partir de dados experimentais constitui a área de pesquisa chamada de Problema Inverso (PI). Para resolução destes problemas [da Silva Neto e Neto 2005] propuseram o método Procura em Rede, este que busca a partir da estimativa miníma dos parâmetros, encontrar os parâmetros que representam os dados experimentais com o melhor coeficiente de determinação (R^2) para aquela estimativa. Assim, neste trabalho foi aplicada a API OpenMP na paralelização do método Procura em Rede, almejando reduzir o tempo computacional deste método, e utilizar de forma eficiente toda capacidade computacional disponível em um sistema paralelo.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 descreve o estudo de caso. A metodologia é apresentada na Seção 4. Resultados são discutidos na Seção 5, seguidos das Conclusões e Trabalhos Futuros.

2. Trabalhos Relacionados

Diefenthäler utiliza o método Procura em Rede para estimar três parâmetros de uma função exponencial para o ajuste de curva referentes a dados coletados do resfria-

*Trabalho parcialmente apoiado por CNPq, CAPES, FAPERGS e FINEP. Pesquisa tem recebido recursos do programa EU H2020 e do MCTI/RNP-Brasil sob o projeto HPC4E de número 689772.

mento da água em diferentes ampolas de vidro [Diefenthäler et al. 2017]. Neste trabalho o método foi paralelizado utilizando OPENMP e apresentou ganhos de $90,82\ \%$ (utilizando 4 *cores*) na redução do tempo computacional se comparado ao método implementado em MATLAB.

Kang compara OPENMP, MPI e MapReduce aplicados sobre um benchmark [Kang et al. 2015]. Os resultados alcançados, indicam que quando o problema pode ser resolvido utilizando recursos de somente uma máquina o OPENMP é aconselhado já que não suporta memória distribuída. Para problemas que utilizam grande poder computacional, é aconselhado o uso do MPI já que este suporta memória distribuída, assim podendo ser executado em *clusters* ou *grids*. A utilização de MapReduce é aconselhada quando o número de dados utilizado seja grande, porém com poucas iterações.

3. Estudo de Caso

São diversas as situações do cotidiano em que observamos o resfriamento natural da água. Quando a água se encontra em um reservatório parado, ela tende a alcançar o equilíbrio térmico com o meio após um determinado tempo de exposição. Para este trabalho foram coletadas 9721 amostras de temperaturas de um ambiente de resfriamento natural da água. Neste estudo as temperaturas iniciaram em 79 graus Celsius e foram até 25 graus Celsius. As coletadas de temperatura foram feitas a cada 1 segundo utilizando-se de um equipamento de mensuração de temperatura ligado a um computador que armazenava os dados.

Nesse trabalho buscou-se aplicar o método Procura em Rede paralelo sobre dados experimentais do resfriamento da água com o objetivo de realizar um ajuste de curva que representa tais dados. Para este ajuste são necessários estimar 4 parâmetros (a, b, c, d) de uma função exponencial dada por: $f(x) = a \cdot exp(b \cdot x) + c \cdot exp(d \cdot x)$.

O método Procura em Rede consiste em definir para cada parâmetro a ser estimado um intervalo com um valor mínimo e máximo que possivelmente contenha um valor ótimo do parâmetro considerado, construindo assim uma rede de intervalos. De posse desta rede, o Problema Direto é resolvido com todas as combinações dos valores que compõem a rede fazendo a busca pelo menor erro, onde este é a menor diferença entre o valor obtido pelo cálculo do Problema Direto e o valor coletado experimentalmente [da Silva Neto e Neto 2005].

4. Metodologia

Foram utilizados dois ambientes nos testes realizados. O primeiro nomeado **Máquina 1** é composto por uma tradicional desktop, com processador Intel Core-I7 4790 de 8 *cores* com frequência base de $3,6$ GHz. Este equipamento possui 16 GB de memória RAM DDR3 de 1600 MHz. O segundo ambiente nomeado de **Máquina 2** possui dois processadores Intel Xeon E5-2640 v2 com 8 *cores* com 2 *SMT-cores*, totalizando 32 *cores* de $2,00$ GHz. Este equipamento possui 64 GB de memória RAM DDR3 de 1600 MHz. Em ambas as máquinas utilizaram-se o sistema operacional Ubuntu com versão de *kernel* $3.16.0-70$. A versão do OPENMP utilizada foi a 4.0 e do compilador gcc a $5.4.0$.

Na implementação, procurou-se utilizar a diretiva de paralelização para laços de repetições, como neste problema existem 4 laços de repetições aninhados no cálculo inverso, a diretiva foi colocada no laço mais externo. Utilizando-se da opção $collapse$ para

reduzir a granularidade do trabalho para cada segmento. Outros cálculos do problema foram paralelizados utilizando o $parallel for$ e $reduction$, buscando aumentar as regiões paralelas.

De modo a utilizar todos os recursos computacionais, o problema foi executado na Máquina 1 com 1, 2, 4 e 8 *cores* e na Máquina 2 com 1, 8, 16 e 32 *cores*. Em ambas as configurações foi utilizado a *flag* de otimização -O2 do compilador gcc. Cada teste foi executado 10 vezes utilizando-se uma média aritmética para os resultados.

5. Resultados

A Figura 1 demostra o ajuste de curva, os pontos em azul são os dados experimentais da temperatura da água e a linha em preto é a curva ajustada. Para este ajuste o valor de R^2 foi de $0,9963$.

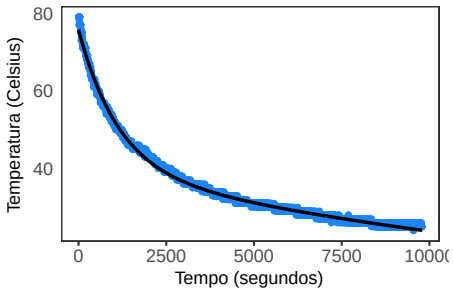
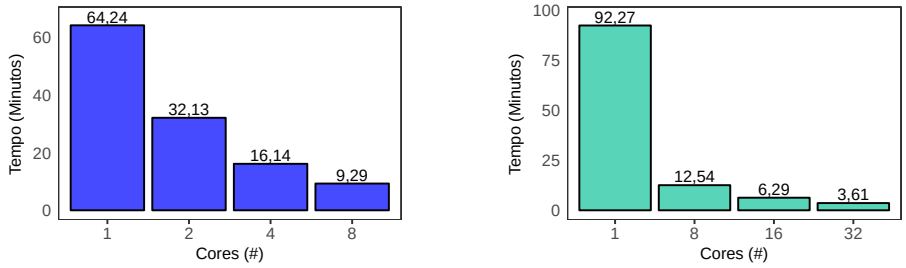


Figura 1. Ajuste da Curva em Relação as Dados Experimentais.

Na Figura 2 são apresentados os tempos de execução mensurados com a execução da versão paralela na Máquina 1 e na Máquina 2 variando a quantidade de *cores*.

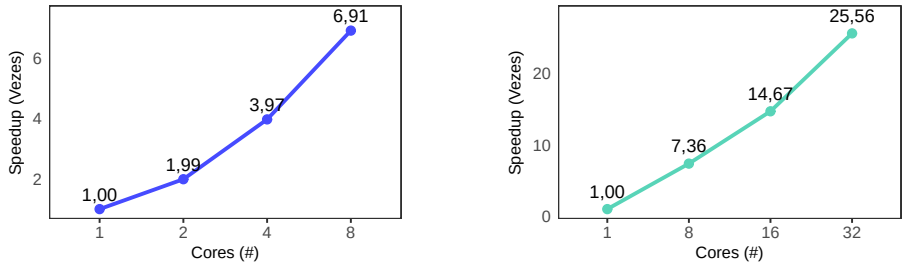


(a) Máquina 1. (b) Máquina 2.

Figura 2. Tempo de Execução do Método Procura em Rede Implementado em OpenMP.

O tempo de execução da versão sequencial foi de $64,24$ minutos na Máquina 1 e de $92,27$ minutos na Máquina 2. O tempo é maior na Máquina 2 pois seu clock base é menor que na Máquina 1. Este tempo foi reduzido para $9,29$ minutos na Máquina 1 e para $12,54$ minutos na Máquina 2 quando utilizado oito cores.

Com o uso da versão paralela, em ambas as máquinas obteve-se ganhos de desempenho (Figura 3). Para 8 *cores* na Máquina 1 o ganho foi de $6,91$ vezes enquanto que na Máquina 2 foi de $7,46$ vezes. O melhor *speedup* ganho foi de $25,56$ vezes na



(a) Máquina 1. (b) Máquina 2.

Figura 3. Speedup da Implementação com OpenMP em diferentes *cores*.

Máquina 2 utilizando 32 *cores*. Isso representa em tempo de execução uma redução de $99,27$ minutos para $3,61$ minutos.

Com o uso da versão paralela, o emprego do método Procura em Rede passa ser viável em pesquisas com dados reais. Devido a grande demanda de processamento ele não tem sido utilizado mesmo apresentando resultados eficientes. Neste nosso estudo, a função exponencial e os parâmetros ajustados pelo método Procura em Rede, foi encontrado uma curva que representa os dados experimentais do resfriamento da água com uma acurácia de $99,63\%$.

6. Conclusões e Trabalhos Futuros

O método Procura em Rede pode ser utilizado para encontrar parâmetros de diversas funções, neste trabalho o mesmo foi utilizado para encontrar os parâmetros de uma função exponencial para o ajuste de curva de dados experimentais do resfriamento da água. Para este problema o método tornou-se eficaz apresentando uma acurácia de $99,63\%$. O uso de paralelismo para o método Procura em Rede, apresentou uma redução no tempo de execução de $25,56$ vezes, reduzindo o tempo de $92,27$ minutos para $3,61$ minutos.

Em trabalhos futuros, pretende-se executar esta aplicação em ambientes com maior disponibilidade de *cores*, como também realizar uma otimização nos acessos a memória. Pretende-se também paralelizar este método para outros problemas, buscando a redução no tempo de execução.

Referências

da Silva Neto, A. J. e Neto, F. D. M. (2005). *Problemas inversos: conceitos fundamentais e aplicações*. EdUERJ.

Diefenthäler, A. T., Avi, P. C., e Padoin, E. L. (2017). Processamento paralelo na determinação da curva de resfriamento da Água pelo método de procura em rede. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, 6(1):1–2.

Kang, S. J., Lee, S. Y., e Lee, K. M. (2015). Performance comparison of openmp, mpi, and mapreduce in practical problems. *Advances in Multimedia*, 2015:7.

Pilla, L. L. e Meneses, E. (2015). Programação paralela em charm++. *ERAD/RS, Gramado, RS, Brasil*.

Projeto HELIX: Concepção de uma SIoT Explorando o Serviço FCM da Google

Rociele Prietsch[2*]; Leonardo João[1] ; Patrick Fernandes[2]; Felipe Haertel[2]
Cleiton Garcia[2]; João Lopes[3]; Adenauer Yamin[1 2]

[1]LUPS – Laboratory of Ubiquitous and Parallel Systems - UFPEL
[2]G3PD - Grupo de Pesquisa em Processamento Paralelo e Distribuído - UCPEL
[3]Instituto Federal Sul-rio-grandense - IFSUL

{silveirarociele,ldrsjoao,patrickjbf,felipe.haertel}@gmail.com
{cleitongarcia,joao.lblopes,adenauer}@gmail.com

Resumo. *O trabalho de pesquisa tratado neste artigo tem como objetivo a concepção de uma SIoT para o Projeto HELIX. Este projeto tem por premissa contribuir para a acessibilidade de pessoas com deficiência visual. A SIoT em desenvolvimento emprega o serviço Firebase Cloud Messaging do Google, e os testes realizados até o momento com as tecnologias envolvidas se mostraram promissores.*

1. Introdução

Segundo dados coletados no censo realizado pelo IBGE (Instituto Brasileiro de Geografia e Estatística)[1], existem no Brasil 45,6 milhões de Pessoas com Deficiência, ou seja, 23,9% do total da população. Mesmo com o auxílio de facilitadores como óculos ou lentes de contato, 35,7 milhões de pessoas afirmam ter dificuldades para enxergar, isto é, 18,60% dos entrevistados de acordo com o censo.

Tendo em conta este cenário, vem sendo concebido o projeto HELIX [Garcia 2017] apresentado neste artigo, o qual tem por premissa atender a acessibilidade de PCDVs (Pessoas com Deficiência Visual) total ou parcial, contribuindo com a inclusão social destas pessoas, promovendo uma maior independência e aumento da qualidade de vida das mesmas. Para tanto, o HELIX integra recursos de hardware, firmware e software, explorando ciência de contexto para constituir uma Rede Social na Internet das Coisas (*Social Internet of Things* - SIoT). A SIoT do HELIX emprega o Subsistema de Adaptação e Reconhecimento de Contexto do middleware EXEHDA [Lopes et al. 2014] O EXEHDA é um middleware que, dentre outras funcionalidades, provê suporte para a aquisição, armazenamento e processamento das informações de contexto necessárias às diferentes funcionalidades providas pelo HELIX.

O objetivo deste artigo é caracterizar como a SIoT do Projeto HELIX está sendo concebida, bem como o serviço Firebase Cloud Messaging (FCM) do Google é empregado no seu desenvolvimento.

---

*Bolsista PIBIC/CNPq

[1]http://www.pessoacomdeficiencia.gov.br

2. Projeto HELIX: Visão Geral e Funcionalidades

A abordagem HELIX considera as premissas da área de tecnologia assistiva, sendo sua questão central atender às necessidades dos usuários tendo como pressupostos potencializar a autonomia e minimizar esforços de configuração, promovendo uma operação o mais transparente possível para as PCDVs. Para tanto, a abordagem HELIX explora a sinergia da ciência de contexto com recursos da computação móvel, disponibilizando uma infraestrutura de SIoT, na qual os objetos inteligentes da IoT podem interoperar.

Considerando as premissas que uma SIoT deve contemplar, bem como aquelas previstas para o HELIX, foram concebidas as funcionalidades a serem contempladas na sua arquitetura, as quais estão sumarizadas a seguir.

Visão Geral das Funcionalidades

Uma visão geral dos participantes da SIoT do HELIX está disponível na Figura 1 e seus atores tem as seguintes características [Garcia 2017]:

- Cuidador Pessoal: indivíduo responsável por prover auxílio à PCDVs, pode ser um indivíduo familiar ou até mesmo um profissional contratado que fique disponível para atender as diferentes notificações do HELIX referente às PCDVs. Uma mesma PCDV poderá ter diferentes cuidadores pessoais cadastrados os quais serão acionados em sequência cuja ordem deverá ser previamente definida.
- Cuidador Corporativo: indivíduo responsável pela zeladoria, auxílio e segurança de pessoas em ambientes corporativos. O qual também irá atender às notificações do HELIX referentes às PCDVs na sua área de cobertura empresarial.
- PCDV Indoor: pessoa com deficiência visual que está ativa em ambientes internos, a qual irá realizar a leitura de QR-Codes enquanto mecanismo de localização. Alguns QR-Codes localizados em posições estratégicas, de conteúdo fixo ou dinâmico (provido por Totens), também poderão ser empregados para disponibilização de informações especializadas. Os QR-Codes são disponibilizados no formato de coluna, para compensar diferenças de altura, e ficam associados às portas existentes no ambiente, as quais são localizadas através de um piso tátil.
- PCDV Outdoor: pessoa com deficiência visual atuando em ambientes externos, para a qual o HELIX irá em períodos de tempo previamente cadastrados informar automaticamente a localização da PCDV.
- Servidor da SIoT: servidor capaz de executar as atividades de reconhecimento de contexto e prover suporte à operação da SIoT, através do envio de notificações entre os atores envolvidos.

As PCDVs e seus cuidadores utilizam um Smartphone com recurso de GPS no qual os aplicativos móveis do HELIX devem ser instalados.

3. Explorando o Serviço FCM na SIoT do Projeto HELIX

Na SIoT o relacionamento social entre os objetos deve acontecer com a menor a intervenção humana possível. Deste modo, em uma SIoT, um objeto com funcionalidades de servidor pode ser responsável por inferir situações em outros objetos, e articular relações, empregando mecanismos de ciência de contexto [Leal et al. 2013].

Uma SIoT dentre as suas premissas considera que devem ser mantidos separados os níveis de "pessoas"e "coisas", permitindo que objetos tenham suas próprias interações

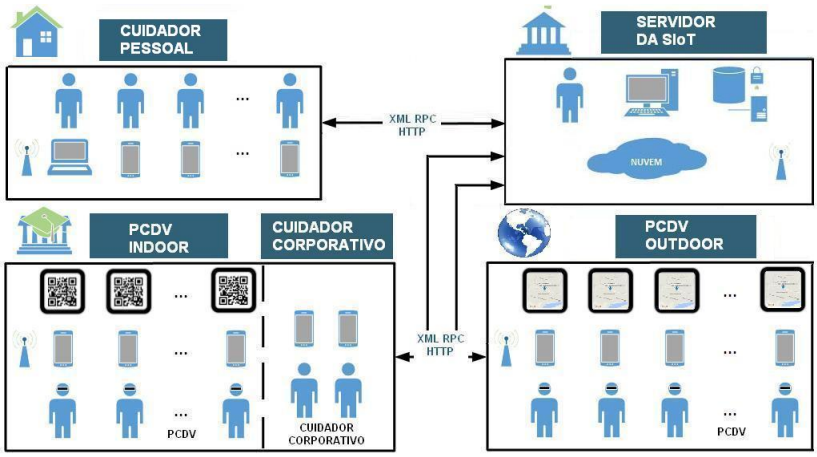


Figura 1. Projeto HELIX: Visão geral das funcionalidades. FONTE: [Garcia 2017]

e os seres humanos especifiquem regras tanto para proteger sua privacidade, como para acessar os resultados das interações autônomas entre os objetos [Atzori et al. 2012].

O serviço Firebase Cloud Messaging (FCM), anteriormente chamado de Google Cloud Messaging, é uma solução multiplataforma para troca de notificações entre dispositivos Android, iOS e servidores em geral, esse serviço é *open source*, o que contribui para sua apropriação em projetos. O FCM oferece uma ampla gama de opções e recursos de notificação, envolvendo diferentes tipos e volumes de dados.

Para fazer uso do serviço é necessário que a aplicação envolvida faça um registro no FCM. Por meio desse registro e criado um identificador único do dispositivo utilizado (*Device ID*). Este ID deve ser empregado quando do envio de mensagem ao servidor do FCM, caracterizando assim o produtor e o destinatário da notificação.[Google 2017]

O serviço FCM envia uma mensagem de *Intent Broadcast* e aguarda que o dispositivo para o qual a mensagem foi enviada processe a mesma. Importante destacar que a aplicação não precisa estar executando neste momento, o framework do FCM disponível no dispositivo destinatário é responsável por "acordar" a aplicação que deve tratar a mensagem, a qual então realiza o processamento pertinente.

4. Estudo de Caso e Tecnologias

Para avaliação da SIoT que está sendo concebida para o Projeto HELIX, está em fase de prototipação um estudo de caso direcionado ao acompanhamento do estado da bateria dos smartphones utilizados pelos PCDVs. Esta solução tem a seguinte organização: (i) uma aplicação Android que é executada no smartphone do PCDV que faz a coleta do estado da bateria; (ii) as regras da SIoT que executam no Servidor de Contexto do EXEHDA, para seleção do cuidador que será enviada a mensagem, consideram a localização do cuidador e o horário que será feito o envio, e (iii) uma aplicação instalada no smartphone do cuidador, que informa sua localização ao servidor da SIoT de tempos em tempos, o que também caracteriza a disponibilidade do mesmo para receber notificações (está ativo).

As notificações enviadas para os cuidadores são disparadas pela SIoT que executa no Servidor de Contexto do EXEHDA empregado pelo HELIX. Para envio das notificações é então utilizado o serviço FCM da Google.

Tanto a coleta do status da bateria no smartphone do PCDV, como confirmações de atendimento por parte dos cuidadores, utilizam o padrão arquitetural REST, sendo caracterizados os smartphones como Servidores de Borda do EXEHDA.

Na atual fase do HELIX, a versão de avaliação da SIoT proposta foi prototipada empregando as seguintes tecnologias:

- Ferramenta Android Studio para desenvolvimento do software dos Assistentes Móveis;
- Smartphones Android, nos quais foram instalados os Assistentes Móveis;
- Framework Django REST para realizar a interoperação dos componentes da arquitetura do HELIX;
- Linguagem Python para programação das funcionalidades da SIoT, junto ao Servidor de Contexto do EXEHDA.

5. Considerações Finais

O esforço de estudo e pesquisa está em andamento, tendo sido realizadas atualizações nas diferentes ferramentas tecnológicas empregadas no Projeto HELIX. Os testes envolvendo estas ferramentas, em particular o serviço FCM do Google tem apresentado resultados promissores.

Uma vez concluída a etapa de prototipação e testes dos diferentes softwares envolvidos, pelos integrantes do Grupo de Pesquisa, será feita uma avaliação envolvendo a comunidade usuária empregando o método TAM (*Technology Acceptance Model*). Neste momento, o projeto será submetido ao um Comitê de Ética em Pesquisa (CEP), uma vez que envolve pessoas.

Como trabalho futuro, destaca-se portar as aplicações desenvolvidas para o Android, para a plataforma iOS da Apple.

Referências

Atzori, L., Iera, A., Morabito, G., and Nitti, M. (2012). The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization. *Computer Networks*, 56(16):3594–3608.

Garcia, C. (2017). Uma Arquitetura para Contribuir com a Acessibilidade de PCDVs Explorando a Internet das Coisas. Dissertação de mestrado, Universidade Católica de Pelotas.

Google (2017). Firebase Cloud Messaging Google Service. Disponível em: $< https : //firebase.google.com/docs/cloud - messaging/ >$. Acesso em Janeiro de 2018.

Leal, A. G., Santos, A. S. d., Noda, M. K., and Rodrigues, L. C. d. S. (2013). Internet social das coisas como agente agregador nas cidades inteligentes no brasil e no mundo. *III Congresso Internacional do Conhecimento e Inovação, CIKI, 2013, Porto Alegre.*, pages 974–986.

Lopes, J., Souza, R., Geyer, C., Costa, C., Barbosa, J., Pernas, A., and Yamin, A. (2014). A middleware architecture for dynamic adaptation in ubiquitous computing. *Journal of Universal Computer Science*, 20(9):1327–1351.

Ranqueamento de Recursos na Internet das Coisas Explorando Algoritmos MCDA

**Juan Burtet, Huberto K. Filho, Renato Dilli,
Felipe C. Gruendemann, Ana Marilza Pernas, Adenauer Yamin**

[1]Laboratory of Ubiquitous and Parallel Systems – UFPEL
Pelotas, RS - Brasil

{jburtet, hkaiser, renato.dilli, fcgruendemann, marilza, adenauer}
@inf.ufpel.edu.br

Resumo. *A IoT caracteriza-se por possuir uma grande quantidade de dispositivos conectados em rede. O procedimento de descoberta e seleção de recursos neste cenário representa um desafio de pesquisa em aberto. Com o emprego do gerador e classificador de recursos concebidos, foi possível avaliar o desempenho do processo de ranqueamento, considerando uma grande quantidade de recursos. Os resultados com o algoritmo MCDA proposto são promissores.*

1. Introdução

A Internet das Coisas (IoT - *Internet of Things*), caracteriza-se pela presença de uma grande quantidade de objetos que pelo seu perfil operacional realizam conexões transientes na Internet, totalizando uma grande quantidade de recursos disponibilizados [Perera 2017]. Atualmente mais de seis bilhões de coisas estão conectadas à Internet disponibilizando serviços aos clientes e há previsão de 100 bilhões até 2025 [Gartner 2015] [BIS Research 2017].

Os recursos podem ser constituídos por *hardwares* heterogêneos, tais como, sensores e atuadores. O acesso aos dispositivos geralmente é realizado através de um ou mais serviços, caracterizando recursos de diferentes naturezas. Um desafio a ser vencido após a identificação e localização destes recursos é ranquear os serviços oferecidos para selecionar o que melhor atende a solicitação do usuário. Os processos de ranqueamento se concentram nas preferências do usuário, que frequentemente estabelecem uma ordem baseada na Qualidade de Serviço (QoS).

A Análise de Decisão de Múltiplos Critérios (MCDA - *Multiple Criteria Decision Analysis*) refere-se à tomada de decisões na presença de critérios múltiplos, geralmente conflitantes. Os algoritmos MCDA visam auxiliar no julgamento da tomada de decisão utilizando um conjunto de objetivos e critérios, estimando seus pesos de importância relativa e estabelecendo a contribuição de cada opção em relação a cada critério de desempenho. MCDA não é apenas um conjunto de teorias, metodologias e técnicas, mas também uma perspectiva específica para lidar com problemas de tomada de decisão [Figueira et al. 2005].

Este trabalho apresenta abordagens para geração e ranqueamento de recursos no contexto da IoT, com objetivo de avaliar o desempenho do algoritmo MCDA proposto. O restante do trabalho está organizado da seguinte forma: na Seção 2 é apresentado o método de ranqueamento de recursos utilizado, na Seção 3 é apresentado tanto um gerador,

como um classificador a ser utilizado para o ranqueamento de recursos, e na Seção 4 são feitas as considerações finais.

2. Modelo para Ranqueamento de Recursos Proposto

O processo de descoberta de recursos engloba a seleção dos recursos mais adequados à requisição do cliente. Nesta seção é apresentado o algoritmo MCDA proposto para o processo de ranqueamento de recursos utilizado neste trabalho.

O algoritmo MCDA proposto é uma adaptação dos algoritmos *The Simple Additive Weighting* (SAW) [Tzeng and Huang 2011] e *Web Service Relevancy Function* (WsRF) [Al-Masri and Mahmoud 2007], com a primeira etapa de normalização de dados proposta por [Liu et al. 2004].

Para a normalização da matriz são definidos dois vetores. O primeiro N = $\{n_1,n_2,...,n_m\}$. O valor de n_j pode ser 0 ou 1. Será 1 quando o aumento de $q_{i,j}$ beneficia a requisição do cliente e 0 quando o aumento de $q_{i,j}$ não beneficia a requisição do cliente. O segundo vetor C = $\{c_1, c_2,...,c_m\}$ contém constantes com o máximo valor normalizado para cada atributo.

As seguintes etapas devem ser realizadas para o cálculo da avaliação dos recursos através do algoritmo MCDA:

1. Normalizar a matriz $Q = (q_{ij})_{n\times m}$ de acordo com a Eq.(1) se o critério deve ser maximizado ou Eq.(2) se o critério deve ser minimizado. Nestas equações, $\frac{1}{n}\sum_{i=1}^{n} q_{i,j}$ é a média dos atributos de qualidade j na matriz Q [Liu et al. 2004].

$$v_{i,j} = \begin{cases} \frac{q_{i,j}}{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}} & if \quad \frac{1}{n}\sum_{i=1}^{n} q_{i,j} \neq 0 \\ & and \quad \frac{q_{i,j}}{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}} < c_j \\ & and \quad n_j = 1 \\ c_j & if \quad \frac{1}{n}\sum_{i=1}^{n} q_{i,j} = 0 \\ & and \quad n_j = 1 \\ & or \quad \frac{q_{i,j}}{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}} \geq c_j \end{cases} \quad (1)$$

$$v_{i,j} = \begin{cases} \frac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}} & if \quad q_{i,j} \neq 0 \\ & and \quad n_j = 0 \\ & and \quad \frac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}} < c_j \\ c_j & if \quad q_{i,j} = 0 \\ & and \quad n_j = 0 \\ & or \quad \frac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}} \geq c_j \end{cases} \quad (2)$$

2. Calcular o vetor pontuação de cada opção disponível. Cada pontuação pode ser calculada usando a Eq.(3)a e o operador $\max(v_j)$ representando o maior valor de atributo normalizado na coluna j. Portanto, precisamos definir uma matriz que represente a contribuição de pesos para cada recurso, onde $w = \{w1, w2, w3, ..., wj\}$. Cada peso nesta matriz representa o grau de importância ou fator de peso associado a uma propriedade QoS específica. Os valores de desses pesos variam de 0 a 1. A Eq.(3)b soma todos os atributos de qualidade para o recurso R_i, onde N representa o número de atributos.

$$h_{i,j} = w_j \left[\frac{v_{i,j}}{max(v_j)}\right]; \quad R_i = \sum_{j=1}^{N} h_{i,j} \quad \text{e} \quad MCD(A_i) = \left[\frac{100 * R_i}{max(R)}\right]. \quad (3)$$

3. Prototipação e Testes

Com objetivo de analisar o desempenho do modelo de ranqueamento proposto, foi concebido um gerador de recursos na linguagem Python, adotada pelo grupo de pesquisa, que produz utilizando a biblioteca *random*, um conjunto de recursos aleatórios. Para cada atributo é selecionado um valor aleatório entre os valores mínimo e máximo definidos pelo seus atributos. Registre-se que estes valores mínimo e máximo tem por base o dataset [Al-Masri and Mahmoud 2007], que traduz um cenário real. Ao fim da criação do conjunto, é feito o ranqueamento de todos os recursos gerados.

As Tabelas 1, 2 e 3 apresentam o processo de ranqueamento de dez recursos produzidos pelo gerador, e após processados pelo classificador de recursos proposto.

Tabela 1. Exemplo de Dataset

RT	AV	TH	RE	LA
302,75	89	7,1	73	187,75
482,00	85	16	73	1
3321,4	89	1,4	73	2,6
126,17	98	12	67	22,77
107,00	87	1,9	73	58,33
107,57	80	1,7	67	18,21
255,00	98	1,3	67	40,8
136,71	76	2,8	60	11,57
102,62	91	15,3	67	0,93
200,00	40	13,5	67	41,66

Tabela 2. Atributos Normalizados

RT	AV	TH	RE	LA
1,70	1,07	0,97	1,06	0,21
1,07	1,02	2,19	1,06	38,56
0,15	1,07	0,19	1,06	14,83
4,07	1,18	1,64	0,98	1,69
4,80	1,04	0,26	1,06	0,66
4,78	0,96	0,23	0,98	2,12
2,02	1,18	0,18	0,98	0,95
3,76	0,91	0,38	0,87	3,33
5,00	1,09	2,10	0,98	41,46
2,57	0,48	1,85	0,98	0,93

Tabela 3. Atributos Classificados

RT	AV	TH	RE	LA	R
0,24	0,91	0,33	0,94	0,00	61
0,15	0,87	0,75	0,94	0,70	86
0,02	0,91	0,07	0,94	0,27	56
0,57	1,00	0,56	0,86	0,03	76
0,67	0,89	0,09	0,94	0,01	66
0,67	0,82	0,08	0,86	0,04	62
0,28	1,00	0,06	0,86	0,02	56
0,53	0,78	0,13	0,77	0,06	57
0,70	0,93	0,72	0,86	0,75	100
0,36	0,41	0,63	0,86	0,02	58

A primeira etapa da classificação é a normalização dos dados. Para tanto, consideramos os vetores N={0,1,1,1,0} e C={5,2,3,2,50}. Os atributos *Response Time* e *Latency* qualificam o recurso com valores baixos obtidos pela Eq.(1) e *Availability*, *Throughput* e *Reability* qualificam com valores altos pela Eq.(2).

A Tabela 3 apresenta os valores dos atributos após aplicar a Eq.(3)a, ou seja dividir o valor normalizado da Tabela 2 pelo maior valor normalizado de cada coluna, multiplicado pelos pesos definidos pelo especialista w={0.70,1.00,0.75,0.94,0.75}. O valor MCDA é obtido aplicando a Eq.(3)b, que irá somar todos os valores de atributos em cada linha. Após é aplicada a Eq.(3)c que irá qualificar o recurso com um valor que vai de 0 a 100. Será dado o valor 100 para o melhor recurso do *dataset*. O resultado da classificação está informado na coluna "R". A classificação final é apresentada ao cliente em ordem decrescente, ou seja, do recurso mais qualificado ao menos qualificado.

Com o emprego do *software* desenvolvido para geração de recursos, foram criados 4 conjuntos com seus atributos de qualidade. Para o processo de criação destes recursos, optou-se por utilizar conjuntos com 3, 5 e 7 atributos com o objetivo de apontar a diferença de tempo no cálculo MCDA considerando diferentes quantidades de atributos. Além disso, foram criados *datasets* de tamanhos diferentes para analisar o aumento de tempo para conjuntos com diferentes quantidades de recursos e com mesmo números de atributos. O tamanho dos *datasets* testados foram de [10.000; 100.000; 500.000;

1.000.000] recursos. O desempenho em cada um dos testes é visto na Tabela 4 (Tempo de criação de *datasets*) e na Tabela 5 (Tempo de classificação MCDA)

Tabela 4. Tempo de geração dos recursos e seus atributos

	Recursos (x 1.000)			
Atrib	10	100	500	1.000
3	0.86s	8.69s	45.87s	88.17s
5	1.30s	13.92s	65.38s	130.17s
7	1.77s	17.40s	87.37s	174.49s

Tabela 5. Tempo de ranqueamento dos recursos

	Recursos (x 1.000)			
Atrib	10	100	500	1.000
3	0.10s	1.01s	5.15s	10.08s
5	0.15s	1.44s	7.20s	14.50s
7	0.18s	1.88s	9.59s	18.79s

Todos os testes foram feitos em uma máquina com Intel I5-3570K CPU @ 3.40 GHz e 4.00 GB de RAM. Com uma análise dos resultados, foi notado um ótimo desempenho no algoritmo SAW para classificação de recursos. Também foi notado que houve um aumento linear no tempo de geração e ranqueamento, conforme o aumento do número de recursos.

4. Considerações Finais

A utilização do algoritmo MCDA proposto para a classificação de recursos no contexto da Internet das Coisas apresentou um ótimo desempenho, considerando a literatura na área [Al-Masri and Mahmoud 2007], mesmo com um grande aumento no conjunto de dados e no número de atributos de qualidade.

Em trabalhos futuros, novos testes serão feitos, comparando a acurácia do ranqueamento obtido pelo algoritmo MCDA com novos métodos de ranqueamento, tais como Lógica *Fuzzy* e Aprendizagem de Máquina.

Referências

Al-Masri, E. and Mahmoud, Q. H. (2007). QoS-based discovery and ranking of Web services. In *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, pages 529–534.

BIS Research (2017). Global Sensors in Internet of Things (IoT) Devices Market, Analysis & Forecast: 2016 to 2022. Technical report.

Figueira, J., Greco, S., and Ehrgott, M. (2005). Multiple criteria decision analysis: state of the art surveys. *Multiple Criteria Decision Analysis State of the Art Surveys*, 78.

Gartner (2015). Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015.

Liu, Y., Ngu, A. H., and Zeng, L. Z. (2004). QoS computation and policing in dynamic web service selection. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73.

Perera, C. (2017). *Sensing as a Service for Internet of Things: A Roadmap*. Leanpub Publishers.

Tzeng, G. H. and Huang, J. J. (2011). *Multiple Attribute Decision Making: Methods and Applications*. A Chapman & Hall book. Taylor & Francis.

SMARTLB: Proposta de um balanceador de carga para redução de tempo de execução de aplicações em sistemas paralelos *

Vinicius R. S. dos Santos[1], **Edson L. Padoin**[1,2]

[1]Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

[2]Universidade Federal do Rio Grande do Sul (UFRGS) - Porto Alegre - RS - Brasil

{vinicius.ribas,padoin}@unijui.edu.br

Resumo. *Este artigo apresenta a proposta de um novo balanceador de carga que almeja a redução do tempo de execução de aplicações paralelas quando executadas em ambientes de memória compartilhada. O algoritmo proposto foi implementado no modelo de programação paralela* CHARM++. *Os resultados mensurados apresentaram speedup de* $1,22$ *à* $1,65$ *em relação ao tempo de execução com diferenças significativas no nível de desbalanceamento final na execução das aplicações.*

1. Introdução

O desenvolvimento de novos sistemas computacionais buscam suprir a demanda de alto desempenho proveniente do crescimento de simulações e pesquisas científicas. Atualmente, a maioria das aplicações paralelas apresentam comportamentos dinâmicos com cálculos baseados em fórmulas complexas. Por conta disso, empresas e instituições buscam adquirir uma infraestrutura suficiente para suportar a demanda computacional destas aplicações [Arruda et al. 2015]. Um problema surge devido ao fato que a maioria destas aplicações apresentam desbalanceamento de carga, impedindo um uso eficiente dos recursos computacionais das máquinas paralelas [Padoin et al. 2014]. Diante deste problema, balanceadores de carga são desenvolvidos almejando melhor distribuir de cargas das tarefas entre as unidades de processamento.

Nesse sentido, este artigo apresenta a proposta de um novo balanceador de carga (BC) denominado SMARTLB. Nossa proposta almeja a redução do tempo total de execução da aplicação por meio da migração de tarefas considerando a carga atual dos cores e das tarefas.

O restante do trabalho está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 apresenta a proposta do balanceador de carga SMARTLB. A Seção 4 descreve a metodologia utilizada na implementação e o ambiente de execução utilizado na realização dos testes. Resultados são discutidos na Seção 5, seguidos das Conclusões e Trabalhos Futuros.

2. Trabalhos Relacionados

Várias aplicações científicas adotam estratégia centralizada de balanceamento de carga. Neste modelo, as decisões de balanceamento de carga são realizadas em um processador específico, com base nos dados de carga tempo de execução [Zheng et al. 2010]. Outras, por sua vez recorrem a esquemas de balanceamento de carga com diferentes estratégias e constroem seu próprio modelo de carga de trabalho para orientar a atribuição de trabalho aos processos.

*Trabalho parcialmente apoiado por UNIJUI e CNPq. Pesquisa tem recebido recursos do edital da VRPGPE de bolsa e PIBIC/UNIJUI.

Balanceadores de carga tem sido desenvolvidos almejando uma melhor distribuição das tarefas entre as unidades de processamento. Do ambiente CHARM++, considerou-se os seguintes balanceadores de carga para implementação da nossa proposta:

- **GREEDYLB**: é um algoritmo guloso. Seu paradigma é frequentemente usado na teoria e na prática de otimização combinatória fazendo com que a grande maioria das tarefas sejam migradas a cada iteração [Bang-Jensen et al. 2004];
- **REFINELB**: éalgoritmo que move tarefas dos cores mais carregados para os menos carregados almejando atingir uma média limitando a quantidade de migrações; e
- **AVERAGELB**: constitui-se de uma melhoria da estratégia do algoritmo GREEDYLB. Seu algoritmo leva em consideração a média aritmética do processador para decidir quais tarefas serão migradas [Freytag et al. 2015].

3. SmartLB

A estratégia utilizada para implementação do balanceamento de carga proposto constitui-se de melhorias nas estratégias utilizadas nos algoritmos GREEDYLB e REFINELB. Nossas melhorias buscam equilibrar as cargas entre os processadores reduzindo o número de migrações. Para tanto, é adotado um valor de *threshold* que determina o desbalanceamento de carga aceitável.

Quando o balanceador é chamado, ele primeiramente verifica as cargas do cores mais e menos carregado e a diferença de carga (desbalanceamento de carga) entre eles. Caso essa diferença for maior que o *threshold* , o algoritmo seleciona tarefas com carga menor ou igual ao desbalanceamento para mover do core mais carregado para o menos carregado.

O SMARTLB foi desenvolvido utilizando o framework de balanceamento de cargas disponibilizado pelo CHARM++. Este framework de balanceamento de carga foi escolhido uma vez que permite tanto a criação de novos BC, quanto a utilização dos BCs disponibilizados pelo ambiente para comparações de resultados.

4. Metodologia

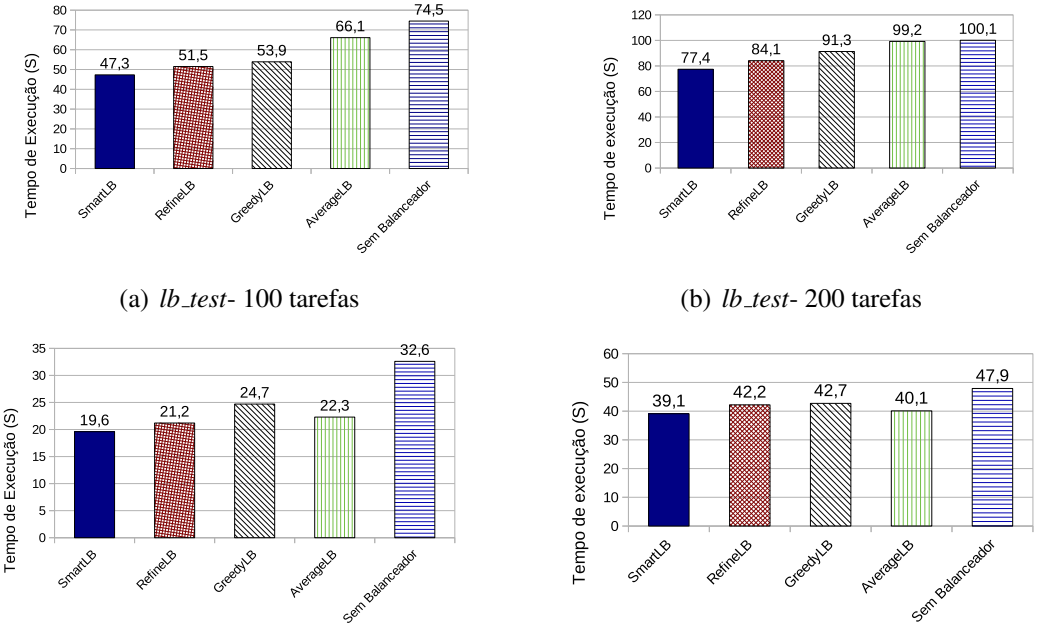
Para analisar os resultados alcançados com o balanceador de carga proposto, testes foram realizados utilizando-se 2 benchmarks disponibilizados pelo ambiente de programação CHARM++. O primeiro, *lb_test*, foi configurado com carga computacional que varia entre 1500ms e 1500000ms. O segundo, *kNeighbor* foi configurado para realizar testes com 100000 mensagens. Ambos os benchmarks foram executados com 150 iterações por tarefa e com sincronização do balanceador a cada 10 iterações. Em ambos os testes foi adotado um *threshold* de valor igual a 5%.

O desempenho alcançado com o BC proposto foi comparado com dois balanceadores de carga disponíveis no CHARM++, o GREEDYLB e o REFINELB; e com o proposta do AVERAGELB. O equipamento utilizado nos testes possui um processador Intel Core i7-4790 de 8 cores. Possui sistema operacional Linux Ubuntu 16.04 com kernel versão 4.4.33. A versão do CHARM++ utilizada para implementação foi a 6.5.1 e compilador g++ de versão 5.4.1.

5. Resultados

Na Figura 1 são apresentados os tempos de execução dos testes realizados com o benchmark *lb_test* e *kNeighbor* para diferentes quantidades de tarefas.

Figura 1. Tempos de execução mensurados durante a execução



(a) *lb_test*- 100 tarefas

(b) *lb_test*- 200 tarefas

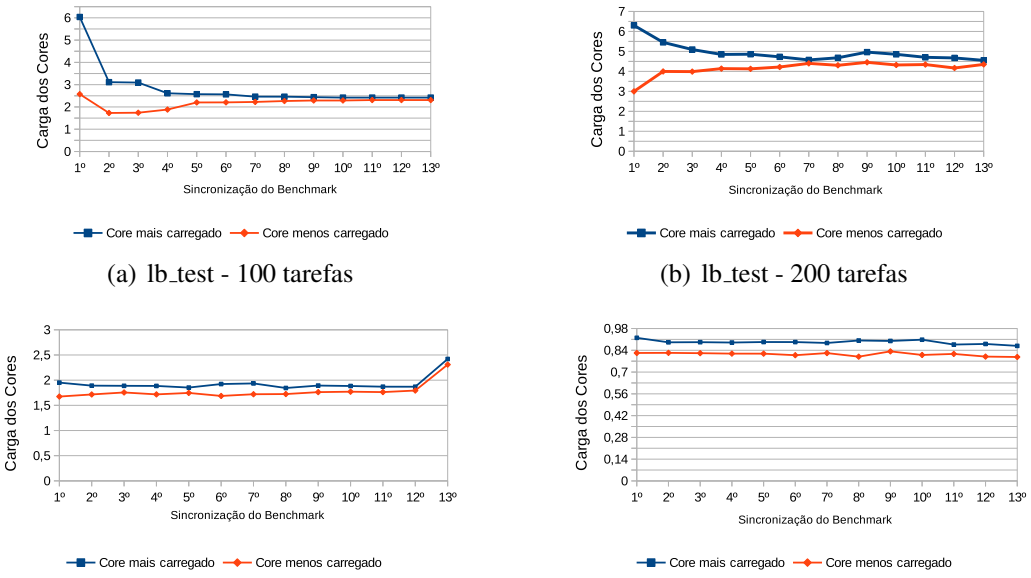
(c) *kNeighbor*- 100 tarefas

(d) *kNeighbor*- 200 tarefas

O BC SMARTLB apresentou melhor desempenho para ambos os benchmarks testados. Para lb_test com 100 tarefas o SMARTLB conseguiu reduzir o tempo para 47,3 segundos, o que representa uma redução de $36,5\%$ em relação a execução sem balanceador e $28,52\%$ em relação ao AVERAGELB. Com 200 tarefas o SMARTLB também apresentou redução de $22,64\%$ em relação a execução sem balanceador e, $7,9\%$ menor que o REFINELB. Quando utilizado com o benchmark *kNeighbor*, o SMARTLB também obteve os menores tempos de execução. Ele foi $20,41\%$ e $8,45\%$ menor que GREEDYLB para 100 e 200 tarefas respectivamente.

Na Figura 2 são apresentados os desbalanceamentos de carga mensurados a cada chamada do balanceador de carga proposto. Em todos os testes, o SMARTLB conseguiu concluir a execução dos benchmarks apresentando o menor de desbalanceamento. Nos testes com o lb_test com 100 e 200 tarefas, na primeira chamada do balanceador a diferenças entre o core mais carregado e o core menos carregado era de $3,46$ e $3,30$ unidades respectivamente. O SMARTLB conseguiu reduzir, até a última sincronização, essas diferenças para $0,12$ e $0,20$ respectivamente. Nos testes realizados com o benchmark Kneighbor com 100 e 200 tarefas, as diferenças entre os cores mais carregados e menos carregados eram de $0,10$ e $0,28$. Com a utilização do SMARTLB, essas diferenças diminuíram para $0,07$ e $0,11$ respectivamente.

Figura 2. Desbalanceamento mensurado durante a execução dos benchmarks



(a) lb_test - 100 tarefas

(b) lb_test - 200 tarefas

(c) *kNeighbor*- 100 tarefas

(d) *kNeighbor*- 200 tarefas

6. Conclusões e trabalhos futuros

Este trabalho apresentou a proposta de um novo balanceador de carga denominado SMARTLB. Os resultados alcançados com o SMARTLB mostraram-se muito consistentes. Conseguiu-se reduzir o nível de desbalanceamento de carga e consequentemente o tempo de execução quando aplicado nos dois benchmarks selecionados.

Como futuros trabalhos, pretende-se realizar melhorias no algoritmo de tomada de decisão do SMARTLB de modo a melhorar o controle de migrações de tarefas. Pretende-se também realizar testes em sistemas paralelos maiores utilizando problemas reais de computação científica bem como comparar com outros balanceadores de carga do estado da arte.

Referências

Arruda, G., Padoin, E. L., Pilla, L. L., Navaux, P. O. A., and Mehaut, J.-F. (2015). Proposta de balanceamento de carga para redução de migração de processos em ambientes multiprogramados. In *XVI Simpósio de Sistemas Computacionais (WSCAD-WIC)*, pages 1–8.

Bang-Jensen, J., Gutin, G., and Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization*, 1(2):121–127.

Freytag, G., Arruda, G., Martins, R. S. M., and Padoin, E. L. (2015). Análise de desempenho da paralelização do problema de caixeiro viajante. In *XV Escola Regional de Alto Desempenho (ERAD)*, pages 1–4, Gramado, RS. SBC.

Padoin, E., Castro, M., Pilla, L., Navaux, P., and Mehaut, J.-F. (2014). Saving energy by exploiting residual imbalances on iterative applications. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10.

Zheng, G., Meneses, E., Bhatele, A., and Kale, L. V. (2010). Hierarchical load balancing for charm++ applications on large supercomputers. In *2010 39th International Conference on Parallel Processing Workshops*, pages 436–444. IEEE.

Suporte ao Paralelismo Multi-Core com FastFlow e TBB em uma Aplicação de Alinhamento de Sequências de DNA

Júnior Löff[1], Dalvan Griebler[1], Edans Sandes[2], Alba Melo[2], Luiz G. Fernandes[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Grupo de Modelagem de Aplicações Paralelas (GMAP), Porto Alegre – RS – Brasil

junior.loff@acad.pucrs.br

[2]Universidade de Brasília (UnB) – Brasília – DF – Brasil

Resumo. *Quando uma sequência biológica é obtida, é comum alinhá-la com outra já estudada para determinar suas características. O desafio é processar este alinhamento em tempo útil. Neste trabalho exploramos o paralelismo em uma aplicação de alinhamento de sequências de DNA utilizando as bibliotecas FastFlow e Intel TBB. Os experimentos mostram que a versão TBB obteve até 4% melhor tempo de execução em comparação à versão original em OpenMP.*

1. Introdução

As sequências biológicas podem ser classificadas em: DNA, RNA e sequência de proteínas. Para estes dois últimos, as sequências são menores e atingem no máximo alguns milhares de caracteres. Em contrapartida, as sequências de DNA são bem mais longas, podendo chegar a milhões de nucleotídeos [Sandes et al. 2016]. Na literatura, vários algoritmos foram propostos para a computação de alinhamento de sequências de DNA. Dentre eles podemos citar dois importantes: o algoritmo Needleman-Wunsch (NW) e o algoritmo Smith-Waterman (SW). No trabalho [Sandes and de Melo 2013] foi introduzido o CUDAlign, uma aplicação real para alinhamento de sequências de DNA que implementa otimizações paralelas para GPUs nos algoritmos NW e SW.

Através do trabalho [Sandes et al. 2016], foi identificado que 90% do código do CUDAlign é independente da plataforma e apenas 10% do código é específico para GPUs. Com base no código do CUDAlign, a arquitetura MASA (*Multiplatform Architecture for Sequence Aligners*) [Sandes et al. 2016] foi proposta como uma solução para aplicações de alinhamento em plataformas heterogêneas. Atualmente, o MASA foi implementado em três versões: uma implementação em OpenMP visando o acelerador Intel Phi e duas versões para CPU implementadas em OpenMP e OmpSs.

O objetivo do trabalho é introduzir o suporte ao paralelismo encontrado nas CPUs (multi-core) com FastFlow (FF) e Thread Building Blocks (TBB) e realizar uma análise do desempenho da aplicação MASA paralelizada. Ambas bibliotecas implementam a linguagem padrão C++, por isso divergem das versões já implementadas no MASA, cujas bibliotecas utilizam diretivas de compilação. Através deste trabalho, vários estudos que utilizam o alinhamento de sequências de DNA podem ser beneficiados. Podemos citar alguns, como: (1) Biologia evolutiva, para estudo dos diferentes organismos e suas relações, identificando a origem e descendência das espécies, assim como sua mudança ao longo do tempo (evolução). (2) Medicina, para identificar, diagnosticar e efetuar tratamentos para doenças genéticas. (3) Ciência forense, amplamente aceita para investigação de crimes e resolução de questões cíveis, penais ou administrativas (como por exemplo, o

teste de paternidade). Este trabalho está organizado da seguinte maneira. A seção 2 contém os trabalhos relacionados. A seção 3 apresenta a aplicação MASA e a seção 4 contém os detalhes da implementação paralela da aplicação. Por fim, na seção 5 são discutidos os resultados obtidos, cujas conclusões são descritas na seção 6.

2. Trabalhos Relacionados

Na literatura, vários trabalhos que contribuem no estudo das técnicas de alinhamento de sequências de DNA podem ser citados. Em [Rajko and Aluru 2004], foi proposto um algoritmo de alinhamento para CPUs que diminui a complexidade do tempo de execução e do armazenamento sem apresentar trocas. No trabalho [Farrar 2007], foram utilizadas instruções *Single-Instruction Multiple-Data* (SIMD) para paralelizar o algoritmo SW à nível de instruções. Em [Aldinucci et al. 2010], foi continuado o trabalho descrito anteriormente, onde estendeu-se o algoritmo SW para as seguintes bibliotecas paralelas: OpenMP, FastFlow, TBB e Cilk. Este artigo merece uma atenção especial, pois é familiar com a nossa pesquisa. O nosso trabalho difere deste pois iremos utilizar como estudo o MASA, uma aplicação de alinhamento que introduz otimizações sobre os algoritmos SW e NW. Além do mais, pretendemos estender o escopo de memória compartilhada já implementado no MASA com OpenMP e OmpSs (diretivas de compilação), utilizando bibliotecas oriundas da linguagem padrão do C++, como TBB e FastFlow.

3. A Arquitetura MASA

MASA é uma arquitetura flexível e customizável para alinhamento de sequências de DNA em diferentes plataformas de *hardware/software*. Esta arquitetura foi desenvolvida em C++ com o paradigma OO. A computação é distribuída entre seis estágios que dividem o processamento do alinhamento. O primeiro estágio, descrito na Figura 1, representa a computação intensa da aplicação, onde a matriz é processada, o escore do alinhamento ótimo (representado pelo x) é calculado e linhas especiais são salvas em memória. Ademais, blocos sem potencial podem ser detidos através da otimização `block pruning`, reduzindo a carga a ser processada. No Segundo, terceiro e quarto estágio a computação é similar, por isso abstraímos o processamento em uma única imagem. Estes estágios representam a computação do aprimoramento/redução da região potencial a conter alinhamentos. Já no quinto estágio, representado pelo último bloco da Figura 1, o alinhamento de DNA é realizado sobre o subconjunto reduzido dos milhões de nucleotídeos das sequências originais. No sexto estágio (opcional), os resultados são visualizados de maneira gráfica ou textual. Um exemplo de alinhamento de DNA é demonstrado a seguir.

```
TGTGG-CGGTCAAAACCC- - -A- - -CTATTTTCTGGATGTAGCGCAGCCTGGT
 |||||  ||||  ||  ||||    |    ||||||||||||||||||||||||||
TGTGGTCGGTTAATACCCCTTATTTCTATTTTCTGGATGTAGCGCAGCCTGGT
```

4. Projeto MASA-TBB e MASA-FastFlow

A arquitetura MASA facilita a implementação de aplicações de alinhamento de sequências em diversas plataformas. Deste modo, utilizou-se o código original do MASA-OpenMP para construir as implementações MASA-FastFlow e MASA-TBB. Atualmente, as bibliotecas FastFlow e TBB apresentam limitações na abstração do paralelismo em implementações bem estruturadas, como está disposto o MASA. Isto exigiu uma pequena

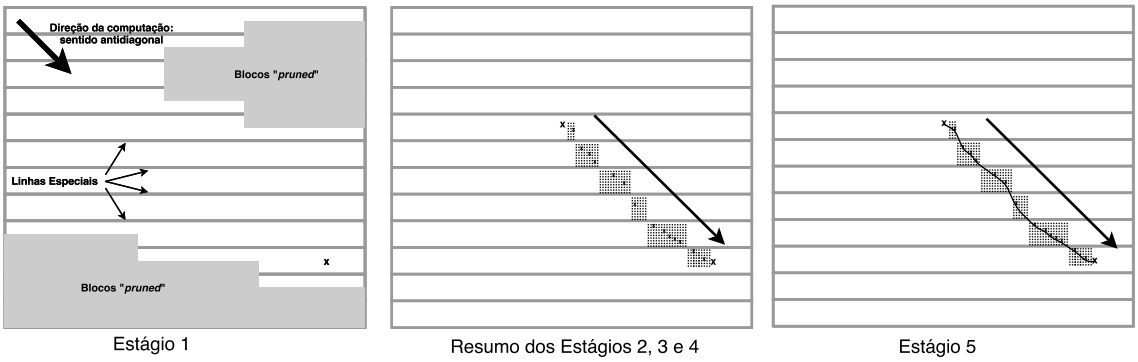


Figura 1. Abstração dos estágios de execução do MASA.

refatoração do código para habilitar o paralelismo. Nos estudos verificamos que a parte mais custosa está no primeiro estágio, onde a matriz de programação dinâmica é calculada. Neste processamento, as tarefas são dependentes, para isto foi utilizado a varredura em direção anti-diagonal (representada pela flecha da Figura 1) onde os dados se comportam independentemente, podendo ser executados em paralelo. Outro desafio foi conseguir obter acesso aos métodos OO após introduzir a biblioteca paralela. Isto acontece porque cada biblioteca cria o seu próprio escopo que é abstraído aos olhos do programador. Para resolver isto, criamos um ponteiro que aponta para a estrutura `this` e o passamos como parâmetro para dentro do escopo paralelo. O esqueleto paralelo utilizado foi um `map` dos blocos gerados após a divisão da matriz com quantidade de blocos igual ao número especificado como parâmetro na função `setPreferredSizes()`.

5. Resultados

Os experimentos foram executados em uma máquina equipada com 24 GB de RAM e dois processadores Intel(R) Xeon(R) CPU E5-2620 v3 2.40GHz com 6 núcleos cada e função hyper-threading, totalizando 24 núcleos. O sistema operacional era Ubuntu Server 64 bits com kernel 4.4.0-59-generic. Além disso, utilizamos o compilador GCC 5.4.0 com a otimização `-O3` e as bibliotecas: Thread Building Blocks (4.4 20151115) e FastFlow (r13). Os testes foram repetidos 5 vezes para cada amostra, onde foi utilizado a média aritmética e calculado o desvio padrão que está plotado no gráfico. Foram utilizadas sequências de DNA reais obtidas do National Center for Biotechnology Information (NCBI) referentes à duas bactérias: AE002160.2 (*Chlamydia muridarum* Nigg) e CP000051.1 (*Chlamydia trachomatis* A/HAR-13), ambas com aproximadamente 1 milhão de nucleotídeos. A matriz de programação dinâmica resultante possui 1,12 trilhão de células.

Nos nossos testes, diferente de executar toda a aplicação, focamos no primeiro estágio, que é o mais custoso e possui o maior potencial para usufruir do paralelismo intenso. Os resultados representados pela Figura 2 são referentes à execução do primeiro estágio sem a otimização `block pruning`. Neste gráfico, representa-se a relação de tempo de execução (eixo das ordenadas) por número de *threads* (eixo das abscissas). As versões em OpenMP, FastFlow e Intel TBB estão representadas no gráfico através das abreviações OMP, FF e TBB respectivamente. O desvio padrão ficou abaixo de 2% no pior caso. Pode ser visto que a versão MASA-FastFlow inicia melhor comparado à versão original MASA-OpenMP e perde desempenho logo em seguida. O MASA-TBB é sutilmente mais rápido que o MASA-OpenMP. Estes detalhes podem ser vistos mais

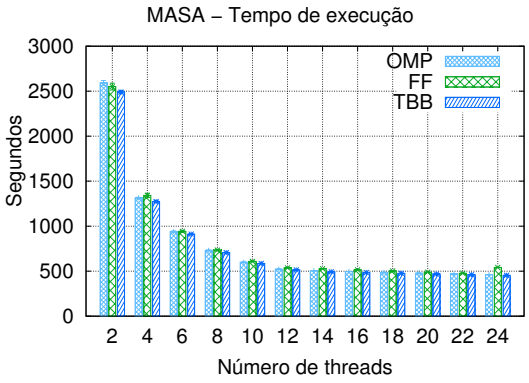


Figura 2. Desempenho do MASA.

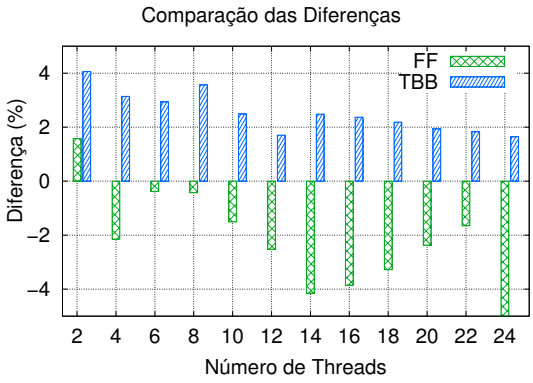


Figura 3. TBB e FastFlow *vs* OMP.

claramente na Figura 3, que representa a diferença do tempo de execução comparado com a versão original do MASA-OpenMP. Nesta figura fica evidente que a versão MASA-TBB executou em até 4% menos tempo que a versão do MASA-OpenMP. Ao chegar em 24 *threads*, a versão MASA-FastFlow ficou pior com um pico de 14% maior tempo de execução que a versão original. Isto acontece pois o FastFlow utiliza uma *thread* adicional para realizar o escalonamento.

6. Conclusões

Este trabalho apresentou a implementação do paralelismo em uma arquitetura de alinhamento de sequências de DNA, denominada MASA. Foram implementadas duas versões paralelas adicionais (MASA-TBB e MASA-FastFlow) e comparadas com a versão original MASA-OpenMP. O MASA-TBB atingiu $10,98$ de *speed-up* enquanto que o MASA-OpenMP obteve $10,79$. Além do mais, o MASA-TBB obteve até 4% de redução no tempo de execução em relação ao MASA-OpenMP, onde considerando a complexidade de uma aplicação de alinhamento, isto pode representar um considerável tempo de execução adicional. Como trabalhos futuros, pretendemos estender o paralelismo para os outros estágios através de uma análise específica de cada um deles, testar outras otimizações nas bibliotecas já implementadas e executar testes com sequências maiores de DNA.

Referências

[Aldinucci et al. 2010] Aldinucci, M., Meneghin, M., and Torquati, M. (2010). Efficient Smith-Waterman on Multi-core with FastFlow. In *PDP*, pages 195–199.

[Farrar 2007] Farrar, M. (2007). Striped Smith-Waterman speeds database searches six times over other SIMD implementations. In *Bioinformatics*, volume 23, pages 156–161.

[Rajko and Aluru 2004] Rajko, S. and Aluru, S. (2004). Space and time optimal parallel sequence alignments. *TPDS*, 15(12):1070–1081.

[Sandes and de Melo 2013] Sandes, E. and de Melo, A. C. M. A. (2013). Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU. *TPDS*, 24(5):1009–1021.

[Sandes et al. 2016] Sandes, E., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., and de Melo, A. C. M. A. (2016). MASA: A Multiplatform Architecture for Sequence Alignerswith Block Pruning. *TOPC*, 2(4):28:1–28:31.

Uma abordagem orientada a agrupamento de tarefas em balanceamento de carga distribuído

Vinicius M. C. T. de Freitas[1], **Laércio L. Pilla**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

vinicius.mct.freitas@gmail.com, laercio.pilla@ufsc.br

Resumo. *Estratégias de reescalonamento global garantem mais desempenho a aplicações de carga imprevisível ao manter uma distribuição eficiente de trabalho nas maiores plataformas paralelas. PackDrop é um algoritmo de escalonamento global distribuído que busca desempenho através do paralelismo em larga escala, enquanto diminui o sobrecusto de troca de mensagens levando o foco da tomada de decisão para o local.*

1. Introdução

Para simulações computacionais complexas de larga escala, como hidrodinâmica ou dinâmica molecular, ganho de desempenho é crucial para que o processamento seja concluído em tempo razoável. Mesmo as maiores plataformas paralelas sofrem ao tentar garantir esse ganho às aplicações, devido a problemas como custos de comunicação e desbalanceamento de carga [Pilla 2014]. Como as cargas dessas simulações podem ser instáveis, um escalonamento inicial satisfatório pode se mostrar inefetivo no futuro.

Estratégias de reescalonamento global podem ser usadas para mitigar este problema, movendo carga de trabalho para unidades de processamento (PUs) diferentes. Contudo, elas devem tomar pouco tempo para que não impactem a aplicação negativamente [Deveci et al. 2015].

Este trabalho apresenta **PackDrop**, uma estratégia de mapeamento global distribuído baseada em refinamento. Ela faz uso de informação local para criar e migrar grupos de tarefas, buscando reduzir tanto a comunicação quanto os custos de migração. Por sua natureza paralela, essa abordagem é capaz de aproveitar grandes sistemas evitando gargalos de comunicação e execução.

2. Metodologia

O processo de tomada de decisão em estratégias de escalonamento global pode ser abstraído para três grandes etapas, sendo estas: (i) dissipação de informação sobre o estado do sistema; (ii) tomada de decisão para o remapeamento de tarefas; e (iii) migração de tarefas.

Abordagens de escalonamento global distribuído previamente propostas avaliam cada tarefa individualmente para migração. Apesar de um balanceamento de carga mais preciso provido por esse tipo de abordagem [de Freitas and Pilla 2017], isso também pode levar a um aumento no tempo de decisão da estratégia.

Em **PackDrop** um conceito diferente é apresentado. Ao agrupar tarefas para migração, avaliando-as como grupo ao invés de individualmente, o foco da tomada de

decisão é local, reduzindo a comunicação nessa etapa da estratégia. Isso permite que menos informação seja compartilhada e uma tomada de decisão mais rápida seja realizada, ganhando desempenho nos passos (i) e (ii) do escalonamento.

O sistema paralelo utilizado neste trabalho foi o `Charm++`, que se mostrou ideal para a implementação do *PackDrop*. Ele é escalável e usado em diversas aplicações científicas, além de apresentar um *framework* de balanceamento de carga independente de aplicação [Acun et al. 2016], além de apresentar uma série de *benchmarks* e implementações de balanceadores de carga.

3. Estratégia Proposta: *PackDrop*

A Figura 1 mostra o fluxo de funcionamento do algoritmo principal, descrito a seguir:

Os dois primeiros passos (1), executados por todos os PUs, são realizar duas reduções, a primeira servindo para agregar a informação sobre a carga média e a segunda sobre o número de tarefas no sistema. Em seguida, cada um dos PUs se divide entre sobrecarregados e subcarregados (2).

Aqueles que são sobrecarregados irão começar a criar seus pacotes para transferência, enquanto os demais irão começar a propagar seus identificadores pela rede através de um protocolo *Gossip* [Menon and Kalé 2013], com o objetivo de informar os sobrecarregados a quem eles devem enviar suas tarefas (3).

Todos os processadores são sincronizados (4) para se iniciar o processo de tomada de decisão (5). A tomada de decisão é feita em conjunto pela fonte e destino de um pacote. Um PU sobrecarregado escolherá aleatoriamente uma série de PUs subcarregados a quem enviará pacotes. Os subcarregados devem calcular sua nova carga após o recebimento de um pacote e responder ao PU sobrecarregado, confirmando ou cancelando o remapeamento. Caso o envio seja cancelado, o PU sobrecarregado tentará enviar o pacote a outro subcarregado. Este processo se assemelha muito a um *three-way handshake* e garante que um PU subcarregado não receba pacotes demais, tornando-se o novo gargalo da aplicação paralela.

Esse processo descreve os passos (i) e (ii) apresentados na Seção 2, enquanto o passo (iii) é resolvido pelo *framework* do `Charm++`.

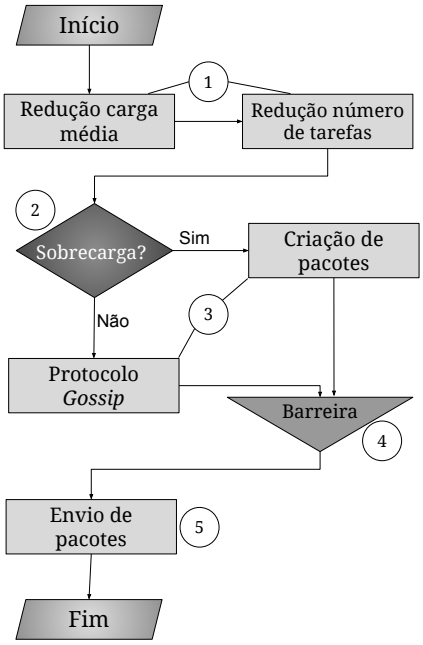


Figura 1. Fluxo do *PackDrop*, seguido por todos os PUs.

4. Avaliação de Desempenho

O principal ganho esperado com a abordagem aqui apresentada é **diminuição no volume de comunicação**. Isso acontece pois um PU é capaz de aceitar múltiplas migrações de uma vez, reduzindo o tempo de decisão.

Dentre os ***benchmarks*** utilizados para teste e avaliação, dois são perfis paralelos sintéticos e dois são *mini-apps* de aplicações de mundo real. Seus parâmetros estão discutidos na Tabela 1.

Benchmark	Dimensão	Comunicação
lb test	18990 (*med*); 36990 (*big*) tarefas	Anel
Stencil 3D	540 (bloco) e 12 (*array*)	Malha tridimensional
LULESH	140 (bloco) e 7 (*array*)	Malha tridimensional
LeanMD	$10 \times 15 \times 10$ (*med*); $15 \times 20 \times 15$ (*big*)	Muitos para muitos

Tabela 1. Parâmetros dos testes para avaliação de desempenho de cada um dos *Benchmarks*.

- **lb test**[1] : perfil paralelo para teste de balanceadores de carga.
- **Stencil 3D**: malha tridimensional que simula uma simples dissipação de calor.
- **LULESH**[2]: *mini-app* de simulações hidrodinâmicas. Segue o padrão computacional estêncil [Pereira et al. 2017].
- **LeanMD**[3]: simulação molecular baseado no potencial de *Lennard-Jones*.

4.1. Estratégias de Escalonamento

- **Greedy**: algoritmo de balanceamento centralizado guloso. Mapeia tarefas para núcleos independente do número de migrações.
- **Refine**: centralizado e baseado em refinamento, busca minimizar o número de migrações de tarefas, enviando-as de núcleos sobrecarregados para subcarregados.
- **Hybrid**: hierárquico, trabalha com diferentes algoritmos dependendo da hierarquia. Usa o *Greedy* dentro dos grupos e o *Refine* entre grupos.
- **Grapevine**: estratégia distribuída e baseada em refinamento. Distribuição probabilística de carga.

4.2. Ambiente de Testes

Todos os testes preliminares foram rodados em 16 nós do *cluster* **Genepi**[4], cada um contando com 2 processadores *Intel Xeon* E5420 QC @ 2.5GHz com 4 *cores* cada, totalizando **128 PUs**, 8GB de memória RAM DDR3 @ 1333MHz e 160GB de armazenamento (HDD). Os nós estão interconectados numa rede *InfiniBand* 20G. O sistema operacional utilizado foi o Ubuntu 14.04, a versão do `Charm++` adotada foi a 6.7.0 e o `GCC` foi utilizado na versão 5.4.0.

4.3. Resultados

A Figura 2 mostra os resultados em $speedup - 1$ relativos a execuções sem balanceamento de carga. Os resultados com ganhos tornam-se positivos e os demais negativos ou nulos.

Os *benchmarks* com padrões estêncil em malhas tridimensionais tiveram um impacto negativo do balanceamento de carga.

Balanceadores distribuídos destacam-se em comparação aos centralizados e hierárquicos conforme o tamanho do sistema cresce [Menon and Kalé 2013]. Portanto, apesar dos resultados semelhantes ao *Refine* em muitos casos, a tendência é que com o crescimento do sistema, a diferença entre eles também cresça.

[1]Disponibilizados com o `Charm++` através de: `http://charmplusplus.org/download/`

[2]Disponível em: `https://codesign.llnl.gov/lulesh.php`

[3]Disponível em: `http://charmplusplus.org/miniApps/`

[4]Experimentos apresentados neste trabalho foram realizados no Grid5000, mantido pelo grupo de propósitos científicos hospedado pela Inria e incluindo CNRS, RENATER e diversas Universidades e outras organizações (ver `https://www.grid5000.fr`).

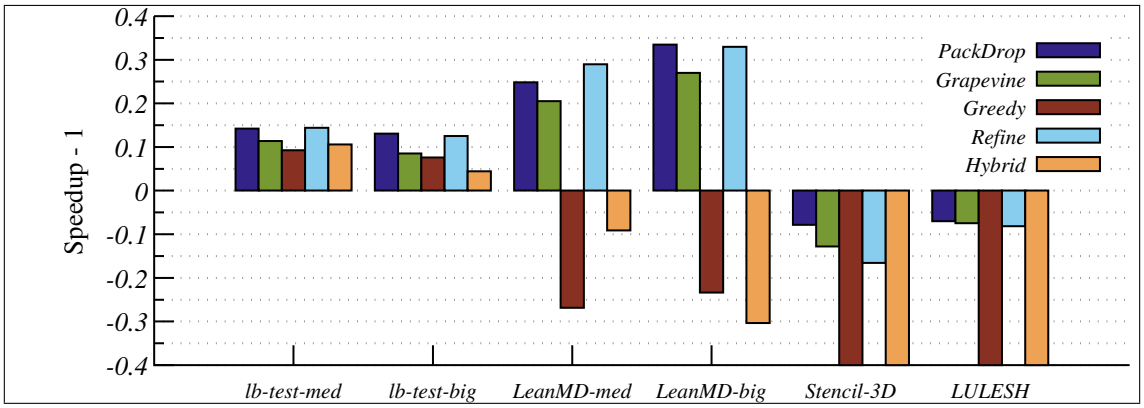


Figura 2. $Speedup - 1$ **de todos os *benchmarks* e aplicações testadas com cada estratégia de balanceamento de carga.**

5. Conclusão

Este trabalho apresentou *PackDrop*, uma estratégia de escalonamento global seguindo uma *abordagem orientada a pacotes*. O principal diferencial dela é a criação de pacotes assim como possibilita a migração de múltiplas tarefas de uma única vez.

Essa nova abordagem mostrou resultados promissores, tendo ganho de desempenho em todos os casos de teste experimentados em relação a outra estratégia distribuída observada, o *Grapevine*.

5.1. Trabalhos Futuros

É importante levar o *PackDrop* a plataformas de mais larga escala, uma vez que os testes realizados neste trabalho ainda não tem a dimensão necessária para mostrar o potencial das estratégias distribuídas.

Referências

Acun, B., Langer, A., Meneses, E., Menon, H., Sarood, O., Totoni, E., and Kalé, L. V. (2016). Power, reliability, and performance: One system to rule them all. *Computer*, 49(10):30–37.

de Freitas, V. M. C. T. and Pilla, L. L. (2017). Comparando diferentes ordenações de tarefas em balanceamento de carga distribuído. In *Anais da Escola Regional de Alto Desempenho do Rio Grande do Sul 2017*.

Deveci, M., Kaya, K., Uçar, B., and Catalyurek, U. V. (2015). Fast and high quality topology-aware task mapping. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 197–206.

Menon, H. and Kalé, L. (2013). A distributed dynamic load balancer for iterative applications. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 15:1–15:11, New York, NY, USA. ACM.

Pereira, A. D., Castro, M., Dantas, M. A. R., Rocha, R. C. O., and Góes, L. F. W. (2017). Extending openacc for efficient stencil code generation and execution by skeleton frameworks. In *2017 International Conference on High Performance Computing Simulation (HPCS)*, pages 719–726.

Pilla, L. L. (2014). *Topology-Aware Load Balancing for Performance Portability over Parallel High Performance Systems*. PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul.

Uma extensão ao Clang para identificação de laços de redução*

Vítor Resing Plentz[1], Edevaldo Santos[1], Gerson Geraldo H. Cavalheiro[1],

[1]Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

{vrplentz, edevaldo.santos, gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Na atualidade a maioria dos programas são construídos de forma sequencial, utilizando para a solução de muitos problemas o emprego de estratégias de execução iterativas, com laços de repetição. É de notório saber que grande parte do custo de processamento se dá nestes laços, não aproveitando o poder das arquiteturas paralelas. Neste artigo é apresentada uma extensão ao compilador Clang para identificação de laços paralelizáveis do padrão Reduce.*

Introdução

A popularização de arquiteturas multiprocessadas, pela tecnologia multicore, tornou acessível o uso de recursos de processamento paralelo em sistemas computacionais de qualquer porte. A consequência imediata é o aumento da demanda por programas que possam tirar benefício imediato do poder computacional disponível. Em outras palavras, desenvolver programas paralelos é, atualmente, uma necessidade e não mais uma opção para busca de melhora de desempenho. No entanto, nem todos os programadores possuem habilidades para identificar código que possa ser paralelizado, nem as ferramentas de programação oferecem recursos suficientes para paralelização automática de código em nível de tarefa.

Observando este fato, considera-se o interesse por uma ferramenta de identificação de padrões em programas que representem porções paralelizáveis. Neste trabalho é proposto o desenvolvimento de estratégias de análise de código fonte para identificação de tais padrões de forma a apresenta-los ao programador para que este possa reprogramar seu trecho de código na forma de um algoritmo paralelo na ferramenta de programação de sua escolha.

O caso de estudo desenvolvido neste trabalho considera o alto custo computacional gerado pelo uso de comandos iterativos (laços) nos programas e que tal recurso algorítmico ocorre com alta frequência em aplicações de diferentes domínios [Alba and Kaeli 2001, Bahi et al. 2007]. O contexto de desenvolvimento, neste momento, está restrito à identificação de padrões iterativos refletindo um modelo de execução de redução (*reduce*).

Foi desenvolvida uma extensão (*plugin*) ao compilador Clang para identificação de trechos de códigos iterativos com o padrão *reduce*. O restante deste artigo documenta o trabalho realizado e o *background* necessário para o mesmo.

*Financiado pela FAPERGS/CNPq PRONEX 16/2551-0000.

Background

Nesta seção são apresentados os conceitos necessários e as ferramentas escolhidas para este trabalho.

Algoritmo iterativo *reduce*

O padrão *reduce* é baseado em operações lógicas/matemáticas não associativas. A entrada neste padrão é dada por um *array*, uma variável de saída, um conjunto de operações e um espaço de iteração. Uma estrutura iterativa sobre o espaço de iteração executa o conjunto de operações apresentado sobre cada posição do *array*, armazenando o resultado de cada iteração, de forma acumulativa, sobre a variável de saída. Um exemplo de algoritmo de redução é a obtenção da soma dos valores de todos os elementos de um *array*.

Ferramental disponível com Clang

Clang [Cla] provê um front-end para compilação de linguagens C-like (C, C++, Objective C/C++, OpenCL) para a LLVM (Low Level Virtual Machine). Basicamente o Clang gera uma representação intermediária que é utilizada pela LLVM. Uma vez que o programa passa pelo Clang o mesmo já está apto para a LLVM em qualquer máquina. A opção por Clang é justificada neste trabalho por possuir possuir facilidades para o desenvolvimento de plugins customizados, usufruindo da estrutura e navegação de *AST's* (Abstract Syntax Tree), que facilitam a identificação de estruturas em um determinado código.

LibTooling - *Matcher* e *MatchFinder*

Entre as bibliotecas que o Clang possui suporte, se encontra o LibTooling. Esta biblioteca oferece primitivas para manipular a AST do código compilado pelo Clang. Com uso desta biblioteca, é possível realizar alterações no código fonte.

Esta biblioteca aplica o conceito de *matcher*. Um *matcher* representa uma expressão para busca por elementos específicos na AST. Os matchers podem ser de três tipos: Node Matchers, Narrowing Matchers e Traversal Matchers.

- **Node Matchers:** Podem ser definidos como o core das expressões, especificam o tipo do nó a ser buscado.
- **Narrowing Matchers:** São utilizados como suporte para os Node Matchers, correspondendo aos atributos do nodo.
- **Traversal Matchers:** Esse tipo Matcher específica a relação de um determinado nodo com outros nodos.

A classe MatchFinder::MatchCallback é utilizada em conjunto com os matchers, com essa classe é possível definir ações sobre um determinado nodo de uma AST que corresponde ao Matcher definido.

Metodologia

Neste trabalho foram definidos um *matcher* que corresponde ao padrão *reduce* e um MatchFinder que responde ao *matcher*.

Matcher

O *matcher* definido é representado pela Figura 1, e segue as seguintes características do padrão:

- Ser uma operação binária não associativa;
- Possuir como operandos um elemento de array e uma váriavel de saída;
- A operação binária deve estar dentro de um laço;
- A expressão não deve possuir mais de um operador (exceto operador de atribuição);
- O acesso ao array não deve possuir nenhuma operação.

```
StatementMatcher LoopMatcher = forStmt(hasLoopInit(declStmt(
                                hasSingleDecl(varDecl(hasInitializer(integerLiteral(equals(0)))).bind("initVarName"))),
                                hasIncrement(unaryOperator(hasOperatorName("++"),
                                  hasUnaryOperand(declRefExpr(to(varDecl(hasType(isInteger()))).bind("incVarName"))))),
                                hasCondition(
                                  binaryOperator(hasOperatorName("<"),
                                  hasLHS(ignoringParenImpCasts(declRefExpr(to(varDecl(hasType(isInteger()))).bind("condVarName")))),
                                  hasRHS(expr(hasType(isInteger())))))
                                ).bind("forLoop");
StatementMatcher arrayClause = arraySubscriptExpr(unless(hasDescendant(binaryOperator())),
                                  hasDescendant(declRefExpr(to(varDecl(hasType(isInteger()))).bind("arrayIndex"))));
StatementMatcher opClause[4] = {binaryOperator(hasOperatorName("+"),
                                  hasDescendant(arrayClause), hasDescendant(declRefExpr(to(varDecl().bind("accOperator")))), unless(hasDescendant(binaryOperator()))) ,
                                binaryOperator(hasOperatorName("-") ,
                                  hasDescendant(arrayClause), hasDescendant(declRefExpr(to(varDecl().bind("accOperator")))), unless(hasDescendant(binaryOperator()))) ,
                                binaryOperator(hasOperatorName("&") ,
                                  hasDescendant(arrayClause), hasDescendant(declRefExpr(to(varDecl().bind("accOperator")))), unless(hasDescendant(binaryOperator()))) ,
                                binaryOperator(hasOperatorName("|") ,
                                  hasDescendant(arrayClause), hasDescendant(declRefExpr(to(varDecl().bind("accOperator")))), unless(hasDescendant(binaryOperator())))
};
StatementMatcher ReduceMatcher =
    binaryOperator(hasOperatorName("="), hasParent(compoundStmt(hasParent(LoopMatcher))) , hasRHS(anyOf(
      opClause[0], opClause[1], opClause[2], opClause[3])), hasLHS(declRefExpr(to(varDecl().bind("accumulator"))))
    ).bind("reduce");
```

Figure 1. Reduce Matcher implementado no plugin

Verificação O trabalho de verificar se as variáveis utilizadas no acesso do array (índice do array) e a váriavel que é iterada no laço são a mesma é realizado no MatchFinder, como mostrado na Figura 2. No caso do padrão ser reconhecido o usuário é notificado.

```
class Finder : public MatchFinder::MatchCallback {
public :
  virtual void run(const MatchFinder::MatchResult &Result) {
      ASTContext *Context = Result.Context;
      const BinaryOperator *BO = Result.Nodes.getNodeAs<BinaryOperator>("reduce");
      // We do not want to convert header files!
      if(!BO || !Context->getSourceManager().isWrittenInMainFile(BO->getOperatorLoc()))
        return;
      const VarDecl *IncVar = Result.Nodes.getNodeAs<VarDecl>("incVarName");
      const VarDecl *CondVar = Result.Nodes.getNodeAs<VarDecl>("condVarName");
      const VarDecl *InitVar = Result.Nodes.getNodeAs<VarDecl>("initVarName");
      const VarDecl *ArrayIndex = Result.Nodes.getNodeAs<VarDecl>("arrayIndex");
      const VarDecl *AccOperator = Result.Nodes.getNodeAs<VarDecl>("accOperator");
      const VarDecl *Accumulator = Result.Nodes.getNodeAs<VarDecl>("accumulator");

      if(!areSameVariable(IncVar, CondVar) || !areSameVariable(IncVar, InitVar)){
        return;
      }else if(!areSameVariable(IncVar, ArrayIndex)){
        return;
      }else if(!areSameVariable(AccOperator, Accumulator)){
        return;
      }
    //  BO->dump();
      llvm::outs() << "Potential Reduce pattern, in the next line you'll find the file and the suspect line.\n";
      BO->getExprLoc().dump(Context->getSourceManager());
      llvm::outs() << "\n";
    }
};
```

Figure 2. MatchFinder implementado no plugin

Resultados Obtidos

Com a utilização do plugin criado, dado um código de entrada (Figura 3) nas linguagens suportadas pelo Clang que possuem algum trecho de código com o padrão *reduce*, a saída do plugin (Figura 4) exibe no terminal a linha onde o padrão foi identificado e que pode ser paralelizado.

```cpp
#include <iostream>

#define ARRAYSIZE 10


int main(){
    int array[ARRAYSIZE] = {0, 1 , 2, 3, 4, 5, 6, 7, 8, 9};
    int sum;
    for(int i = 0; i < ARRAYSIZE; i++){
        sum = sum + array[i]; //caracteriza o padrão reduce
    }
    for(int i = 0; i < ARRAYSIZE; i++){
        sum = sum + array[i+1]; //dependência de dados
    }
    int j = 0;
    for(int i = 0; i < ARRAYSIZE; i++){
        sum = sum + array[j]; //não caracteriza o padrão reduce definido
    }

    for(int i = 0; i < ARRAYSIZE; i++){
        sum = sum / array[i]; //operação associatiava
    }

    for(int i = 0; i < ARRAYSIZE; i++){
        sum  = sum + sum; // não envolve o vetor
    }

    for(int i = 0; i < ARRAYSIZE; i++){
        sum = sum + array[i] + sum; //não caracteriza o padrão reduce definido
    }
    return 0;
}
```

Figure 3. Exemplo de teste com código em C++ realizado no plugin

```
vplentz@vplentz-Inspiron-7460:~/clang-llvm/build$ ./bin/reduction tests/test_red.cpp --
Potential Reduce pattern, in the next line you'll find the file and the suspect line.
/home/vplentz/clang-llvm/build/tests/test_red.cpp:10:13
```

Figure 4. Saída do plugin, utilizando o arquivo de teste exibido acima.

Conclusão

Programar de forma sequencial apesar de muitas vezes ficar em desvantagem quanto ao desempenho da programação paralela, também possui características boas, como a fácil legibilidade do código e a agilidade para a criação do mesmo. Com o tipo de ferramenta proposta neste artigo, é possível aproximar os dois estilos de programação, desta forma o programador pode programar sequencialmente, usufruir de seus benefícios e após essa etapa, utilizar o plugin para melhorar o desempenho da aplicação.

Agradecimento

Externam-se agradecimentos à FAPERGS (Fundação de Amparo a Pesquisa do Estado do Rio Grande do Sul) e ao projeto GreenCloud pela concessão da bolsa de pesquisa.

References

Clang: a C language family frontend for LLVM. `https://clang.llvm.org/`. Acessado em: 07/01/2018.

Alba, M. d. R. and Kaeli, D. R. (2001). Runtime predictability of loops. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 91–98.

Bahi, J. M., Contassot-Vivier, S., and Couturier, R. (2007). *Parallel Iterative Algorithms: From Sequential to Grid Computing (Chapman & Hall/Crc Numerical Analy & Scient Comp. Series)*. Chapman & Hall/CRC.

Uso de Cache de Pilha em Processadores Atuais

Eduardo Guerra[1], **Francis B. Moreira**[1], **Phillipe O. A. Navaux**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{eguerra,fbmoreira,navaux}@inf.ufrgs.br

Resumo. *Em processadores de uso geral, é comum as aplicações utilizarem uma área de memória designada a uma pilha de execução, a fim de manter informação sobre o estado de subrotinas e armazenar variáveis locais a estas subrotinas. Dada a frequência de uso da pilha, otimizar os acessos a esta região de memória pode resultar em ganhos de desempenho na execução de aplicações. Sendo assim, este artigo tem como objetivo explorar o uso de uma memória cache dedicada exclusivamente à área de pilha na memória e analisar os impactos causados por ela no desempenho do processador.*

1. Introdução

A área de pilha é uma porção de memória contígua alocada por cada thread em execução, a qual mantém o contexto do escopo atual em que a aplicação se encontra. A sua alocação é feita no início da aplicação e pode crescer durante a execução, essa porção de memória empilha os endereços de retorno de subrotinas, assim como variáveis locais durante a execução de blocos de código. Neste artigo, é proposto o uso de uma memória cache adicional, a qual armazenaria apenas valores presentes na região da memória da pilha.

Além de remover um fluxo de dados da cache de dados, que não precisaria mais guardar os valores vindos da região da pilha, a nova memória cache pode ser acessada mais eficientemente, ao escolher 1 projeto focado em tempo de acesso. Assim, existe a possibilidade de aumentar a taxa de acertos na cache de dados e reduzir a latência para os acessos feitos na área da pilha. Espera-se portanto que o uso de uma cache de pilha resulte em um ganho de desempenho no processador.

2. Motivação

Para estudar a taxa de utilização da pilha, foi utilizado o Pin [Luk et al. 2005], um instrumentador dinâmico de binários da Intel. Através deste, foi possível contar o número total de acessos à memória realizados pela aplicação instrumentada, assim como o número de acessos à área de pilha na memória.

Como é possível ver na Tabela 1, o número médio de acessos à pilha é de 23% do número total de acessos para a carga de benchmarks paralelos da NASA (NPB) de classe A executados por 8 threads. Portanto, espera-se que reduzir o tempo de acessos à área de pilha leve a ganhos de desempenho no processamento, já que estes acessos constituem parte considerável dos acessos à memória. Atualmente, processadores convencionais comumente possuem uma latência de acesso à cache de dados de 4 ciclos. Porém, a latência de acesso à uma memória cache pode ser reduzida ao simplificar sua configuração.

Normalmente, a área de pilha é pequena em relação à aplicação, e possui alta localidade espacial e temporal [Alshegaifi and Huang 2016]. Assim, é possível reduzir a

Table 1. Porcentagem de acessos à região de memória da pilha.

Benchmark	Acessos à Pilha	Acessos Normais	Total	Porcentagem
bt.A.x	55196892056	98930861522	154127753578	36
cg.A.x	36949882	3191073910	3228023792	1
ep.A.x	6101896235	6656570233	12758466468	48
ft.A.x	396415853	8881748247	9278164100	4
is.A.x	75797435	1520603248	1596400683	5
lu.A.x	4380719119	16070986652	20451705771	21
mg.A.x	1318181417	5172286702	6490468119	20
sp.A.x	5836812962	81979881482	87816694444	7
ua.A.x	16571936500	85406001858	101977938358	16
Média	9990622384	34201112650	44191735035	23

latência de acesso ao utilizar uma cache menor e mais simples. Na Tabela 2 são demonstrados tempos de ciclo de caches do simulador FinCACTI [Shafaei et al. 2014] baseados em uma tecnologia de 7nm, para um processador de frequência 3.33 Ghz (0.3 ns de ciclo). Aqui, mostra-se que o mapeamento direto influi fortemente no tempo de acesso à cache. Este mapeamento normalmente gera conflitos entre endereços, resultando em subutilização do espaço da cache e uma taxa de erros mais alta. Porém, assumindo que a localidade da pilha em um escopo limita seu uso a regiões pequenas, essa desvantagem não existe.

Table 2. Tempos de acesso, em ciclos de processador, para parâmetros de cache diferentes.

Tamanho da Cache	Associatividade	Tempo de Ciclo	Ciclos de Processador
4 KB	1 via	0.221225 ns	1 ciclos
4 KB	2 vias	0.721745 ns	2 ciclos
4 KB	4 vias	2.64095 ns	9 ciclos
8 KB	1 via	0.221225 ns	1 ciclos
8 KB	2 vias	0.721745 ns	2 ciclos
8 KB	4 vias	2.64095 ns	9 ciclos
16 KB	1 via	0.221225 ns	1 ciclos
16 KB	2 vias	0.721745 ns	2 ciclos
16 KB	4 vias	0.721745 ns	2 ciclos
32 KB	8 vias	0.721745 ns	2 ciclos

Como discutido anteriormente, o tamanho da área de pilha possui localidade espacial entre escopos, e localidade temporal para um escopo. Assim, pode-se utilizar o mapeamento direto sem sofrer perdas significativas de desempenho causados por conflitos de linhas. Para os testes realizados na Seção de testes, o tamanho escolhido será 16KB, tornando possível o armazenamento simultâneo de múltiplas páginas de memória da pilha, enquanto mantém-se o tempo de acesso baixo.

3. Metodologia

Para realização dos testes de uso da região da pilha, foi utilizado o instrumentador Pin [Luk et al. 2005], versão 2.14. Para os experimentos de desempenho, foi utilizado o simulador ZSim [Sanchez and Kozyrakis 2013]. Para as simulações, o código fonte do

simulador foi modificado para detectar os acessos à área de pilha e direcioná-los para uma cache de pilha. A detecção é feita utilizando o instrumentador Pin-[Luk et al. 2005], que verifica se os acessos à memória foram realizados pelos registradores dedicados à pilha, *Stack Pointer* e *Base Pointer*. A configuração da máquina simulada possui 8 cores com execução fora de ordem, baseados na arquitetura Westmere [Kurd et al. 2010], com frequência de 3.3 Ghz. A configuração das caches está detalhada na Tabela 3. O conjunto

Table 3. Parâmetros das memórias cache do sistema simulado.

Cache	L1 Dados	L1 Pilha	L1 Instruções	L2	L3
Tamanho	32KB	16KB	32KB	256KB	8MB
Associatividade	8 vias	1 via	8 vias	4 vias	16 vias
Latência	4 ciclos	1 ciclo	4 ciclos	14 ciclos	34 ciclos

de benchmarks usado foi o NAS Parallel Benchmarks (NPB) [Bailey et al. 1991]. Todas aplicações foram executadas, com exceção à aplicação DC. A classe "A" foi usada para todos os benchmarks, por tratar-se da menor classe inclusa na lista de classes de problemas de testes padrão, reduzindo o tempo de simulação. Os benchmarks foram compilados com gcc 5.4.0.0.

4. Experimentos

Nesta sessão, é feita uma análise sobre o desempenho de um processador com cache para pilha. Os resultados referentes ao benchmark NPB são mostrados na Figura 1, expressos em números de ciclos de relógio utilizados pela thread principal de cada simulação. Na primeira coluna, encontra-se o número de ciclos utilizado pela aplicação sem a presença da cache de pilha. Na segunda coluna, encontra-se o número de ciclos utilizados pela mesma aplicação com a atuação da cache de pilha.

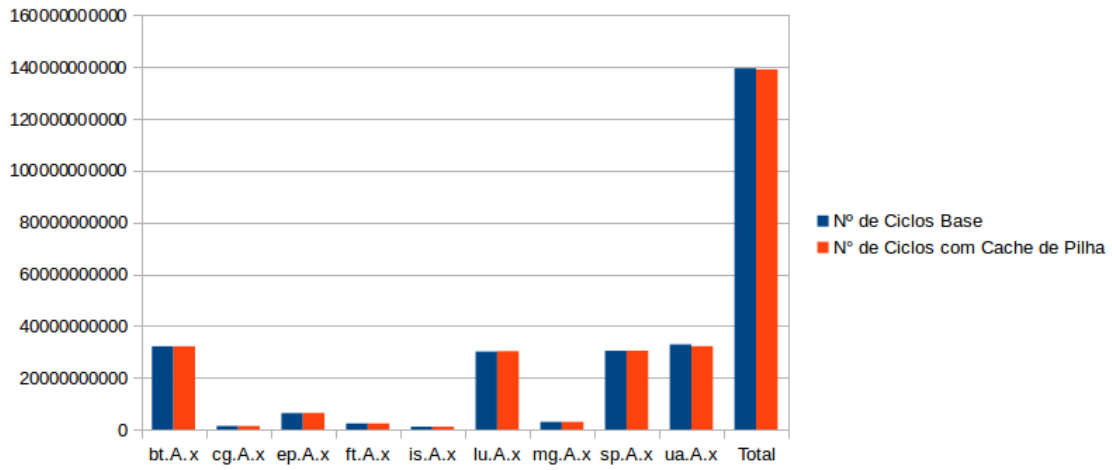


Figure 1. Tempo de execução em ciclos de processador para benchmarks NPB com 8 threads.

Como é possível observar na Figura 1, os resultados obtidos não condizem com a expectativa. A presença da nova memória cache teve impacto negligível. Aqui, é importante notar peculiaridades na implementação de coerência de cache do simulador, o qual apresentou piora no desempenho em testes com latência de invalidação reduzida. Apesar

da cache de pilha ser mais rápida, ela possui uma taxa de erros muito alta, o que seria justificável caso ocorra baixa localidade na área de pilha. Assim, temos evidência contrária aos resultados obtidos em [Alshegaifi and Huang 2016].

5. Conclusão

Os testes finais não demonstraram diferença significativa na execução dos benchmarks ao utilizar uma cache exclusiva para a área de pilha. Porém, existem indicativos de que a localidade espacial da cache de pilha precisa ser melhor explorada. No trabalho futuro será desenvolvido um *prefetcher* para utilizar a localidade espacial da cache de pilha. Deve-se também explorar o funcionamento da coerência de cache e analisar a implementação desta no simulador ZSim, o qual não parece suportar tal modificação.

Agradecimento

Esse trabalho recebeu fundos de MCTI/RNP-Brasil através do projeto HPC4E, segundo acordo nº 689772.

References

Alshegaifi, A. K. and Huang, C.-H. (2016). Impact of stack caches: Locality awareness and cost effectiveness. *World Academy of Science, Engineering and Technology, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, 10(4):545–551.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. (1991). The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5:63–73.

Kurd, N. A., Bhamidipati, S., Mozak, C., Miller, J. L., Wilson, T. M., Nemani, M., and Chowdhury, M. (2010). Westmere: A family of 32nm ia processors. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 96–97. IEEE.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200. ACM.

Sanchez, D. and Kozyrakis, C. (2013). Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 475–486. ACM.

Shafaei, A., Wang, Y., Lin, X., and Pedram, M. (2014). Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices. In *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pages 290–295. IEEE.

Fórum de Pós-Graduação

Gerson Geraldo H. Cavalheiro[1],
Dalvan Griebler[2]

O Fórum de Pós-Graduação da Escola Regional de Alto Desempenho é um espaço onde estudantes de mestrado e doutorado podem expor seus trabalhos em diversos estágios de desenvolvimento, instigados por recomendações de outros pesquisadores da área. Assim, o Fórum consolida-se através dos anos como um canal de discussão bastante produtivo que estimula a troca de ideias, atrai potenciais interessados e fomenta a interação e o aumento da produção científica da comunidade de Alto Desempenho do Estado do Rio Grande do Sul. Na edição deste ano do Fórum, 40 trabalhos foram aceitos para a apresentação. Os resumos foram avaliados por pelo menos três revisores, que procuraram fornecer comentários e sugestões para o aprimoramento dos trabalhos.

[1]Gerson Cavalheiro possui graduação em Informática pela Pontifícia Universidade Católica do Rio Grande do Sul (1990), mestrado em Computação pela Universidade Federal do Rio Grande do Sul (1994) e doutorado em Informatique Systèmes et Communications - Institut National Polytechique de Grenoble (1999). Atualmente é presidente adjunto do fórum de PG da Sociedade Brasileira de Computação - Porto Alegre e professor adjunto III da Universidade Federal de Pelotas. Tem experiência na área de Ciência da Computação, com ênfase em Processamento de Alto Desempenho, atuando principalmente nos seguintes temas: ambientes de execução, processamento de alto desempenho, aplicações, programação concorrente e ambiente de programação.

[2]Dalvan Griebler possui graduação em Redes de Computadores pela Sociedade Educacional Três de Maio - SETREM (2009), mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS (2012) na área de Processamento Paralelo e Distribuído (PPD), doutorado em Informática pela Universitá di Pisa - UNIPI (2016) na área de Modelos de Programação Paralela e doutorado em Ciência da Computação pela PUCRS (2016) na área de PPD. Atualmente é Pós-Doutorando na PUCRS, pesquisador associado no Grupo de Modelagem de Aplicações Paralelas - GMAP e Professor na SETREM. Foi fundador e atualmente é coordenador e pesquisador do Laboratório de Pesquisas Avançadas para Computação em Nuvem - LARCC da SETREM. Trabalha atualmente nos seguintes tópicos: Abstração de Alto Nível para Exploração do Paralelismo; Linguagens Específicas de Domínio; Projeto de Compiladores; Geração Automática de Código Paralelo; Programação Paralela; Computação em Nuvem; Plataformas de Gerenciamento de Nuvens; Computação de Alto Desempenho; Análise de Desempenho de Sistemas; e Redes de Computadores.

Abordagem Paralela de Evolução Diferencial Aplicado ao Problema de Predição de Estrutura de Proteína

Renan S. Silva[1], **Rafael S. Parpinelli**[1]

[1]Departamento de Ciências da Computação –
Universidade do Estado de Santa Catarina (UDESC) – Santa Catarina – SC – Brasil

renan.samuel.da.silva@gmail.com, rafael.parpinelli@udesc.com

Resumo. *Este trabalho explora a utilização do algoritmo de Evolução Diferencial paralelizado com MPI. O speedup obtido permite a utilização de operadores de busca de maior custo computacional. Os resultados preliminares indicam que há um speedup significativo da aplicação.*

Proteínas são macromoléculas essenciais para as formas de vida conhecidas, desempenhando papéis regulatórios, metabólicos e estruturais [Walsh 2002]. Sua estrutura tridimensional, que está diretamente associada a sua funcionalidade, é determinada experimentalmente em laboratório através de um processo lento e caro. Atualmente, o problema de predição de estrutura de proteína (PSPP) é considerado um problema em aberto tanto na Ciência da Computação quanto na Bio-informática estrutural e é considerado $\mathcal{NP}$-difícil [Guyeux et al. 2014]. Portanto, a utilização de métodos exatos se torna inviável até mesmo para instâncias de pequeno porte. Sendo assim, abre-se oportunidade para utilização de heurísticas e meta-heurísticas para solução do problema. Considerando a complexidade do problema envolvido, algoritmos meta-heurísticos básicos demonstram limitações e são incapazes de obter soluções significantes em instâncias mais complexas. Logo, o emprego de métodos avançados e a agregação de informações sobre o problema torna-se necessário para uma melhor solução do problema. Além da dificuldade do problema, tem-se ainda a necessidade de avaliar um grande número de funções para determinar a qualidade das soluções durante o processo de otimização, ocupando a maior parte do tempo de execução. Deste modo, a aplicação de uma arquitetura computacional paralela eficiente torna-se necessária para poder escalar a complexidade do PSPP. Com isto em mente, este trabalho tem como objetivo explorar paralelismo no algoritmo de Evolução Diferencial (*Differential Evolution - DE*). O algoritmo DE é um otimizador para problemas de domínio contínuo [Storn and Price 1997] e foi utilizado com sucesso no PSPP em [Narloch and Parpinelli 2017] e [Oliveira et al. 2017]. O modelo de ângulos e torsões é o empregado neste trabalho.

A proposta de paralelização consiste em utilizar uma estratégia *master-slave* para efetuar a avaliação da energia potencial residual das conformações. O processo *master* é responsável por executar todas as operações relacionadas ao DE enquanto que os processos *slave* são responsáveis pela avaliação das conformações. Para medir o desempenho do método mediu-se o tempo médio total gasto em cada execução do DE. Variou-se o número de processos *slave* entre 1 e 4 e comparando com a implementação serial. A máquina utilizada para testes possui um processador Intel Core i5-3570k com 4 núcleos e 16 Gb de RAM. O DE foi executado 10 vezes com os seguintes parâmetros: $Cr = 1.0$ e $F = 0.5$, 5000 gerações e 100 indivíduos. Os testes foram conduzidos na proteína 1ZDD, que possui 34 aminoácidos e 179 ângulos a serem otimizados. Como esperado, as

	Tempo de execução	Speedup
Serial	72m	1.00
1 *slave*	74m	0.97
2 *slaves*	43m	1.82
3 *slaves*	34m	2.07
4 *slaves*	56m	1.28

Tabela 1. Tempo médio por execução e *speedup*

aplicações com e sem paralelização não apresentam diferenças significativas nos resultados da otimização.

Na Tabela 1 pode-se observar que o aumento do número de processos *slave* reduz o tempo médio gasto por execução do algoritmo DE. Observa-se que quando o número total de processos é menor ou igual ao número de núcleos da máquina obtêm-se uma redução no tempo de processamento. Já quando este número é extrapolado o processo *master* passa a competir por recursos e isto leva a atrasos na comunicação, tendo em vista que o processo *master* é o mais sensível a gargalos. Comparando a implementação serial com a paralela utilizando apenas um *slave* nota-se que não há um aumento significativo no tempo de processamento, indicando que existe apenas um pequeno *overhead* devido a utilização do MPI. Analisando-se o *speedup* pode-se observar que o ganho máximo ocorre quando há 3 processos *slave*. O fato do *speedup* estar a quase uma unidade de distância do valor ideal 3 é devido a existência de trechos de códigos seriais no algoritmo. Ao sobrecarregar os núcleos o *speedup* cai consideravelmente.

A implementação foi desenvolvida com python3.5 e mpi4py. O uso destas ferramentas permitiu uma rápida prototipagem do método e foi capaz de apresentar uma redução significativa de velocidade no processo de predição, possibilitando novos direcionamentos na pesquisa. Trabalhos futuros incluem a paralelização de mais porções do código com o objetivo de melhorar o *speedup*. Pode-se ainda utilizar controle automático de parâmetros e aplicar um modelo distribuído em ilhas como modelo de paralelismo.

Referências

Guyeux, C., Côté, N. M.-L., Bahi, J. M., and Bienia, W. (2014). Is protein folding problem really a np-complete one? first investigations. *Journal of bioinformatics and computational biology*, 12(01):1350017.

Narloch, P. H. and Parpinelli, R. S. (2017). The protein structure prediction problem approached by a cascade differential evolution algorithm using rosetta. In *6th Brazilian Conference on Intelligent Systems*, pages 294–299.

Oliveira, M., Borguesan, B., and Dorn, M. (2017). Sade-spl: A self-adapting differential evolution algorithm with a loop structure pattern library for the psp problem. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 1095–1102. IEEE.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Walsh, G. (2002). *Proteins: biochemistry and biotechnology*. John Wiley & Sons.

Algoritmo de *handover* ciente de Qualidade de Experiência e Qualidade de Serviço em redes veiculares heterogêneas

Iago Medeiros[1], **Lucas Pacheco**[1], **Denis Rosário**[1], **Jéferson Nobre**[2], **Eduardo Cerqueira**[1]

[1] Universidade Federal do Pará (UFPA), Belém, PA – Brasil

[2] Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS – Brasil

{iago.medeiros, lucas.pacheco}@itec.ufpa.br, {denis,cerqueira}@ufpa.br

jcnobre@unisinos.br

Resumo. *Este artigo apresenta um algoritmo de handover ciente de Qualidade de Experiência (QoE) e Qualidade de Serviço (QoS) em redes veiculares de cenário heterogêneo e sua avaliação de desempenho, em que mostra resultados melhores com um menor custo de handover. O algoritmo considera QoE e QoS como métricas de entrada para o Analytic Hierarchy Process (AHP) determinar o handover, melhorando assim o QoE final dos vídeos transmitidos.*

1. Introdução

De acordo com estudos da [Cisco 2016], estima-se que em 2021 o consumo de vídeo ocupará 82% do total de conteúdo trafegado na Internet, já sendo um dos maiores gargalos da rede atualmente. Além disso, os usuários estão mais exigentes, impondo um nível de Qualidade de Experiência (QoE) cada vez mais alta. Quando os dispositivos conectados são os próprios veículos, existem alguns problemas inerentes desse cenário.

Um problema recorrente em redes veiculares é a mobilidade, onde a entrega de vídeo se torna comprometida enquanto o *handover* para o dispositivo (*User Equipment* - UE) precisa ser feito de forma transparente (ou seja, sem que o usuário perceba a troca de torres ou degradação de QoE). Além disso, a heterogeneidade das Estações Rádio-Base (ERBs) também dificulta o processo devido as suas características distintas, tais como potência de transmissão e área de cobertura. Todo *handover* acarreta em uma alta carga de sinalização para as ERBs envolvidas e sua execução deve ser cuidadosamente executada. Portanto, existe uma necessidade de se ter um equilíbrio entre o número de *handover* realizados com o QoE fornecido para obter um bom desempenho.

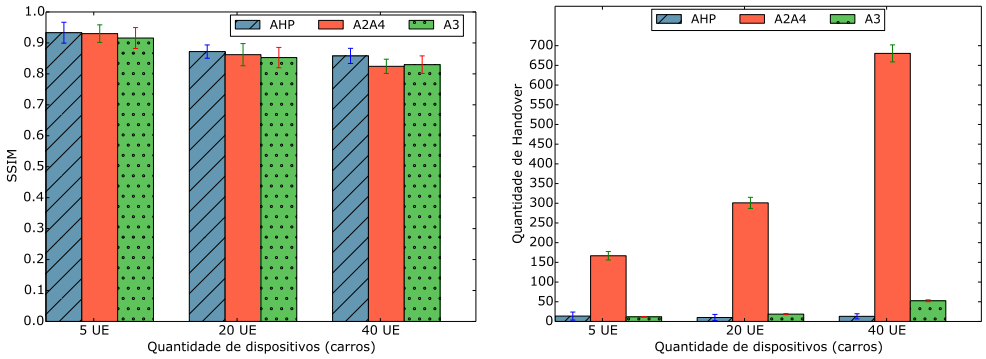
Este artigo apresenta um algoritmo ciente de QoE e Qualidade de Serviço (QoS) para realizar *handover* em redes veiculares heterogêneas. O algoritmo proposto considera o *Analytic Hierarchy Process* (AHP) para escolha da antena que o dispositivo deve se conectar, com base em multi-critérios (potência de sinal, QoE, QoS), devido à sua rapidez. Desta forma, o algoritmo decide se um determinado UE deve ou não realizar o *handover*. Com base em resultados de simulação, a proposta apresentou uma decisão mais eficiente em um cenário pequeno, em termos de diminuição de quantidade de *handover*, mantendo, ou até em alguns casos, melhorando o QoE dos vídeos entregues aos usuários.

2. *Handover* ciente de QoE e QoS eficiente em custos

O algoritmo proposto considera três características das ERBs para tomada de decisão sobre o *handover*: (i) a potência de sinal leva em consideração a qualidade do sinal transmitido pela

antena; (ii) o *Mean Opinion Score* (MOS) preditivo, que é uma forma de medição de QoE na qual atribui-se um resultado numérico para cada vídeo transmitido; e (iii) a Taxa de Entrega de Pacotes (TEP), que é um fator de QoS. Todos estes fatores são critérios de avaliação usados no AHP, que estima o quanto um fator é mais importante que outro, e serve para quantizar as diferentes ERBs disponíveis para *handover*, caso o QoE esteja baixo. Para fomento da técnica do AHP, a ordem de importância atribuída foi obtida experimentalmente [Medeiros et al. 2017] e é possível verificar que MOS é 2 vezes mais importante que potência do sinal, que por sua vez é 2 vezes mais importante que TEP.

Foram realizadas simulações no NS-3.27 em um cenário com 14 antenas LTE e uma quantidade de veículos (UE) variando em 5, 20 ou 40. Os carros locomovem-se no cenário com velocidade variando entre 0 km/h a 160 km/h, onde cada um requisita um vídeo do servidor 10 segundos (s) após o início da simulação até 20s antes do fim. 28 nós fixos consomem um tráfego de dados paralelos em todas as ERBs, para congestioná-las e deixar o cenário mais realista. A composição geográfica das antenas foi de 7 macrocélulas distantes 500 metros entre si, com outras 7 microcélulas próximas. Foram usadas 30 simulações para garantir um intervalo de confiança de 95%. Foram testados um algoritmo puramente por potência de sinal (A2A4), outro com potência mais um *time to trigger* e histerese (A3), além do algoritmo de *handover* proposto. A Figura 1(a) mostra o SSIM (qualidade do vídeo final transmitido) nos 3 cenários distintos onde quanto mais próximo de valor 1, melhor o vídeo entregue. Figura 1(b) mostra a Quantidade de *Handover* nos mesmos 3 cenários, em que é preferível ter um valor bem baixo. O AHP consegue entregar o vídeo com uma qualidade igual (cenário para 5 carros) ou até mesmo superior (cenário com 20 e 40 carros) aos concorrentes realizando menos *handovers*.



(a) SSIM para os 3 algoritmos (b) Quantidade de HO para os 3 algoritmos

Figura 1. Resultados de Simulação

3. Conclusão

Este trabalho apresentou um algoritmo mais eficiente em termos de *handover* e ciente de QoE e QoS. Tal algoritmo permite uma menor carga de sinalização para a rede, e proporciona melhor entrega de conteúdo para os veículos consumidores em um cenário heterogêneo. Como trabalho futuro, espera-se avaliar o algoritmo com outros vídeos mais exigentes.

Referências

Cisco (2016). Cisco Visual Networking Index: Forecast and Methodology, 2015–2020. Technical report, Tech Report.

Medeiros, I., Pacheco, L., Rosário, D., and Cerqueira, E. (2017). Handover em redes heterogêneas baseado em ahp para transmissão de vídeo. *Anais da XVI Escola Regional de Informática Norte 2 (ERIN II 2017)*.

Análise da Influência do Fluxo das Correntes de Ar em Data Center de Pequeno e Médio Porte

Ademir Camillo Junior, Maurício A. Pillon

[1]Programa de Pós-Graduação em Computação Aplicada (LabP2D/PPGCA – DCC)
Universidade do Estado de Santa Catarina (UDESC) – Joinville/SC – Brasil
ademir.junior@edu.udesc.br, mauricio.pillon@udesc.br

Resumo. *Em datacenters de pequeno e médio porte é comum a utilização de sistemas de refrigeração inadequados. Este artigo apresenta uma análise dos fluxos de ar quente e frio, através da arquitetura MonTerDC e Ansys CFD.*

1. Introdução

Em data centers (DC) de pequeno e médio porte, é comum o uso de sistemas de refrigeração que não são adequados para este fim. Apenas 8% dos DC Brasileiros utilizam sistema de refrigeração de acordo com as normas e utilizam o padrão de climatização do tipo CRAC (Computer Room Air Conditioner) [Schneider 2014]. Os demais, por utilizarem sistemas de refrigeração que não permitem o controle do fluxo de ar, nem o direcionamento das correntes de ar frio para determinadas áreas, estão suscetíveis a formação de zonas de calor. Consequentemente, o aumento do consumo de energia com refrigeração e a diminuição da vida útil dos equipamentos [ASHRAE 2016].

Nestes ambientes, alguns dos fatores que influenciam a formação de zonas térmicas indesejáveis são: (i) o posicionamento dos equipamentos e racks, (ii) as correntes de ar frio/quente e (iii) a carga de processamento dos servidores. Assim, dos aspectos relacionados, ações podem ser tomadas para otimizar ou reduzir a ocorrência de zonas térmicas de acordo com a situação real do DC.

2. Arquitetura e Experimentos

Com o objetivo de identificar os fluxos de ar presentes em um DC, foram realizados experimentos utilizando a arquitetura MonTerDC e softwares simuladores (Ansys Fluent) [Camillo et al. 2017]. A arquitetura permite o mapeamento das correntes de ar e a identificação da formação de zonas térmicas indesejáveis, através da coleta da temperatura por sensoriamento e a apresentação dos dados com a simulação em 3D.

Os experimentos foram realizados utilizando blocos de processamento [Camillo et al. 2017] que permitiram simular a carga de trabalho dos servidores em diferentes regiões do DC. Através da coleta das temperaturas pelos sensores e a simulação utilizando CFD (Computational Fluid Dynamics) [Wibrow, 2015] foi possível identificar as zonas térmicas e os fluxos de ar quente/frio. A Figura 1 (a/b) foi gerada a partir dos dados coletados do DC com 100% de carga, em todos os blocos de processamento. Nota-se que, devido a circulação de ar frio, do condicionador de ar (Figura 1 (a)), as áreas A e B recebem refrigeração direta e as áreas C e D indiretamente, através das correntes de retorno. No entanto, a região E e F não recebem a circulação de ar frio através das correntes do fluxo de ar.

Entretanto, não são apenas as correntes de ar frio do condicionador de ar que influenciam as demais regiões do DC. Quando se adiciona cargas de trabalho nos servidores, neste caso 100% em todos, cada equipamento produzirá sua própria corrente de ar quente, influenciando a distribuição de ar frio e consequentemente a criação de zonas de calor (G, H e I), como observado na Figura 1 (b).

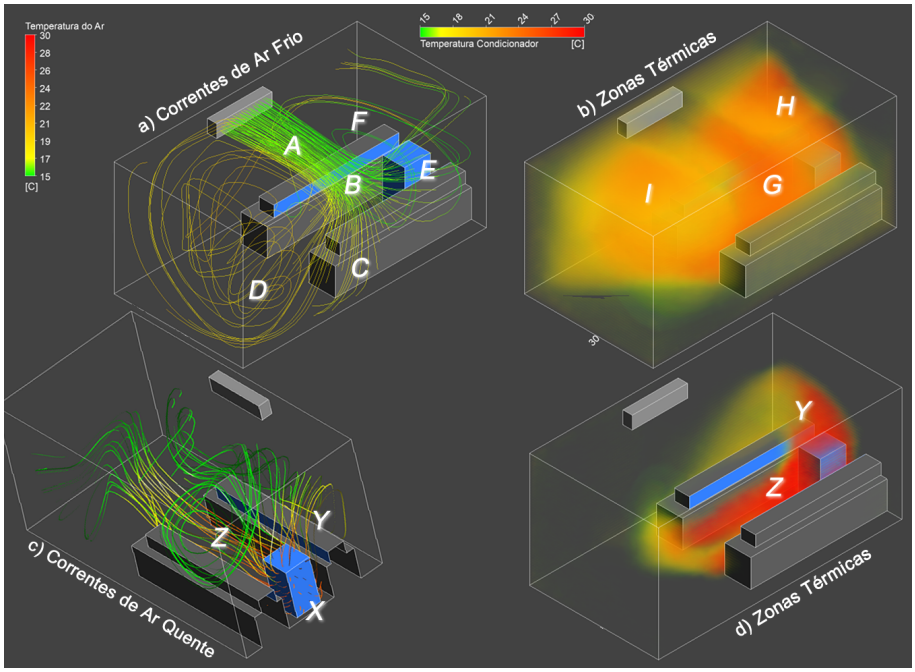


Figura 1. Zonas Térmicas e Correntes de Ar

Também é possível observar, através do sensoriamento em diversas regiões do DC, que ao adicionar cargas de trabalho em locais específicos, outras regiões que recebiam correntes de ar foram diretamente influenciadas, elevando a sua temperatura, mesmo que os servidores do bloco mais próximo estivessem sem carga de trabalho. Esse aumento da temperatura ocorre devido ao fluxo de ar quente dos servidores adjacentes, como pode ser observado na Figura 1 (c). Neste caso, apenas os servidores do rack X estão com 100% de carga de trabalho, gerando as correntes quentes Y e Z em direções diferentes. Devido a isso, zonas térmicas são formadas nos pontos de concentração de calor e falta de refrigeração (F e E), como observado na Figura 1 (d).

3. Considerações Parciais

Os experimentos realizados permitiram a análise da influência do fluxo de ar quente e frio em pequenos e médios DC. Como próximos passos, tem-se a integração da arquitetura com o gerenciador de cloud OpenStack e renderização 3D online.

Referências

ASHRAE (2016). Ashrae tc9.9, data center networking - issues and best practices.

Camillo, A. J., Miers, C. C., Koslovski, G. P., and Pillon, M. A. (2017). *Análise de zonas térmicas em data center não-crac*. WSCAD.

Schneider, E. (2014). Soluções em climatização para data center. Brasília/Brasil. XIV Encontro Nacional de Empresas Projetistas e Consultores da Abrava.

Avaliação de Desempenho da Implementação Baseada em Tarefas e Fluxo de Dados do Método de Lattice-Boltzmann

Gabriel Freytag[1], João V. F. Lima[1], Claudio Schepke[2]

[1]Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

[2]Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete – RS – Brasil

{gfreytag,jvlima}@inf.ufsm.br, schepke@unipampa.edu.br

Resumo. *Várias classes de fluidodinâmica computacional utilizadas em diversas áreas normalmente demandam um grande poder computacional. Neste trabalho apresentamos uma implementação do método de Lattice-Boltzmann baseada em tarefas com dependências OpenMP com speedup de 30.20 em relação à implementação baseada no paralelismo de iterações de laços de repetição.*

1. Introdução

Arquiteturas *multicore* de memória compartilhada são comumente uma das alternativas utilizadas na aceleração do processamento de aplicações que demandam um grande poder computacional, um exemplo disso são aplicações de fluidodinâmica computacional. O método de Lattice-Boltzmann (LBM) é uma aplicação de fluidodinâmica computacional que se destaca dos demais métodos convencionais pela simplicidade e propensão ao paralelismo [Chen and Doolen 1998]. Na literatura a paralelização do método em arquiteturas de memória compartilhada comumente é realizada baseada no paralelismo de iterações de laços de repetição e, dessa forma, o objetivo deste trabalho é avaliar o desempenho da paralelização do LBM utilizando tarefas OpenMP com dependências, introduzidas na versão 4.0, em uma arquitetura *multicore* de memória compartilhada do tipo NUMA em comparação à uma implementação baseada no paralelismo de iterações. Ambas as implementações são extensões da implementação de Schepke e Diverio (2007).

2. Implementações

Na implementação apresentada por Schepke e Diverio (2007) existem seis funções que são executadas por cada nó sobre o seu subdomínio e que aplicam suas operações sobre os dados armazenados em duas matrizes tridimensionais, uma principal e outra temporária. Já na implementação baseada em tarefas cada função foi transformada em uma tarefa por meio da cláusula `task` da API OpenMP. Além disso, definiu-se as dependências de cada tarefa por meio da cláusula `depend` de acordo com as operações de leitura e escrita que a tarefa realiza em ambas as matrizes de um subdomínio. Apesar das dependências das seis tarefas às impedir de serem executadas paralelamente em um mesmo subdomínio, não às impede de serem executadas paralelamente em subdomínio distintos, com exceção de uma tarefa. Como a tarefa de cópia das bordas dos subdomínios depende da leitura da matriz temporária de subdomínios vizinhos e escrita na matriz temporária do subdomínio para o qual foi gerada, essa tarefa somente pode ser executada paralelamente em subdomínios não vizinhos. Para remover essa restrição e permitir que todas sejam executadas paralelamente nos diferentes subdomínios desenvolveu-se uma segunda implementação que utiliza matrizes auxiliares (*buffers*) para a cópia das bordas, onde adicionou-se uma sétima

tarefa que copia as bordas dos vizinhos para os *buffers* que posteriormente são copiadas pela tarefa de cópia das bordas dos *buffers* para a matriz temporária do subdomínio.

3. Resultados e Discussões

Os experimentos foram realizados em uma máquina com 8 nós NUMA, cada nó com um processador Intel Xeon SandyBridge E5-4617 de 6 núcleos de processamento e 64 GB de memória RAM, um total de 48 núcleos e 512 GB de memória. Além disso, os tamanhos dos reticulados utilizados foram 32, 64, 96, 128, 192, 256 (múltiplos de 32). Nas implementações baseadas em tarefas os reticulados foram particionados em 2, 4, 8 e 16 subdomínios por dimensão. Cada experimento foi repetido 10 vezes. Na Figura 1 é apresentado a média dos melhores tempos de execução obtidos em cada reticulado, independentemente do particionamento utilizado (nas implementações baseadas em tarefas), das implementações baseada no paralelismo de iterações de laços de repetição (OpenMP), baseada no paralelismo de tarefas com dependências (OpenMP Tasks) e baseada no paralelismo de tarefas com dependências e com *buffers* (OpenMP Tasks Buffer) com ambos os tamanhos de reticulado utilizando 12, 24, 36 e 48 núcleos (2, 4, 6 e 8 nós NUMA). Como é possível observar, especialmente a partir do tamanho de reticulado 128, o desempenho da implementação baseada em tarefas com *buffers* se sobressai expressivamente em relação às demais. Além disso, com 24, 36 e 48 núcleos a implementação baseada em tarefas obteve os maiores tempos de execução em todos os tamanhos de reticulado.

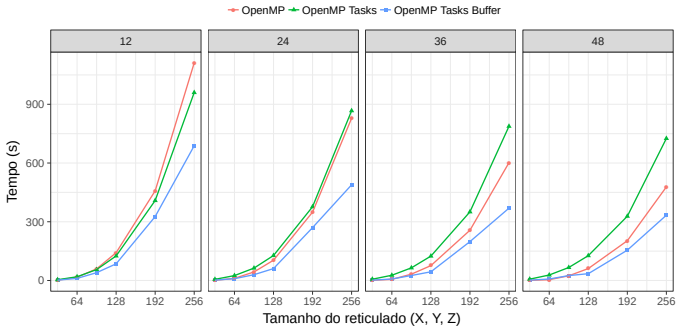


Figura 1. Tempos de execução por tamanho do reticulado em ambas as implementações.

4. Conclusão e Trabalhos Futuros

Como pôde ser observado, a implementação baseada em tarefas com *buffers* obteve os melhores tempos de execução. No reticulado de tamanho 256 e 48 núcleos obteve um *speedup* de 30.20 em relação à implementação baseada no paralelismo de iterações. Como um trabalho futuro sugere-se a análise do desempenho de uma implementação baseada em tarefas OpenMP com dependências para arquiteturas de memória distribuída.

Referências

Chen, S. and Doolen, G. D. (1998). Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364.

Schepke, C. and Diverio, T. A. (2007). Distribuição de dados para implementações paralelas do método de lattice boltzmann. Master's thesis, Universidade Federal do Rio Grande do Sul.

Avaliação de Desempenho de Tecnologias de Comunicação na Plataforma Arduino no Contexto de VANETs

Ricardo Silveira Rodrigues[1], **Marcia Pasin**[1]

[1]Laboratório de Sistemas de Computação – Universidade Federal de Santa Maria (UFSM)
Caixa Postal 15.064 – 91.501-970 – Santa Maria – RS – Brasil

{rrodrigues,pasin}@inf.ufsm.br

Resumo. *Este trabalho apresenta uma avaliação prática de desempenho das tecnologias LoRa e Wi-Fi na Plataforma Arduino para aplicações relacionadas a segurança em VANETs. A proposta deste trabalho é avaliar se a latência usando essas tecnologias não ultrapassa o limite de tempo permitido para esse tipo de aplicação. Os resultados preliminares mostram que o objetivo pode ser atingido em ambas tecnologias com certas restrições.*

1. Introdução

Redes veiculares ou VANETs (*Vehicular Ad Hoc Network*) são redes formadas por veículos e por equipamentos espalhados em estradas, chamados *Road Side Units* (RSU)[Rodrigues 2016], que trocam mensagens entre si, podendo ser *Vehicle-to-Vehicle* (V2V), *Vehicle-to-Infrastructure* (V2I) ou *Infrastructure-to-Infrastructure* (I2I). Em VANETs, veículos são equipados com sensores sem fio que permitem a comunicação com outros veículos e com os arredores.

As aplicações em VANETs são geralmente classificadas como relacionadas a segurança e não relacionadas a segurança. As aplicações relacionadas a segurança são classificadas em três categorias: assistente de motorista (evitar colisões cooperativamente, navegação em estradas e mudança de pista), informações de alerta (zona em obras e alerta de limite de velocidade) e aviso de atenção (obstáculos na estrada e outras condições vitais). As aplicações não relacionadas a segurança são referentes ao conforto dos passageiros e eficiência do tráfego.

Por se tratar de VANETs, os dispositivos que fazem a comunicação estão acoplados a veículos que podem estar andando em altas velocidades. Nesse contexto, a latência na troca de mensagens entre os participantes da rede não pode ser alta, principalmente nas aplicações relacionadas a segurança, onde as vidas dos motoristas envolvidos podem depender da velocidade em que as mensagens são trocadas.

A Plataforma Arduino[1] é uma plataforma de prototipação eletrônica que permite que testes envolvendo sensores eletrônicos possam ser realizados com custo de implementação relativamente baixo. Com a plataforma é possível instalar módulos de comunicação em veículos e realizar testes práticos, tanto em laboratório em veículos robôs quanto em veículos reais.

[1]www.arduino.cc

2. Proposta

[Eze et al. 2014] analisa que aplicações relacionadas a segurança necessitam ter uma latência de no máximo 100 milissegundos (ms) para que possam ser efetivas em avisar os veículos necessários a tempo de evitar acidentes.

Este trabalho tem o objetivo de avaliar a latência de diferentes tecnologias de comunicação usando a plataforma de prototipação Arduino. Inicialmente será medida a latência usando a tecnologia LoRa (módulo Dragino LoRa Shield) e a tecnologia Wi-Fi (módulo ESP8266), e posteriormente outras tecnologias de comunicação que possam ser usadas no contexto de VANETs.

3. Resultados preliminares e trabalhos futuros

Os resultados preliminares realizados mostrou que é possível obter uma latência média de 27 ms usando a tecnologia LoRa, porém a medição foi feita usando uma comunicação direta *node-to-node* entre dois módulos. Espera-se que usando *gateways* para a comunicação, como é definido no protocolo LoRaWAN, a latência aumente consideravelmente. Utilizando o módulo Wi-Fi, foi possível obter uma média de 75 ms de latência em uma conexão TCP/IP, porém a latência máxima por vezes passa de 100 ms. Espera-se conseguir que a latência máxima não passe do limite de 100 ms estabelecido.

Para trabalhos futuros, pretende-se usar a tecnologia de comunicação mais adequada entre as testadas para desenvolver aplicações em VANETs usando a Plataforma Arduino para prototipação.

Referências

Eze, E. C., Zhang, S., and Liu, E. (2014). Vehicular ad hoc networks (vanets): Current state, challenges, potentials and way forward. In *2014 20th International Conference on Automation and Computing*, pages 176–181.

Rodrigues, R. (2016). Um Estudo Sobre Técnicas de Descrição de Percurso de Veículos Usando a Plataforma Arduino. Monografia (Bacharel em Ciência da Computação), UFSM (Universidade Federal de Santa Maria), Santa Maria, Brasil.

Yousefi, S., Mousavi, M. S., and Fathy, M. (2006). Vehicular ad hoc networks (vanets): Challenges and perspectives. In *2006 6th International Conference on ITS Telecommunications*, pages 761–766.

Avaliação do mecanismo de checkpoint no HDFS em um cenário com falha de DataNode

Paulo V. M. Cardoso, Patrícia Pitthan Barcelos

Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

pcardoso@inf.ufsm.br, pitthan@inf.ufsm.br

Resumo. *A técnica de Checkpoint and Recovery apresenta-se de forma eficiente no contexto de tolerância a falhas, atenuando problemas de confiabilidade e disponibilidade em sistemas de alto desempenho. O Apache Hadoop, usado para trabalhar com quantidades massivas de dados, implementa checkpoint estático em seu sistema de arquivos distribuído. Este trabalho apresenta uma validação do checkpoint no Hadoop através da indução de falhas no DataNode.*

1. Introdução

O contexto da computação de alto desempenho exige o uso de um número cada vez maior de componentes em sistemas computacionais. Essa característica influencia a queda de confiabilidade e disponibilidade, já que tais sistemas são mais propensos a falhas. Uma técnica de tolerância a falhas bastante utilizada é o *Checkpoint and Recovery* (CR), que consiste no salvamento do estado do serviço (*checkpoint*) e na recuperação pós falha.

O Apache Hadoop, *framework* usado para processar e armazenar grandes quantidades de dados, usa a técnica de CR. Porém, o *checkpoint* no Hadoop possui atributos de configuração estáticos. Ou seja, o período de salvamento de *checkpoints* não pode ser alterado em tempo de execução, sendo que a escolha do atributo é determinante tanto para o desempenho de aplicações quanto para o nível de confiabilidade do sistema.

O trabalho apresenta uma validação do mecanismo de *checkpoint* estático do Hadoop em situações de falhas no DataNode, elemento responsável pelo armazenamento de dados do HDFS. As falhas são induzidas em períodos específicos de execução, de forma que o DataNode não volte a ser executado (falha de *crash*). Assim, o sistema deve manter sua execução e realocar os dados perdidos para novos nós através dos *checkpoints* salvos. A validação é feita por uma análise de desempenho do tempo de execução do Hadoop.

2. Checkpoint

O *checkpoint* no Hadoop é implementado para tolerar falhas no HDFS [White 2015]. Para isso, o namespace do HDFS é replicado no arquivo FSImage, armazenado em disco no sistema de arquivos local do NameNode. Esse arquivo mantém informações sobre o mapeamento de blocos para arquivos e propriedades do sistema. Para evitar a criação de novo FSImage a cada operação do HDFS, um *log* de edições (EditLog), também mantido em disco local, armazena as últimas transações realizadas após a criação do FSImage.

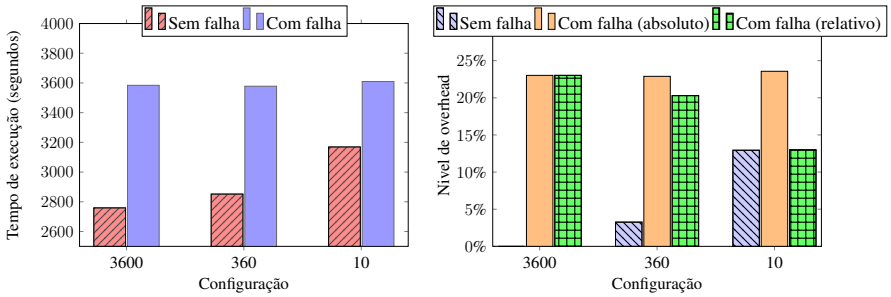
O *merge* entre o FSImage e o EditLog consiste no processo de *checkpoint*. Esse procedimento é realizado quando o NameNode inicia e, posteriormente, é feito de forma

periódica. O intervalo de *checkpoint* padrão do Hadoop é 3600s, mas pode ser disparado se o HDFS atingir um número determinado de transações. O *checkpoint* é realizado pelo SecondaryNameNode (SNN), que mantém uma cópia do FSImage e a atualiza a cada *checkpoint*, solicitando o arquivo de edições ao NameNode.

3. Resultados e Trabalhos Futuros

A experimentação foi realizada a partir do *benchmark* TestDFSIO, disponibilizado pela distribuição do Hadoop. A aplicação foi usada para testar o HDFS a partir de operações de escrita com 20 arquivos de 16GB cada. As execuções foram executadas com 20 amostras em 8 nós da plataforma Grid5000 [Grid'5000 2017].

A falha de *crash* é induzida no DataNode do nó executor no tempo de 20% da média *baseline* (3600 segundos de *checkpoint*) com o comando *kill* do Linux, emulando-se falha de *crash*. Utilizou-se o fator de replicação 3 (padrão). A Figura 1(a) mostra os resultados com variação do período de *checkpoint*. O tempo médio de execução é mais alto com frequências maiores. Com falhas, porém, a sobrecarga tem um comportamento linear. *Checkpoints* mais atualizados tendem a economizar o tempo de atualização do *namespace* com EditLogs menores, já que a recuperação de réplicas perdidas requisita uma quantidade considerável de operações no NN, a fim de manter o fator de replicação.



(a) Tempo de execução (b) Análise de sobrecarga.

Figura 1. *Overhead* da variação de *checkpoint* no TestDFSIO.

A Figura 1(b) mostra o *overhead* observado em um cenário sem falhas, relacionado ao *baseline* (absoluto) e ao resultado de cada variação no teste sem falha (relativo). O impacto do período de *checkpoint* não interfere de forma significativa no *overhead* absoluto da aplicação, porém a observação relativa revela que uma periodicidade maior indica uma recuperação mais eficiente. Percebe-se que a exigência por um nível de confiabilidade mais alto, de fato, auxilia no processo de recuperação. Por outro lado, essa configuração sofre com sobrecargas em cenários sem falhas.

Trabalhos futuros investigarão o comportamento das aplicações em cenários de falha transiente no NameNode, já que o *checkpoint* é um elemento essencial de recuperação do nó mestre do HDFS. A partir dessas análises, o uso do *checkpoint* dinâmico proposto será aplicado de forma a verificar sua usabilidade no contexto de falhas do Hadoop.

Referências

Grid'5000 (2017). *Grid5000 Homepage*. http://www.grid5000.fr/, (acessado em dezembro de 2017).

White, T. (2015). *Hadoop: The Definitive Guide, 4th Edition*. "O'Reilly Media, Inc.".

Avaliação preliminar dos tempos de requisições de memória sobre o conjunto de *benchmarks* MediaBench

Giovane de Oliveira Torres[1]*, Rodrigo Costa de Moura[1], Laércio Lima Pilla[2], Maurício Lima Pilla[1]

[1]Centro de Desenvolvimento Tecnológico – UFPel
Pelotas – RS – Brasil

[2]Departamento de Informática e Estatística – UFSC
Florianópolis – SC – Brasil

{gdotorres,rcmoura,pilla}@inf.ufpel.br, laercio.pilla@ufsc.br

Resumo. *Com as necessidades por melhorias em memórias, é importante avaliar alternativas para o futuro, como o uso de memórias não voláteis. Este artigo faz um profiling do tempo gasto em requisições de memória do MediaBench, com a finalidade que futuramente seja implementado um algoritmo diferente para controladora de memória e avaliado o seu impacto nas operações de leitura e escrita.*

1. Introdução

Existem diversas demandas nas questões de desempenho e consumo energético – em especial, as memórias usadas hoje podem estar atingindo seu limite, o que permite explorar alternativas como o uso de memórias não voláteis (MNVs), as quais apresentam características interessantes como baixo consumo energético e maior densidade. Porém, estas memórias tem problemas que incluem custo alto em operações de escrita e baixa durabilidade do material [Meena et al. 2014].

Assim, é importante buscar meios para superar os problemas nas MNVs para que sejam usadas como memórias principais. Em alguns sistemas, o gerenciamento de tempo das requisições de memória (RMs) é crucial – com isso, o uso de MNVs potencializa este problema com o tempo. Este trabalho visa estudar o tempo gasto em operações de memória por algumas aplicações, i.e., fazer um *profiling* de *benchmarks* para que futuramente, com a implementação de um algoritmo diferente para tratamento de RMs (o qual levará em conta a assimetria entre as operações em MNVs) verifique-se o impacto deste nos tempos de operações de memória.

2. Metodologia e Resultados

Para a realização deste trabalho foram escolhidas duas ferramentas de simulação, NVMain [Poremba e Xie 2012] e Gem5 [Binkert et al. 2011], já que o uso de ambas em conjunto permite uma simulação mais próxima do tempo real. Para a extração de resultados, foram executados as aplicações do conjunto de *benchmarks* MediaBench [Lee et al. 1997], à exceção de `pgp` e `pegwit`, devido ao código-fonte não fornecer suporte natural a uma arquitetura de 64 *bits*. Os *benchmarks* foram submetidos a 4 testes considerando tecnologias diferentes para memória principal. Em um destes foi usado a

*Bolsista CNPq

DRAM, servindo como base de tecnologia atual. Para os testes restantes, escolheram-se 3 tecnologias diferentes de MNVs para simulação, STT-RAM (*Spin Transfer Torque RAM*), PCRAM (*Phase Change RAM*) e RRAM (*Resistive RAM*), por serem consideradas como prováveis tecnologias para substituir as utilizadas atualmente [Meena et al. 2014].

Como resultados, as simulações iniciais mostraram que a quantidade de RMs de leitura atende por 93% do total de requisições geradas em média (os 7% restantes são escritas), independente da tecnologia de memória simulada. Na DRAM, o total de tempo dispendido em geração e atendimento das RMs é pouco superior a 23% em média nos *benchmarks* simulados, sendo cerca de 22% para leituras e 1% para escritas, sendo explicado pela grande diferença no número de RMs geradas para os diferentes tipos de operações, já que o tempo dispendido em uma operação tanto de leitura quanto de escrita é similar.

Já nas simulações envolvendo MNVs, a diferença está no tempo gasto com operações de escrita, o qual aumenta. Em STT-RAMs, as requisições de escrita atendem por 2,5% do tempo total de simulação, enquanto que em RRAMs e PCRAMs este tempo fica em 3% e 6%, respectivamente – o acréscimo é decorrente das velocidades de escrita serem superiores às de leitura nas MNVs.

3. Conclusões

Este artigo fez um estudo preliminar sobre os tempos gastos com RMs nas aplicações do MediaBench, para que futuramente com a implementação de um algoritmo diferente para controladora de memória seja verificado seu impacto sobre as RMs.

Verificou-se que foram geradas quantidades de RMs de leitura (93% do total de requisições para 7% de escritas). Ao utilizar DRAM como memória principal, o tempo dispendido com requisições de leitura e escrita corresponde respectivamente a 22% e 1% do total do tempo gasto na simulação. Nas execuções envolvendo MNVs, o tempo dispendido nas RMs de escrita torna-se maior devido à característica de assimetria entre as operações.

Referências

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., e Wood, D. A. (2011). The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7.

Lee, C., Potkonjak, M., e Mangione-Smith, W. H. (1997). Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 30, pages 330–335, Washington, DC, USA. IEEE Computer Society.

Meena, J. S., Sze, S. M., Chand, U., e Tseng, T.-Y. (2014). Overview of emerging nonvolatile memory technologies. *Nanoscale research letters*, 9(1):1.

Poremba, M. e Xie, Y. (2012). Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397. IEEE, IEEE.

Combinando Elasticidade Reativa e Proativa para Aumentar o Desempenho de Aplicações HPC

Vinicius F. Rodrigues[1], **Rodrigo da R. Righi**[1]

[1]Programa de Pós-Graduação em Computação Aplicada – UNISINOS

{vfrodrigues,rrrighi}@unisinos.br

Resumo. *A elasticidade de recursos em ambientes de Computação em Nuvem é uma característica explorada por aplicações que demandam alto desempenho e gerenciamento de recursos sob demanda. Porém, a definição dos parâmetros de configuração dessa funcionalidade dependem diretamente do conhecimento e experiência do usuário. Neste contexto, este artigo busca apresentar uma análise detalhada de desempenho da técnica de elasticidade híbrida Live Thresholding em que não há necessidade de configuração de parâmetros.*

1. Introdução

A elasticidade de recursos em ambientes de Computação em Nuvem oferece inúmeros benefícios para aplicações que demandam variável poder computacional ao longo de sua execução [Al-Dhuraibi et al. 2017]. Porém, estratégias de elasticidade geralmente necessitam a correta configuração de parâmetros, o que não é uma tarefa trivial. A maioria das soluções emprega o controle de elasticidade de forma reativa ou proativa/preditiva [Farokhi et al. 2015, Nikravesh et al. 2015]. No presente trabalho, é proposta a técnica de elasticidade Live Thresholding (LT), a qual combina os dois métodos. LT calcula automaticamente os limites inferior (T_i) e superior (T_s) inicializando seus valores em 0% e 100% respectivamente. A carga de processamento (CPS) do ambiente, a qual é calculada periodicamente, baseia-se no consumo de CPU dos recursos e é utilizada para definição dos limites e disparo de ações de elasticidade. Em cada período de monitoramento, é calculada a variação da carga atual ($CPS(o)$) para carga do período anterior ($CPS(o-1)$), atribuindo esse valor para ΔCPS ($\Delta CPS = CPS(o) - CPS(o-1)$). Os limites são reajustados utilizando esse valor de variação em que, se negativo, seu módulo é adicionado ao limite inferior. Caso contrário o módulo da variação é subtraído do limite superior. Em cada período, após a reconfiguração dos limites, ações de elasticidade são disparadas se CPS violar um dos limites. Para realização de experimentos, foram investigadas 6 abordagens diferentes A_z ($A_z|z \in \{a, b, c, d, e, f\}$) para tratar a adaptatividade dos limites após uma ação de elasticidade em um ambiente de Nuvem privada. Quando o limite inferior é violado o valor desse limite pode ser reconfigurado usando uma das estratégias apresentadas na Equação 1. Para a reconfiguração do limite superior, as estratégias investigadas são apresentadas pela Equação 2.

$$\mathrm{T}_i = \begin{cases} 0 & para\ A_a \\ \frac{CPS(o)}{2} & para\ A_b \\ CPS(o-1) - \left|\frac{CPS(o-1)-CPS(o)}{2}\right| & para\ A_c \end{cases} \quad (1)$$

$$\mathrm{T}_s = \begin{cases} 1 & para\ A_d \\ CPS(o) + \frac{1-CPS(o)}{2} & para\ A_e \\ CPS(o-1) + \left|\frac{CPS(o-1)-CPS(o)}{2}\right| & para\ A_f \end{cases} \quad (2)$$

2. Resultados Preliminares

A Figura 1 apresenta a execução de uma aplicação paralela com todas as combinações de LT. As figuras (a), (b) e (c) apresentam comportamento similar e o uso da estratégia A_a.

A variação da estratégia de reconfiguração do limite superior causou variações somente em momentos em que recursos foram adicionados. Nos cenários das figuras (d), (e) e (f) o número de recursos disponíveis foi diferente para cada um. A maior diferença ocorreu no cenário (e) em que a quantidade disponível de recursos chegou a 8 VMs quando a carga já estava diminuindo próximo aos 1000 segundos. Isso ocorreu pois uma nova ação de elasticidade já havia sido iniciada e os recursos ficaram disponíveis somente após esse ponto. Da mesma forma, os cenários (g), (h) e (i) empregam A_c com diferentes estratégias para calcular o limite superior. Enquanto em (g) e (i) duas ações de elasticidade removeram recursos no primeiro pico de queda da carga, em (h) nenhum recurso foi removido nesse pico. Isso ocorreu pois quando a carga começou a diminuir uma ação de elasticidade já estava sendo realizada. Como duas ações de elasticidade não são disparadas concorretemente, nenhuma operação para remoção de recursos foi permitida naquele intervalo.

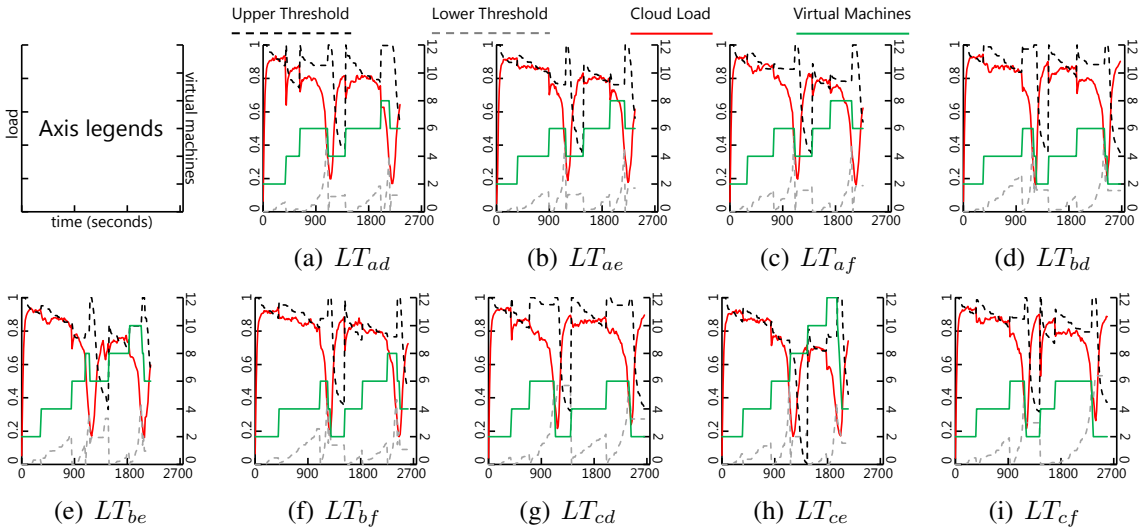


Figura 1. Comportamento da elasticidade para uma aplicação de carga variável.

3. Conclusão

Este artigo apresentou diferentes modelos de configuração de limites de elasticidade explorando a combinação de técnicas reativas e proativas. Os limites são adaptados automaticamente a cada atividade periódica de monitoramento, bem como quando uma ação de elasticidade ocorre. Resultados com uma aplicação que apresenta uma carga variável demonstraram que os melhores resultados foram obtidos pela estratégia LT_{ce}.

Referências

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017). Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, PP(99):1–1.

Farokhi, S., Jamshidi, P., Brandic, I., and Elmroth, E. (2015). Self-adaptation challenges for cloud-based applications : A control theoretic perspective. In *10th International Workshop on Feedback Computing (Feedback Computing 2015)*. ACM.

Nikravesh, A. Y., Ajila, S. A., and Lung, C.-H. (2015). Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 35–45, Piscataway, NJ, USA. IEEE Press.

Controle da Qualidade de Serviço para o Sistema Moodle em Redes Definidas por *Software*

Anderson H. S. Marcondes[1,2], Guilherme Piegas Koslovski[2]

[1]Instituto Federal Catarinense (IFC) – São Francisco do Sul – SC – Brasil

[2]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

anderson.marcondes@ifc.edu.br, guilherme.koslovski@udesc.br

Resumo. *O paradigma das Redes Definidas por Software (SDN) permite que o administrador programe a infraestrutura de rede para aumentar o desempenho de uma aplicação específica, oferecendo uma capacidade superior ao observado em uma arquitetura de rede tradicional. Este trabalho tem por objetivo apresentar os resultados obtidos na priorização do tráfego do sistema Moodle através da ferramenta SDN4Moodle, implementada em uma SDN.*

1. Introdução

Em uma arquitetura de rede tradicional, baseada na pilha de protocolos TCP/IP, o desenvolvimento de políticas para gerenciamento de tráfego é limitado pela rigidez no gerenciamento dos recursos de comunicação [Handley 2006]. Usualmente, o desempenho de uma aplicação distribuída é relacionado com o volume de dados trafegados, sendo limitado pela justiça de compartilhamento buscada pelo protocolo TCP [Cerf and Icahn 2005] [Dukkipati et al. 2011]. O paradigma das Redes Definidas por *Software* (SDN – *Software-Defined Networking*) propõe a separação do plano de dados do plano de controle da rede [McKeown et al. 2008]. A motivação para essa abordagem é baseada na diferenciação e priorização de tráfego, ou seja, o administrador consegue disponibilizar fatias isoladas da rede respeitando os requisitos das aplicações.

Em ambientes acadêmicos, a utilização e o desempenho de sistemas de gestão de aprendizagem (LMS – *Learning Management System*), tal como o Moodle, é diretamente afetado por tráfegos concorrentes. Com a proposta apresentada em [Marcondes and Koslovski 2017], o objetivo deste trabalho é apresentar os resultados do desenvolvimento de uma aplicação SDN, denominada SDN4Moodle, para encaminhamento prioritário de fluxos relacionados com as atividades dos usuários do Moodle.

2. Resultados

A sequência dos passos da navegação do usuário no Moodle e da coleta dos dados do tráfego da rede são apresentadas da seguinte forma: *login* do usuário (C), execução de atividade de texto (D) e envio das respostas para o servidor Moodle(E), execução de *streaming* de áudio (intervalo F-G), execução de vídeo de baixa (H-I), média (J-L) e alta resolução (L-M). Por fim, é executado o *logoff* do usuário.

A Figura 1 apresenta os resultados alcançados pela ferramenta SDN4Moodle, composta por uma topologia SDN em árvore com um e dois usuários do Moodle, com tráfego secundário cruzado gerado pelo *software iperf*. Apesar do cenário 1 mostrar que a

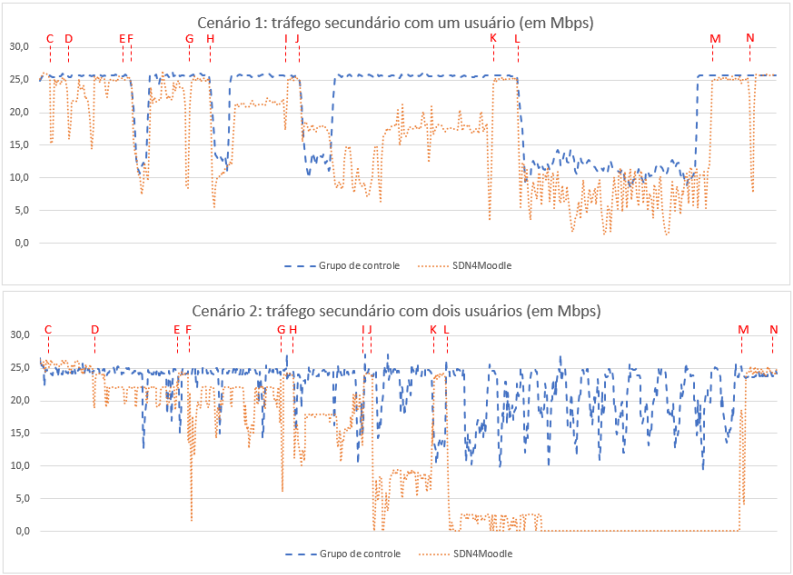


Figura 1. Comparativos da largura de banda do tráfego secundário do *iperf* com um e dois usuários com e sem o controle da SDN4Moodle (vazão em Mbps).

largura de banda consumida pelo tráfego secundário é reduzida quando ocorrem os eventos do Moodle, ao utilizar o controle efetuado SDN4Moodle, é com a concorrência de mais de um usuário do Moodle (cenário 2) que o resultado é realçado. Na comparação da largura de banda do tráfego secundário com e sem o uso do controle proposto neste trabalho, é possível perceber que à medida que os usuários executam as atividades, maior é a queda da vazão do tráfego secundário.

3. Considerações Finais

É recorrente o uso de SDN para otimização de aplicações comunicantes. Assim, este trabalho apresenta os resultados da priorização de fluxos relacionados com o LMS Moodle. A solução implementada, denominada SDN4Moodle, identifica as atividades efetuadas pelos usuários do Moodle, reconfigurando o encaminhamento de fluxos de acordo com a qualidade de serviço necessária para acessar os recursos (áudios, vídeos, textos interativos, entre outros). Para tal finalidade, o controle de métricas padronizado pelo protocolo OpenFlow é utilizado. A análise experimental indicou a diminuição na interferência do tráfego primário do Moodle, bem como uma potencial utilização em cenários acadêmicos.

Referências

Cerf, V. G. and Icahn, R. E. (2005). A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, 35(2):71–82.

Dukkipati, N., Mathis, M., Cheng, Y., and Ghobadi, M. (2011). Proportional rate reduction for tcp. In *Proc. of the 11th ACM SIGCOMM Conf. on Internet Measurement*.

Handley, M. (2006). Why the Internet only just works. *BT Technology Journal*, 24:119–129.

Marcondes, A. and Koslovski, G. P. (2017). Encaminhamento de Fluxos em Redes Definidas por Software: Estudo com um Sistema de Gestão de Aprendizagem. In *Escola Regional de Alto Desempenho (ERAD/RS 2017)*, Ijuí, RS, BR. Sociedade Brasileira de Computação (SBC).

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Desempenho de Rede na Nuvem Pública

Eduardo Roloff, Luciano Paschoal Gaspary, Philippe O. A. Navaux

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{eroloff,paschoal,navaux}@inf.ufrgs.br

Resumo. *A computação em nuvem oferece acesso à uma variada gama de opções computacionais, com diversas opções de instâncias com tamanhos, configurações e preços diferentes. Este trabalho busca comparar o desempenho de rede de instâncias na nuvem contra máquinas tradicionais.*

1. Introdução e Motivação

A Computação em Nuvem [Mell et al. 2011] trouxe um novo paradigma para o meio acadêmico e corporativo. As principais mudanças que o paradigma trouxe são a possibilidade de utilização de recursos sem a necessidade de aquisição dos mesmos (*pay-per-use*) e o acesso a recursos de maneira elástica, onde o usuário pode configurar uma quantidade, virtualmente, infinita de recursos de acordo com sua necessidade. No que se refere a Processamento de Alto-Desempenho (PAD) a Computação em Nuvem tem o potencial de substituir a necessidade de aquisição de *clusters* para execução de aplicações científicas. Um usuário de PAD pode ter acesso a um *cluster* com centenas ou milhares de *cores*, de maneira instantânea, sem qualquer necessidade de aquisição e configuração de equipamentos. A avaliação de desempenho em ambientes virtuais já apresentou diversos resultados e se mostra bastante promissora. No entanto, pouco se estudou sobre como se pode aproveitar as características de Computação em Nuvem para PAD.

2. Resultados

Um dos fatores mais importantes em ambientes de *cluster* é a rede, dessa maneira esse trabalho tem seu foco na avaliação de rede em diversos ambientes. Para os testes de redes, foi usado a suíte de testes *Intel MPI Benchmarks* [1]. Dentre os diversos testes existentes na suíte, selecionamos o teste de PingPong que basicamente é a troca de mensagens entre dois nós. Como ambientes de testes, foram testadas algumas instâncias dos maiores provedores públicos de nuvem: Amazon e Microsoft. Da Amazon foram escolhidas as instâncias C3.8X e X1.32X, com 32 e 128 *cores* respectivamente. Da Microsoft foram escolhidas as instâncias A4, A9, A11, D5, G4 e G5 usando de 8 à 32 *cores*. Para fins de comparação foram usados dois clusters tradicionais, o Cluster 1 é o cluster econome do GRID5000 e o Cluster 2 é o Cluster Draco do GPPD.

A Figura 1 mostra os resultados de largura de banda e a Figura 2 apresenta os resultados de latência, ambos gráficos estão em escala logarítmica. O Cluster-1 e as instâncias A9, possuem tecnologia de rede *Infiniband*, o Cluster-2 possui rede no padrão Gigabit Ethernet, as demais instâncias não possuem indicação de rede. Tanto para largura de banda, como para latência, podemos perceber claramente três grupos: as máquinas

[1]https://software.intel.com/en-us/articles/intel-mpi-benchmarks

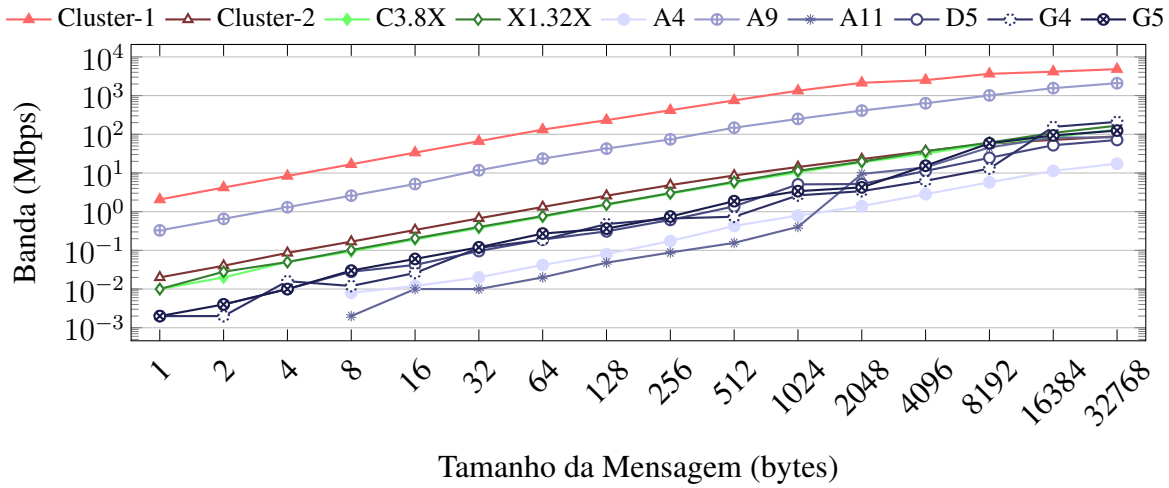


Figura 1. Resultados de largura de banda para o teste de PingPong.

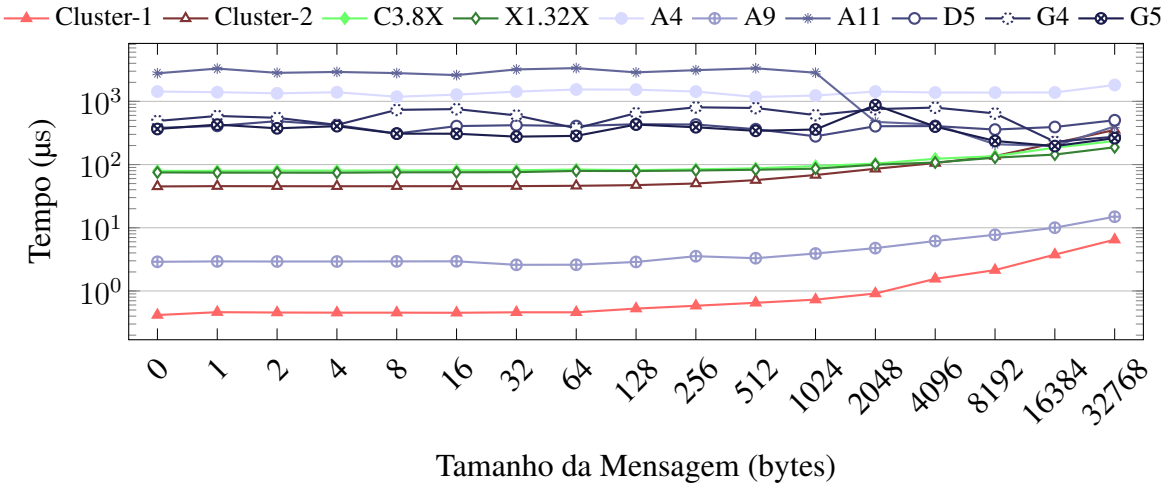


Figura 2. Resultados de latência para o teste de PingPong.

com *Infiniband*, o Cluster-2 e as instâncias da Amazon e o restante das instâncias da Microsoft. Mesmo a instância da nuvem com melhor configuração de rede, instância A9, teve uma sensível perda de desempenho perante uma máquina tradicional, indicando que a virtualização ainda representa um grande sobrecusto.

3. Conclusões e Trabalhos Futuros

Com os resultados de nossos experimentos, podemos perceber que a rede ainda é um grande gargalo da nuvem em relação à ambientes tradicionais. Mesmo instâncias de nuvem equipadas com *infiniband* apresentaram degradação em relação ao *cluster* tradicional dotado da mesma rede. Como trabalhos futuros, iremos continuar a caracterização de instâncias de nuvem em busca de elaborar um modelo de escolha da instância que melhor se adapte a uma aplicação do usuário.

Referências

[Mell et al. 2011] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

Desempenho do Hadoop MapReduce sobre um *Data Center* com Virtualização do Controle de Congestionamento

Vilson Moro[1], **Guilherme Piegas Koslovski**[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

vilson.moro@edu.udesc.br, guilherme.koslovski@udesc.br

Resumo. *Algoritmos legados de controle de congestionamento comprometem o desempenho das aplicações executadas em MVs. Criar uma camada de virtualização para traduzir esses algoritmos para uma versão otimizada é a proposta para contornar tal limitação. O presente trabalho analisa a execução do Hadoop MapReduce em ambientes legados, otimizados pela virtualização do controle de congestionamento.*

1. Introdução

Os *data centers* de nuvens IaaS hospedam aplicações de múltiplos inquilinos em máquinas virtuais (MVs). A complexidade de configuração das aplicações, aliada com as dependências de versões de bibliotecas e sistemas operacionais, constituem fatores limitantes para a atualização da MV. Em MVs legadas a versão do algoritmo de controle de congestionamento do TCP está aquém dos últimos avanços, não interpretando as marcações do núcleo da rede (ECN, *Explicit Congestion Notification*). Com ECN, os *switches* antecipam e informam a possível ocorrência de congestionamento, evitando a perda de pacotes [Alizadeh et al. 2010]. É fato que este ambiente TCP heterogêneo (legados e atualizados) compromete o desempenho das aplicações [Cronkite-Ratcliff et al. 2016].

A virtualização do controle de congestionamento (VCC) implementa uma camada de tradução na qual a conexão iniciada com um algoritmo legado é traduzida para o algoritmo atualizado do *data center* [Alizadeh et al. 2010]. Para analisar a VCC, o presente trabalho discute o desempenho de Hadoop MapReduce em execução sobre MVs legadas.

2. Virtualização do Controle de Congestionamento

A abstração oferecida pelos hipervisores permite que o *data center* utilize um único algoritmo para controle de congestionamento baseado em ECN. A Fig. 1 apresenta o cenário do VCC. O TCP legado envia um pacote solicitando a conexão (1) e o hipervisor acrescenta o bit *ECE* (2). O destinatário confirma a solicitação (3). Novamente, o hipervisor intercepta o pacote e notifica o remetente (4), que inicia o envio dos dados (5). O hipervisor acrescenta o bit *ECT* informando que esse fluxo é capaz de transportar informação de congestionamento (6). O pacote é reconhecido (7) e transferido para o remetente (8). Ocorrendo congestionamento, o bit *ECE* é ativado (9). O emissor reduz o envio de dados através do estrangulamento da janela de recepção (10). Por fim, o remetente continua o envio de dados (11). O destinatário recebe os dados e o tamanho da janela é ajustado (12).

3. Resultados Preliminares e Trabalhos Futuros

Para realizar os testes foi utilizada a ferramenta MRemu [Neves et al. 2015] que emula a comunicação do Hadoop MapReduce. O *data center* é compartilhado com tráfego TCP

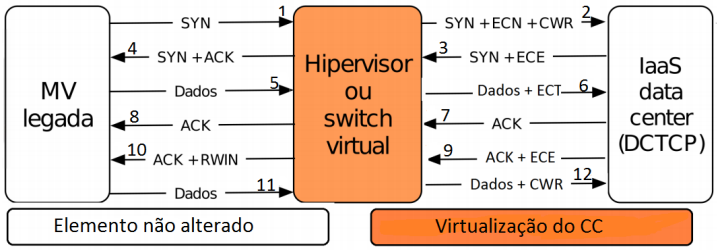
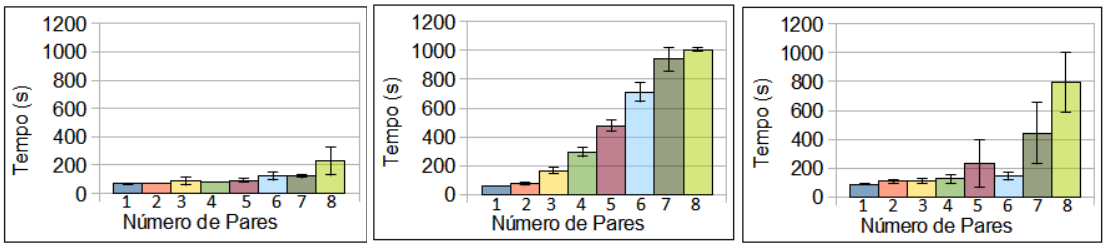


Figura 1. Virtualização do controle de congestionamento em provedores IaaS.



(a) Com ECN. (b) Sem ECN. (c) VCC.

Figura 2. Tempo de execução do Hadoop MapReduce.

controlado (ferramenta *iperf*). A topologia é composta por 2 *switches* interconectados por um enlace de 1 Gbps. Cada *switch* conecta 8 servidores (1 Gbps). O experimento foi realizado em uma MV (2 vCPUs e 8 GB RAM). Os recursos de processamento não representaram gargalos pois MRemu emula a comunicação, sem efetuar o processamento. O tráfego *iperf* foi sempre hospedado em MVs atualizadas. Por sua vez, o Hadoop MapReduce foi executado sobre: *(i)* um sistema atualizado (com ECN), Fig. 2(a); *(ii)* um sistema legado (sem ECN), Fig. 2(b); *(iii)* um sistema legado com VCC, Fig. 2(c). Os gráficos apresentam a média e o desvio padrão do tempo de execução da aplicação (10 execuções e um intervalo de confiança de 95%). O eixo *X* representa o número de pares *iperf* competindo pelos recursos de comunicação, sobretudo pelo enlace entre os *switches*. Em resumo, pode-se constatar um aumento do tempo quando o ambiente apresenta hospedeiros heterogêneos. Ao aplicar VCC, o impacto é diminuído, aproximando os valores do cenário com ECN. Em trabalhos futuros, a formação de fila nos *switches* será analisada, bem como outras aplicações comumente executadas em nuvens serão discutidas.

Agradecimentos. Os autores agradecem ao LabP2D, UDESC e FAPESC.

Referências

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Datacenter TCP (DCTCP). *SIGCOMM Com. Rev.*, 41(4).

Cronkite-Ratcliff, B., Bergman, A., Vargaftik, S., Ravi, M., McKeown, N., Abraham, I., and Keslassy, I. (2016). Virtualized congestion control. In *Proc. of the SIGCOMM Conference*, pages 230–243, New York, NY, USA. ACM.

Neves, M. V., Rose, C. A. F. D., and Katrinis, K. (2015). Mremu: An emulation-based framework for datacenter network experimentation using realistic mapreduce traffic. In *MASCOTS*, pages 174–177. IEEE.

Em Direção a Soluções Distribuídas para Balanceamento de Carga Ciente de Comunicação

Vinicius M. C. T. de Freitas[1], **Laércio L. Pilla**[1], **Márcio Castro**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brazil

vinicius.mct.freitas@gmail.com, {laercio.pilla,marcio.castro}@ufsc.br

Resumo. *A distribuição homogênea de carga nos grandes sistemas paralelos atuais é um desafio ligado às aplicações de comportamento imprevisível. O reescalonamento global de tarefas é uma possível solução para este problema, contudo diversos fatores podem ser levados em conta ao realizá-lo. Este trabalho se propõe a explorar tanto o paralelismo das plataformas, quanto carga de tarefas e a comunicação entre elas.*

1. Introdução

A crescente complexidade de simulações computacionais cria uma demanda por soluções paralelas cada vez mais eficientes. Para atingir uma escalabilidade forte dessas aplicações, uma série de fatores deve ser levada em conta. Entre eles, o balanceamento de carga das aplicações paralelas é crucial para garantir a eficiência do sistema. Algumas simulações computacionais possuem carga de trabalho variável e imprevisível [Menon et al. 2013], que tornam mapeamentos estáticos pouco eficientes.

Conforme o tamanho das plataformas computacionais cresce, o desempenho de soluções de escalonamento global centralizadas diminui, uma vez que estas não exploram o paralelismo das plataformas. Estratégias hierárquicas e distribuídas surgem como uma forma de mitigar esse problema. Utilizando informação local e buscando se aproveitar do paralelismo do sistema, essas abordagens tornam-se mais eficientes em larga escala [Menon et al. 2013].

Uma distribuição homogênea da carga de trabalho é o principal objetivo das atuais soluções do estado da arte. Quando a topologia e custos de comunicação são levados em conta, simulações com uma alta taxa de comunicação podem ter o seu desempenho melhorado. Aplicações científicas como simulações de propagação de onda, dinâmica molecular e refinamento adaptativo de malha podem se beneficiar de um balanceamento que não interfira muito em seus custos de comunicação.

Existem diferentes formas de se considerar o problema da comunicação de tarefas. Algumas estratégias de escalonamento utilizam bibliotecas de particionamento de grafos [Bhatele et al. 2011], enquanto outras calculam os benefícios de migrações [Pilla et al. 2012]. Contudo, dentre estas estratégias do estado da arte, nenhuma é capaz de tanto aproveitar o paralelismo do sistema quanto contabilizar diferentes níveis de comunicação.

2. Proposta

Propomos um balanceador paralelo capaz de ponderar carga e comunicação na escolha de carga de trabalho para migração. Essa estratégia deverá determinar quais os melhores

candidatos para migração e entidades de processamento a recebê-las. Novas heurísticas devem ser desenvolvidas para realizar essas tomadas de decisão, baseadas nas métricas apresentadas a seguir.

Nos maiores supercomputadores do mundo, a comunicação deve ser levada em conta através da topologia de rede dessas máquinas. Topologias de árvore e de malha são diferentes entre si, mas ambas podem se beneficiar do cálculo da média de *hops* por byte [Bhatele et al. 2011] para medir custos de comunicação.

Além de fatores simples como $hops/byte$ e avaliação de qualidade da migração, fatores mais complexos podem ser aliados a estes. Informações específicas de cada ambiente como largura de banda e *jitter* da rede podem ser usados para avaliar custos de comunicação e migração; assim como informações individuais da aplicação (e.g., número de mensagens enviadas entre nós).

O desenvolvimento dessa nova estratégia de escalonamento global será realizado num sistema paralelo com alto grau de escalabilidade. Isso viabiliza comparações com o estado da arte e uso nativo em *benchmarks* para avaliação de desempenho. O `Charm++` [Acun et al. 2016] apresenta o potencial de escalabilidade desejado, além de prover uma série de balanceadores do estado da arte e aplicações para compará-los.

3. Conclusão

A localidade dos dados e processos em uma aplicação paralela é crucial para garantir desempenho, evitando gargalos de execução atrelados à troca de mensagens em arquiteturas multi-computadas [Hoefler et al. 2014]. Sendo assim, um mapeamento de tarefas só é realmente eficiente se garante uma boa distribuição das cargas de trabalho sem aumentar demais os custos de comunicação.

Apesar do avanço no estado da arte, o contínuo crescimento das plataformas computacionais ainda pode se beneficiar de soluções distribuídas sensíveis aos padrões de comunicação das aplicações e às topologias das plataformas. Nossa nova proposta de escalonamento distribuído ciente de comunicação está inserida no contexto de uma dissertação de mestrado com início em Março de 2018.

Referências

Acun, B. et al. (2016). Power, reliability, and performance: One system to rule them all. *Computer*, 49(10):30–37.

Bhatele, A. et al. (2011). Heuristic-based techniques for mapping irregular communication graphs to mesh topologies. In *2011 IEEE International Conference on High Performance Computing and Communications*, pages 765–771.

Hoefler, T. et al. (2014). *An Overview of Topology Mapping Algorithms and Techniques in High-Performance Computing*, pages 73–94. John Wiley Sons, Inc.

Menon, H. et al. (2013). A distributed dynamic load balancer for iterative applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 15:1–15:11, New York, NY, USA. ACM.

Pilla, L. L. et al. (2012). A hierarchical approach for load balancing on parallel multi-core systems. In *International Conference on Parallel Processing*, pages 118–127. IEEE.

Exploração de paralelismo na etapa de legalização de circuitos digitais através do uso de estruturas de dados geométricas

Sheiny Fabre[1], **Laércio Pilla**[1], **José Luís Güntzel**[1]

[1]Laboratório de Computação Embarcada
Florianópolis – Universidade Federal de Santa Catarina – SC – Brasil

sheiny.fabre@posgrad.ufsc.br

Resumo. *Após o posicionamento inicial dos elementos de um circuito integrado, estes precisam ser legalizados para considerar regras de fabricação. Algoritmos de legalização devem tratar grandes quantidades de dados, produzindo uma solução determinística, com a menor perturbação possível do posicionamento. Este trabalho propõe o uso da estrutura de dados KD-tree para particionar o circuito viabilizando a legalização em paralelo das partições.*

1. Introdução

A evolução do processo de fabricação de *Circuito Integrado*s (CIs) juntamente com as ferramentas de automação de projetos eletrônicos viabilizaram o projeto de CIs cada vez mais complexos, os quais possuem um maior número de transistores e regras de projeto [Flynn et al. 2007]. O projeto de CIs requer uma sequência de etapas, dentre as quais a síntese física é a responsável por gerar a descrição fabricável. A síntese física busca posicionar e rotear os *layouts* das portas lógicas, *flip-flops*, (referenciados por "células") sobre uma região 2-D [Kahng et al. 2011].

Primeiramente é necessário encontrar um posicionamento inicial para todos elementos de forma a otimizar métricas como por exemplo: redução do comprimento das interconexões [Alpert et al. 2012]. Para tratar circuitos contemporâneos complexos, isto é feito ignorando algumas restrições de legalidade do circuito, portanto células podem estar sobrepostas e desalinhadas em relação às linhas e colunas predeterminadas pelas regras de desenho. Para possibilitar a fabricação do circuito, a legalização deve remover as violações ignoradas pelo posicionamento, realizando a menor quantidade possível de movimentos para preservar a qualidade da solução [Darav et al. 2017].

Algoritmos de legalização precisam tratar grandes quantidades de dados de forma rápida e determinística, para preservar a qualidade da solução. O uso de uma estrutura de dados inapropriada para trabalhar com objetos geométricos, tais como os *layouts* das células, exige manipulações ou adaptações, aumentando o tempo de execução da técnica. Portanto este trabalho propõe o uso da estrutura de dados KD-tree para particionar o circuito, possibilitando o uso de algoritmos paralelos de legalização [Bentley 1975].

2. Uso de KD-tree na legalização

A estrutura da KD-tree é representada por uma árvore binária onde cada nó representa um ponto em um espaço k dimensional. A KD-tree 2-D é construída recursivamente onde a cada iteração par a árvore é dividida verticalmente, e nas interações ímpares horizontalmente. Para determinar onde a árvore será dividida utiliza-se o ponto mediano em relação

ao eixo da respectiva iteração. Na Figura 1, por exemplo: na primeira iteração a árvore é dividida verticalmente em relação ao eixo x onde está a célula A, e na iteração seguinte a divisão será horizontal em relação ao eixo y pelas células B e C, e assim por diante. A árvore naturalmente representa um particionamento, pois cada nó possui uma área de alcance e todos nós abrangidos podem ser obtidos visitando os descendentes do nó.

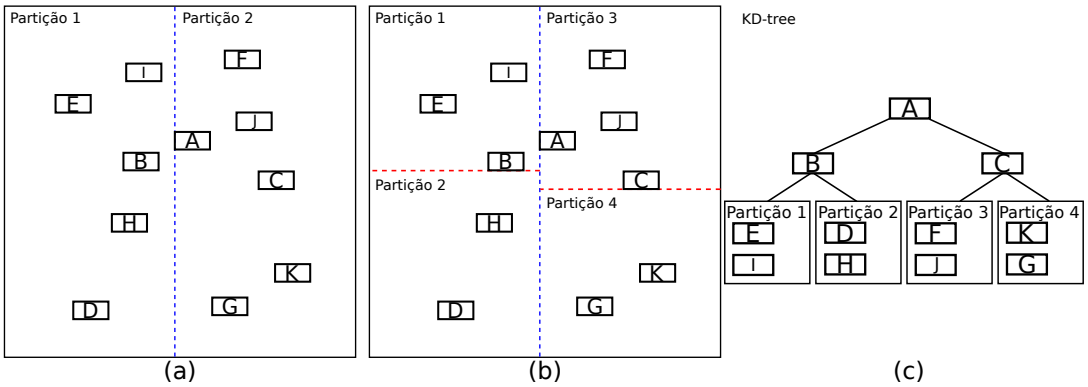


Figura 1. (a): está ilustrado a primeira iteração, (b): recursivamente, as partições estão divididas horizontalmente, (c): a estrutura final da KD-tree.

Esta estrutura de dados viabiliza a legalização em paralelo, pois cada subárvore (partição) possui uma área de alcance e um conjunto independente de células. Sendo assim, é necessário somente legalizar e fixar os nós até um determinado nível da árvore e consequentemente as subárvores desses nós serão as partições a serem legalizadas em paralelo. Por exemplo, na Figura 1 as células A, B e C seriam legalizadas e fixadas sequencialmente, e as quatro subárvores podem ser legalizadas em paralelo.

3. Trabalhos futuros

A validação técnica proposta será feita utilizando o algoritmo Abacus [Spindler et al. 2008] juntamente com os circuitos fornecidos pela competição ICCAD2017 [Darav et al. 2017], onde serão avaliados os tempos de execução, número de partições e a perturbação na qualidade da solução.

Referências

Alpert, C., Li, Z., Nam, G.-J., Sze, C. N., Viswanathan, N., and Ward, S. I. (2012). Placement: Hot or not? In *Proc. of ISPD*, pages 283–290. ACM.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Darav, N. K., Bustany, I. S., Kennings, A., and Mamidi, R. (2017). Iccad-2017 cad contest in multi-deck standard cell legalization and benchmarks.

Flynn, D., Aitken, R., Gibbons, A., and Shi, K. (2007). *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media.

Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. (2011). *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media.

Spindler, P., Schlichtmann, U., and Johannes, F. M. (2008). Abacus: fast legalization of standard cell circuits with minimal movement. In *Proc. of ISPD*, pages 47–53. ACM.

Grau de Paralelismo Adaptativo na DSL SPar

Adriano Vogel, Luiz Gustavo Fernandes

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS),
Grupo de Modelagem de Aplicações Paralelas (GMAP), Porto Alegre – RS – Brasil

adriano.vogel@acad.pucrs.br

Resumo. *As aplicações de stream apresentam características que as diferem de outras classes de aplicações, como variação nas entradas/saídas e execuções por períodos indefinidos de tempo. Uma das formas de responder a natureza dinâmica dessas aplicações é adaptando continuamente o grau de paralelismo. Nesse estudo é apresentado o suporte ao grau de paralelismo adaptativo na DSL (Domain-Specific Language) SPar.*

1. Introdução

A demanda por processar cargas dinâmicas em tempo real trouxe o paradigma de *streams*, sendo aplicações com características únicas [Andrade et al. 2014], como processamento contínuo e com variações constantes no tráfego. Diversas áreas precisam coletar e analisar dados produzidos por diversos dispositivos (ex: câmeras, radares). Objetivos frequentes de desempenho nessas aplicações são alto *throughput* e baixas latências.

Executar em paralelo é uma forma de oferecer ganhos de desempenho para aplicações de *stream*. Porém, a complexidade de programação somada a necessidade de conhecimento da arquitetura dificultam a tarefa de exploração do paralelismo. SPar [Griebler et al. 2017] é uma DSL interna para paralelismo de *stream* que busca aumentar a produtividade de código usando anotações e atributos no código sequencial, fazendo com que os programadores apenas anotem potenciais regiões a serem paralelizadas. Com essas anotações, o compilador da SPar reconhece os atributos anotados e gera código paralelo com o auxílio da biblioteca FastFlow. O paralelismo é explorado na forma de pipeline que pode conter estágios replicados com um grau de paralelismo fixo que é definido pelo programador [Griebler et al. 2017].

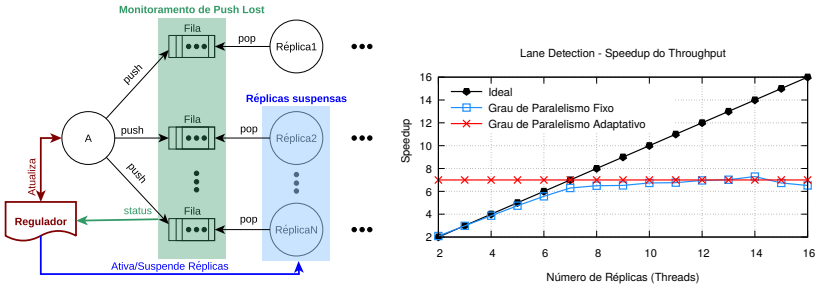
O objetivo deste trabalho é evitar que o desenvolvedor necessite especificar o grau de paralelismo adequado para uma aplicação de processamento de *stream* na DSL SPar. Além de abstrair, a dinamicidade também se faz necessária neste tipo de aplicação, pois a execução é normalmente indefinida e as cargas de trabalho variam durante a execução. Desta forma, o grau de paralelismo deve ser adaptativo para manter um bom desempenho e utilizar os recursos computacionais de forma eficiente.

Este trabalho compartilha características com outros estudos, como usar o *framework* FastFlow [Sensi et al. 2016], e o cenário de *Stream* com [Gedik et al. 2014]. Contrastando com os estudos relacionados que trazem algoritmos com execução adaptativa, esse trabalho abstrai dos programadores a necessidade de definir o grau de paralelismo. O objetivo futuro é gerar código paralelo na DSL SPar com essa funcionalidade.

2. Paralelismo Adaptativo

Grau de paralelismo adaptativo ocorre através de um mecanismo que monitora a *runtime* durante a execução. Tal mecanismo decide sobre o aumento ou a diminuição do grau

de paralelismo. Na SPar, o grau de paralelismo é o número de réplicas de um estágio em uma região de paralelismo de *stream*. O mecanismo implementado adapta o grau de paralelismo considerando estatísticas de congestionamento (usando um *threshold*) nas filas das *threads* de processamento, como mostrado na Figura 1(a).



(a) Mecanismo para Adaptação do Paralelismo.

(b) Speedup do Throughput.

Figura 1. Resultados.

O mecanismo foi avaliado[1] em uma aplicação de vídeo que faz a detecção de pistas em rodovias e usando um vídeo de *input* de 5.25MB (640x360 pixeis) . O resultado do *throughput* de execução do programa paralelo com grau de paralelismo adaptativo foi comparado com execuções do paralelismo no modo fixo e com o sequencial. A Figura 1(b) apresenta o *speedup* do *throughput* das execuções paralelas. O modo adaptativo usa um número variante de réplicas de acordo com os congestionamentos das filas das *threads* de processamento enquanto o fixo usa o mesmo grau de paralelismo na execução. O melhor *speedup* da aplicação foi com 14 réplicas no modo fixo, pois a escalabilidade da aplicação é limitada. O resultado usando um grau de paralelismo adaptativo foi próximo ao melhor caso do fixo e melhor que as demais execuções. Além disso, tornar o grau de paralelismo adaptativo considerando um objetivo de desempenho de *throughput* e latência está sendo implantado na SPar bem como a avaliação do *overhead* dos mecanismos que adaptam o grau de paralelismo.

Referências

[Andrade et al. 2014] Andrade, H., Gedik, B., and Turaga, D. (2014). *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press.

[Gedik et al. 2014] Gedik, B., Schneider, S., Hirzel, M., and Wu, K.-L. (2014). Elastic scaling for data stream processing. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1447–1463.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):20.

[Sensi et al. 2016] Sensi, D. D., Torquati, M., and Danelutto, M. (2016). A reconfiguration algorithm for power-aware parallel applications. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(4):43.

[1]Nodo com 8 cores (16 threads) e 24GB de RAM

Memórias Transacionais em Arquiteturas NUMA: Explorando Localidade de Dados e Processos

Douglas Pereira Pasqualin[1], **André Rauber Du Bois**[1], **Maurício Lima Pilla**[1]

[1]CDTec – Universidade Federal de Pelotas (UFPel)
Rua Gomes Carneiro 1 – 96.010-610 – Pelotas – RS – Brazil

{dp.pasqualin,dubois,pilla}@inf.ufpel.edu.br

Resumo. *Memórias transacionais (MTs) são um paradigma para programação concorrente, que utilizam metadados, normalmente centralizados, para garantir propriedades de consistência. Esta característica leva a excesso de contenção, sobretudo quando executados em arquiteturas NUMA (Non-Uniform Memory Architectures). Este artigo apresenta um estudo inicial sobre trabalhos com foco no aumento de desempenho de MT em arquiteturas NUMA.*

1. Introdução

Memória transacional (MT) é um paradigma para programação concorrente que visa a substituição do uso de bloqueios para gerenciar a concorrência dos programas. MT delimita trechos de códigos que devem ser executados de forma atômica, sendo que trechos de códigos que executem sem conflitos devem efetuar um *commit*, ou seja, efetivar as alterações na memória. Em caso de conflitos, uma das transações deve efetuar *abort* e reiniciar a execução, até que consiga efetivar o *commit* [Grahn 2010]. Apesar de MT poder ser desenvolvida tanto em *hardware* (MTH) quanto em *software* (MTS), este trabalho está focando somente em MTS. Muitos algoritmos de MTS utilizam um relógio global para versionamento de locais de memória, ou outros metadados centralizados com o objetivo de garantir propriedades de consistência [Grahn 2010]. Estes metadados geram muita contenção, sobretudo em arquiteturas NUMA (*Non-Uniform Memory Access*).

Este artigo apresenta um estudo de revisão bibliográfica sobre trabalhos com foco no aumento de desempenho de MTS em arquiteturas NUMA. Além disso, são comentadas oportunidades de pesquisa que não foram abordadas nos trabalhos pesquisados.

2. Trabalhos relacionados e oportunidades

Os trabalhos relacionados com MTS em arquiteturas NUMA ou com características similares à NUMA, procuram explorar a localidade de dados ou processos e podem ser divididos em três grupos. O primeiro é a separação do relógio global, por exemplo, um individual por processador físico ou por *thread* [Avni and Shavit 2008, Chan and Wang 2011, Marlier et al. 2014, Mohamedin et al. 2016]. Essa abordagem elimina a contenção do relógio centralizado, porém adiciona efeitos colaterais como a incidência de falsos *aborts*, em virtude dos relógios separados não estarem sincronizados. E caso se opte pela sincronização, um custo adicional de comunicação será adicionado. A segunda abordagem é utilizar formas de versionamento diferenciados (atrasado ou adiantado) entre conflitos que ocorram entre transações executando dentro do próprio nó NUMA ou entre nós [Wang et al. 2012]. Por fim, a última abordagem é a criação de heurísticas para efetuar a migração de *threads* entre nós, levando em consideração somente a localidade da memória

[Chan et al. 2015]. Entretanto, novas pesquisas [Gaud et al. 2015, Diener et al. 2015] sobre arquiteturas NUMA, demostram que a localidade, apesar de ainda ser importante, não é a principal causa de perda de desempenho, e sim a falta de balanceamento entre os nós.

3. Considerações finais

Apesar de MT ser um tópico bastante pesquisado, ainda existem questões com oportunidades de pesquisa, como por exemplo, melhorar o desempenho dos algoritmos quando executados em arquiteturas NUMA. Os trabalhos pesquisados focam em garantir a melhor localidade de memória e processos, evitando o acesso a nós remotos. Porém, pesquisas mais recentes apontam que o balanceamento de carga deve ser priorizado, sendo que localidade de dados deve ser o segundo parâmetro.

Com base nesses argumentos, existem algumas opções que podem ser escolhidas para a continuação do trabalho a ser desenvolvido em MTS: incluir técnicas para balanceamento de carga em arquiteturas NUMA; aprimorar a exploração da localidade dados, porém tentando reduzir a incidência de falsos *aborts*; estudar técnicas utilizadas para sincronização de relógios em sistemas distribuídos e aplicá-las em arquiteturas NUMA.

Referências

Avni, H. and Shavit, N. (2008). Maintaining consistent transactional states without a global clock. In *Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity*, SIROCCO '08, pages 131–140, Berlin, Heidelberg. Springer-Verlag.

Chan, K., Lam, K. T., and Wang, C. L. (2015). Cache affinity optimization techniques for scaling software transactional memory systems on multi-CMP architectures. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 56–65.

Chan, K. and Wang, C. L. (2011). TrC-MC: Decentralized software transactional memory for multi-multicore computers. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 292–299.

Diener, M., Cruz, E. H. M., and Navaux, P. O. A. (2015). Locality vs. Balance: Exploring data mapping policies on NUMA systems. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 9–16.

Gaud, F., Lepers, B., Funston, J., Dashti, M., Fedorova, A., Quéma, V., Lachaize, R., and Roth, M. (2015). Challenges of memory management on modern NUMA systems. *Commun. ACM*, 58(12):59–66.

Grahn, H. (2010). Transactional memory. *J. Parallel Distrib. Comput.*, 70(10):993–1008.

Marlier, P., Sobe, A., and Sutra, P. (2014). A locality-aware software transactional memory. Euro-TM Workshop on Transactional Memory (WTM 2014).

Mohamedin, M., Palmieri, R., Peluso, S., and Ravindran, B. (2016). On designing NUMA-aware concurrency control for scalable transactional memory. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, pages 45:1–45:2, New York, NY, USA. ACM.

Wang, R.-b., Lu, K., and Lu, X.-c. (2012). Aware conflict detection of non-uniform memory access system and prevention for transactional memory. *Journal of Central South University*, 19(8):2266–2271.

Meta-modelo de soluções paralelas visando a reutilização e portabilidade de componentes

Alexandre de Limas Santana, Vinicius Marino Calvo Torres de Freitas, Laércio Lima Pilla

[1]Departamento de Informática e Estatística (INE),
Universidade Federal de Santa Catarina (UFSC),
Caixa Postal 476, 88040-900, Florianópolis - SC, Brasil.

{alexandre.limas.santana,vinicius.mct.freitas}@gmail.com, llpilla@inf.ufsc.br

Resumo. *O reuso e a portabilidade de componentes de software são tópicos pertinentes em todas as áreas da computação. Em face da variedade de ferramentas e a complexidade dos sistemas, fazem-se necessárias abordagens que permitam o reuso de componentes em soluções paralelas e distribuídas. Sendo assim, este trabalho apresenta um meta-modelo para soluções paralelas de forma a explorar o reuso de componentes em diferentes sistemas.*

1. Introdução

Devido ao esforço necessário no desenvolvimento de componentes de sistemas paralelos, ferramentas como o Charm++ [Acun et al. 2016] são empregadas para auxiliar o seu desenvolvimento. Através de abstrações de conceitos relevantes, tais como informações de topologia do ambiente, é possível desenvolver módulos independentes entre si de forma ágil. Todavia, tais abstrações amarram a implementação destes componentes ao sistema paralelo, fazendo com que a interoperabilidade com diferentes sistemas de execução paralela seja limitada.

A limitação de interoperabilidade acarreta em esforços adicionais por conta da reimplementação de componentes. Escalonadores globais são componentes aplicáveis em diversas soluções paralelas que variam desde o controle de tráfego de pacotes na nuvem [Alizadeh et al. 2014] até aplicações de simulação científica de alto desempenho [Mei et al. 2011]. Uma vez que a portabilidade desse ou de outros componentes são limitados apenas ao mesmo sistema paralelo, avanços científicos destes componentes gerariam retrabalho até alcançar os seus diferentes níveis de aplicabilidade.

Neste trabalho, nós propomos uma discussão sobre as relações entre os diferentes componentes de uma solução paralela, sendo essa representada por um meta-modelo genérico. Com isso, objetivamos ressaltar possibilidades de desacoplamento de componentes como, em especial, o subsistema de balanceamento de carga.

2. Proposta

Nesta seção apresentamos a configuração de uma solução paralela através de um meta-modelo genérico. De forma a tornar o componente de balanceamento de carga independente do restante do sistema, discutiremos os requisitos para portabilidade de componentes do sistema paralelo.

Um meta-modelo de agrupamento de componentes em uma solução paralela é apresentado na Figura 1, onde componentes são agrupados em blocos conceituais, compondo uma pilha com níveis crescentes de abstração. O bloco da **Plataforma Paralela**

e Distribuída representa a plataforma física de execução e seus componentes. O bloco **Sistema Paralelo e Distribuído** agrupa os componentes responsáveis por criar, gerir e integrar as abstrações de paralelismo com a plataforma física. Os blocos **Aplicação Paralela e Distribuída** e **Balanceador de Carga** representam respectivamente a aplicação e o escalonador global. As dependências desses últimos blocos com o **Sistema Paralelo e Distribuído** são cruciais, uma vez que é a partir destas abstrações que pode-se descrever e manusear o paralelismo de forma ágil.

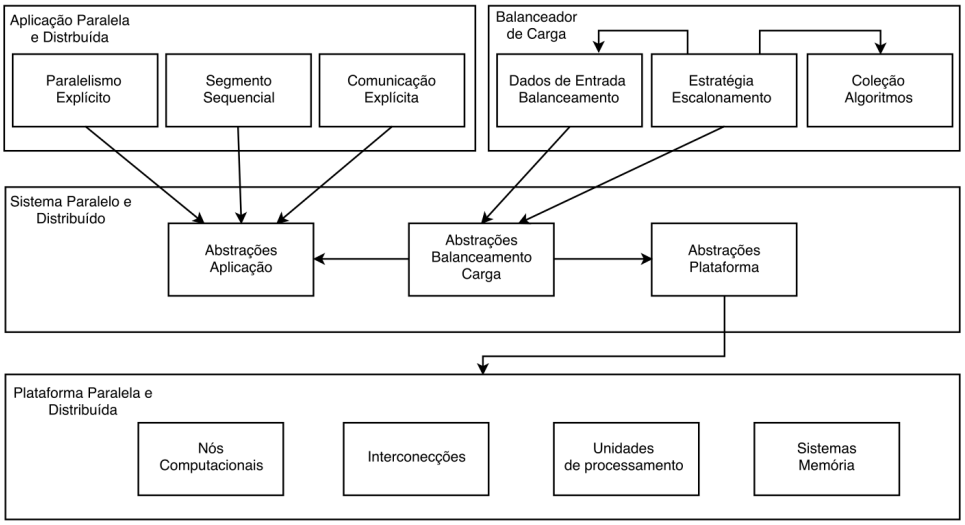


Figura 1. Pilha de Camadas de Solução Paralela.

Através deste meta-modelo, é possível perceber que os limitantes à portabilidade ocorrem quando setas de dependência incidem em um bloco diferente de onde originam-se. A respeito do subsistema de balanceamento de carga, as soluções adotadas no estado da arte consolidam seu respectivo bloco com o de **Sistema Paralelo e Distribuído**, desviando do meta-modelo e inviabilizando a portabilidade. Alternativamente, é possível alcançar a portabilidade com a criação de uma camada intermediária que, por sua vez, é incumbida de adaptar o bloco com seus vizinhos na hierarquia. Efetivamente, esta camada isola o módulo adaptado do restante do sistema. Dessa forma, ambos os componentes pertinentes às duas pontas da seta podem manter-se concisos, portáveis, e independentes sem alterações nos seus códigos-fonte. Como resultado desta técnica, a portabilidade de um componente para outro sistema paralelo não requer toda a reimplementação de um bloco, apenas da criação de um adaptador deste para uma diferente instância do meta-modelo.

Referências

Acun, B. et al. (2016). Power, reliability, and performance: One system to rule them all. *Computer*, 49(10):30–37.

Alizadeh, M. et al. (2014). Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 503–514, New York, NY, USA. ACM.

Mei, C. et al. (2011). Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime. In *Proceedings of the 2011 ACM/IEEE conference on Supercomputing*, Seattle, WA.

MigraVI: Uma proposta para Migração de Infraestruturas Virtuais em Ambientes de Computação em Nuvem

Euclides Cardoso Júnior[1]**,Charles Christian Miers**[1]**, Guilherme Piêgas Koslovski**[1]

[1]Programa de Pós-Graduação em Computação Aplicada – PPGCA
Universidade do Estado de Santa Catarina – (UDESC)

euclides.c.jr@hotmail.com, {charles.miers, guilherme.koslovski}@udesc.br

Resumo. *Provedores de Infraestrutura como Serviço (IaaS) possibilitam aos clientes a alocação de recursos computacionais que podem ser dinamicamente reservados. Em um mercado heterogêneo, um cliente pode ficar preso a um provedor de serviço (vendor lock-in). Neste sentido, a interoperabilidade entre provedores é um desafio. Dessa forma, este trabalho propõe uma solução para a migração de infraestruturas virtuais (IVs) entre provedores de serviço.*

1. Introdução

Em computação em nuvem, provedores de IaaS fornecem serviços elásticos de IV. Uma IV é composta por máquinas, contêineres e *switches* virtualizados, interconectados por uma rede virtual privada [Liu et al. 2012]. Cada recurso pode ser individualmente redimensionado para atender a carga de trabalho requerida pelas aplicações dos clientes. Provedores públicos renomados (*e.g.,* Amazon EC2 e Google) oferecem IVs com a terminologia de Nuvens Virtuais Privadas (VPC). Eventualmente, os clientes podem desejar a migração e realocação de IVs em outros provedores ou regiões geográficas.

Independentemente das questões burocráticas dos provedores envolvidos, uma migração possui diversos aspectos técnicos desafiadores que devem ser tratados. As dificuldades ocorrem em função da falta de padronização e até mesmo por problemas legais. Sobretudo, mesmo existindo abordagens centradas nos clientes para facilitar a interoperabilidade e portabilidade, provedores relutam em adotar essas tais padrões, tendo em vista que a incompatibilidade entre provedores de nuvem pode proteger seus interesses individuais [Kaur et al. 2017]. Não obstante, soluções e tecnologias que auxiliem na realização da migração de IVs encontram-se em um estado inicial de desenvolvimento [López García et al. 2016].

2. Motivação e Proposta

Realizar a migração de IVs ainda é um assunto em aberto e requer uma atenção especial, pois além de migrar os recursos hospedeiros e aplicações, toda a rede subjacente que faz parte da IV deve ser concomitantemente migrada. Nesse cenário, dois aspectos importantes da rede devem ser considerados:

1. Manutenção das conexões na rede privada interna. Após a migração, as aplicações devem retomar a execução sem a necessidade de estabelecimento de novas conexões, ou seja, devem se reencontrar internamente na nuvem.
2. Manutenção das comunicações externas com as aplicações hospedadas na IV e outros serviços de nuvem. Estas conexões são realizadas através de endereços de IP públicos

e na migração não é possível manter os mesmos endereços de IP, pois cada provedor tem a sua própria faixa de IPs. Assim, soluções para o redirecionamento do tráfego da rede devem ser utilizadas, com o objetivo de manter as conexões abertas e funcionando.

Esse trabalho apresenta uma proposta para migração de IVs entre provedores de nuvens distintos, privados ou públicos. A proposta é denominada MigraVI, que pode ser caracterizada como um corretor de nuvem, o qual irá disponibilizar para o cliente uma interface com a qual ele pode realizar a migração de sua IV entre provedores. Especificamente, MigraVI desempenha a migração de IVs compostas por contêineres, interconectados por uma rede virtual privada. Apesar de existirem outras ferramentas para a orquestração de contêineres (*e.g.,* Kubernetes), a migração dos dados salvos em memória de contêineres é pouco abordada. Esse trabalho apresenta uma proposta para a migração utilizando a ferramenta CRIU (*Checkpoint/Restore In Userspace*). A proposta compreende agentes que executam nos provedores de origem e destino, preparando um cenário para movimentação dos processos, dados e volumes.

3. Resultados Preliminares e Considerações

Um protótipo de MigraVI baseado em OpenStack foi implementado, denominado MigraVI-OpenStack. A Figura 1 apresenta os resultados preliminares obtidos a partir dos testes realizados. Os testes foram realizados na nuvem do Labortório de Processamento Paralelo e Distribuído (LabP2D) localizado na UDESC e consistem na migração de uma IV entre projetos diferentes. Os resultados estão decompostos pelo percentual das atividades realizadas durante a migração. A migração dos contêineres consume 31,27% do tempo, sendo importante ressaltar que somente durante esse período que as aplicações ficam indisponíveis. Outro ponto à ser observado é a atividade *Outros*, que é composta pelo tempo de comunicação entre as nuvens de origem e destino e a cópia de arquivos entre essas nuvens. Inicialmente, tais atividades foram consideradas secundárias. Ao finalizar os testes preliminares observou-se que estas atividades consumiram 24,16%. Como trabalho futuro será realizado novamente os testes com a aferição completa destas atividades.

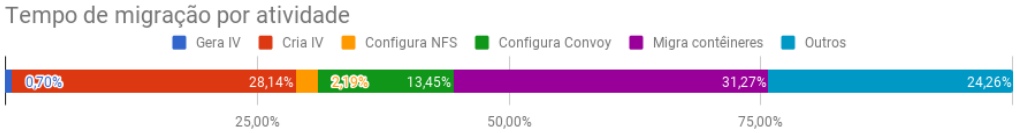


Figura 1. Resultados Preliminares com o protótipo MigraVI-OpenStack.

Referências

Kaur, K., Sharma, D. S., and Kahlon, D. K. S. (2017). Interoperability and portability approaches in inter-connected clouds: A review. *ACM Comput. Surv.*, 50(4):49:1–49:40.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2012). *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. CreateSpace Independent Publishing Platform, USA.

López García, A., Fernández del Castillo, E., and Orviz Fernández, P. (2016). Standards for enabling heterogeneous iaas cloud federations. *Comput. Stand. Interfaces*, 47(C):19–23.

MPSoC para Aumento da Resolução Espacial de Imagens Hiperespectrais

Felipe Viel, Valderi R. Q. Leithardt, Cesar Albenes Zeferino

[1] Laboratório de Sistemas Embarcados e Distribuídos - LEDS
Universidade do Vale do Itajaí (UNIVALI)
Caixa Postal 360 – CEP 88302-202 – Itajaí – SC – Brasil

felipeviel@edu.univali.br, {valderi, zeferino}@univali.br

Resumo. *Imagens hiperespectrais são utilizadas em aplicações espaciais. Essas imagens possuem alta resolução espectral, levando a um alto volume de dados. Mesmo assim, em alguns casos, é necessário aumentar a resolução espacial da imagem com um algoritmo. Este trabalho insere-se nesse contexto propondo o desenvolvimento de um sistema multiprocessado para aumentar a resolução espacial de imagens hiperespectrais a bordo de nanossatélites.*

1. Introdução

Com o passar dos anos, houve um aumento na quantidade de informações captadas por sensores em veículos espaciais, como satélites e nanossatélites. Por consequência, há uma maior quantidade de dados para serem armazenados, processados e transmitidos. Aplicações baseadas em imagens hiperespectrais ou HSIs (Hyperspectral Images) se inserem nesse contexto. Muitos veículos espaciais apenas capturam as imagens e as enviam para a estação base na Terra para processamento. Porém, a latência dessa comunicação é alta, pois a largura de banda disponível entre os veículos espaciais e a Terra é limitada – da ordem de 2 Mbps para dados em satélites pequenos [González et al. 2012] [Mandl et al. 2016].

Para lidar com essa restrição e reduzir a latência de comunicação, é necessário processar as imagens a bordo [Pintoa et al. 2016] [Ginosar et al. 2016]. Essa demanda é mais pertinente em nanossatélites, pois possuem um *downlink* ainda mais limitado, e há uma tendência de se realizar o sensoriamento da superfície da Terra com o uso de HSIs [Mandl et al. 2016]. Além disso, aplicações baseadas em HSIs requerem o o aguçamento (sharpening) da imagem para aumentar sua resolução espacial e a acurácia ao processar-lá [Vrabel et al. 2002] [Kwan et al. 2017]. Portanto, sistemas de processamento de HSIs a bordo devem ser capazes de lidar com as demandas de computação e de comunicação.

2. Solução Proposta

Para aumentar a resolução espacial de HSIs em nanossatélites, neste trabalho, propõe-se a implementação de um algoritmo de aguçamento em hardware, utilizando uma arquitetura MPSoC (Multiprocessor System-on-Chip. O MPSoC integrará vários processadores de propósito específico (SPPs – Specific Purpose Processor) para reduzir o tempo de execução e o consumo de energia em relação a uma implementação monoprocessada baseada em software.

Para tanto, o sistema utilizará como infraestrutura de comunicação uma NoC, possibilitando a integração de vários núcleos de aguçamento e permitindo a integração de

núcleos que realizem etapas de processamento e pós-processamento com HSIs e atender aos requisitos de computação e de comunicação. A Figura 1 apresenta o diagrama de blocos do sistema proposto neste trabalho. Os blocos contornados por linhas contínuas representam a região em que é realizado o aguçamento. Os demais blocos representam os demais componentes que podem compor o sistema computacional proposto.

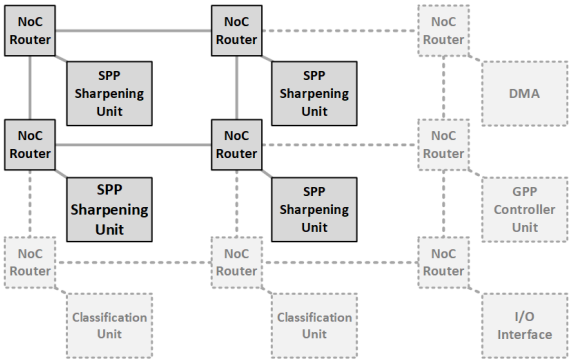


Figura 1. MPSoC com infraestrutura de aguçamento dedicada destacada

3. Considerações Finais

O provimento de uma infraestrutura em hardware dedicada ao aumento da resolução espacial de HSIs para processamento a bordo permitirá a integração de outras etapas de processamento e controle. Este trabalho contribui com o desenvolvimento de uma infraestrutura on-chip para sensoriamento remoto com o uso de nanossatélites.

Referências

Ginosar, R., Aviely, P., Israeli, T., and Meirov, H. (2016). Rc64: High performance rad-hard manycore. In *Aerospace Conference, 2016 IEEE*, pages 1–9. IEEE.

González, C., Mozos, D., Resano, J., and Plaza, A. (2012). Fpga implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 50(2):374–388.

Kwan, C., Choi, J. H., Chan, S., Zhou, J., and Budavari, B. (2017). Resolution enhancement for hyperspectral images: A super-resolution and fusion approach. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.

Mandl, D., Huemmrich, K., Crum, G., Ly, V., Handy, M., and Ong, L. (2016). Hyperspectral cubesat constellation for rapid natural hazard response. In *30th AIAA/USU Conference on Small Satellites*.

Pintoa, R., Berrojo, L., Garcia, E., Trautnerb, R., Rauwerdac, G., Sunesen, K., Redantd, S., Habince, S., Andersson, J., and Lópezf, J. (2016). Scalable sensor data processor: Architecture and development status. In *DASIA 2016 - Data Systems In Aerospace*, volume 736 of *ESA Special Publication*, page 36.

Vrabel, J. C., Doraiswamy, P., McMurtrey, J. E., and Stern, A. (2002). Demonstration of the accuracy of improved-resolution hyperspectral imagery. In *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery VIII*, volume 4725, pages 556–568. International Society for Optics and Photonics.

Otimizando Algoritmos de Machine Learning com Mapeamento de Threads e Dados*

Matheus S. Serpa, Arthur M. Krause, Eduardo H. M. Cruz, Philippe O. A. Navaux

[1] Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970, Porto Alegre – RS – Brasil

{msserpa, amkrause, ehmcruz, navaux}@inf.ufrgs.br

Resumo. *Impulsionada pelo desenvolvimento de novas tecnologias, como carros autônomos, o aprendizado de máquina tornou-se rapidamente um dos campos mais ativos da ciência da computação. Neste artigo, nos concentramos no mapeamento de threads e dados para o Intel Xeon Phi Knights Landing. Estudamos o impacto das estratégias de mapeamento, revelando que, com políticas de mapeamento inteligentes, pode-se reduzir o tempo de execução em até 18,5%.*

1. Introdução

O aprendizado de máquina é um dos tópicos da Ciência da Computação com maior crescimento, ajudando diversos campos do conhecimento devido a sua abordagem para a compreensão de grandes conjuntos de dados. Normalmente, algoritmos de aprendizado de máquina são acelerados através de *Graphics processing units* (GPUs). No entanto, os novos processadores de muitos núcleos podem ser uma escolha melhor para essas cargas de trabalho paralelas, que anteriormente eram dominadas por GPUs [Witten et al. 2016].

Nesses processadores, *threads* costumam compartilhar dados, os quais são enviados através das interconexões *intra* e *inter-chip*, dependendo da localização das *threads* e dos dados. Como consequência, mapear *threads* que se comunicam mais entre si próximas umas das outras na hierarquia de memória é uma forma de reduzir a latência de comunicação. Além disso, os dados podem estar alocados em bancos locais ou remotos, sendo que ler dados de um banco remoto resulta em latências mais altas.

Este artigo tem como objetivo mostrar que algoritmos de aprendizado de máquina podem ser acelerados utilizando um mapeamento de *threads* e dados inteligente. Para tanto, avaliamos diferentes políticas de mapeamento no Xeon Phi "Knights Landing".

2. Metodologia

A análise dos trabalhos relacionados mostra diversos trabalhos utilizando mapeamento de *threads* e dados para melhorar o desempenho de aplicações paralelas. Cruz et al. [Cruz et al. 2016] apresenta um método que usa o tempo que cada entrada fica na TLB para fazer o mapeamento. Esse método não leva em consideração a memória rápida presente no Xeon Phi. You et al. [You et al. 2017] refatora algoritmos de *machine learning* com objetivo de melhorar a comunicação. Esses trabalhos são limitados a algoritmos e arquiteturas específicas.

*Este trabalho foi parcialmente financiado por recursos do projeto HPC4E (www.hpc4e.eu), financiamento nº 689772 do acordo internacional entre o programa H2020-EU e o MCTI/RNP-Brasil. Também foi financiado pela Intel sobre o projeto Modern Code e Petrobras.

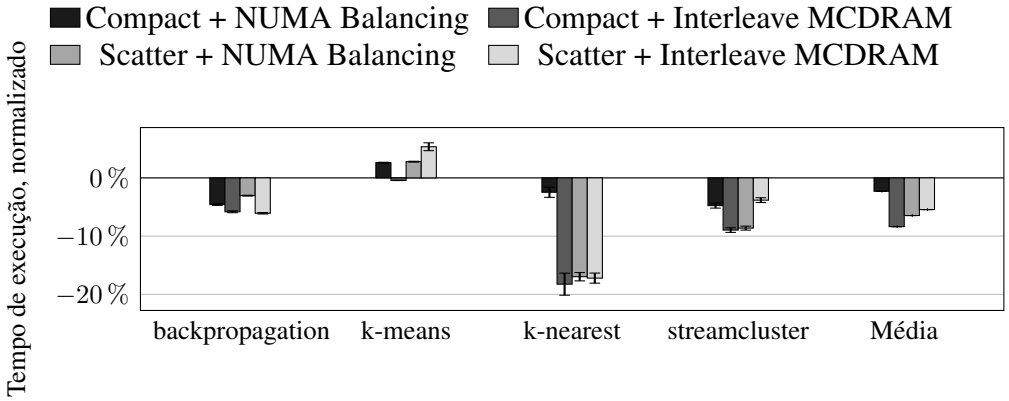


Figura 1. Mapeamento de *Threads* e Dados no Knights Landing.

O objetivo das políticas de mapeamento é melhorar o uso dos recursos, sendo que cada política foca em um aspecto. A política *compact thread* foca em agrupar *threads* vizinhas em *cores* próximos na hierarquia. Enquanto a *scatter thread* foca no inverso. O mapeamento de dados *NUMA Balancing* migra páginas para o nó da última *thread* que realizou o acesso. A política *Interleave* aloca páginas consecutivas em nós consecutivos.

Para os experimentos, uma máquina Knights Landing, com um processador Intel Xeon Phi 7250 de 68 *cores* físicos, que permite execução de 272 *threads*, foi utilizada. Esse processador também tem uma memória rápida (MCDRAM) de 16 GB. Os experimentos são a média de 30 execuções aleatórias. O desvio padrão foi calculado pela distribuição *t-student* com intervalo de confiança de 95%. Os algoritmos foram retirados do *benchmark* Rodinia em linguagem C paralelizados com OpenMP.

3. Resultados Preliminares e Conclusão

A Figura 1 apresenta os resultados sendo que o maior ganho de desempenho foi de 18.5% para um mapeamento de *threads scatter* com um mapeamento de dados *interleave MCDRAM* no algoritmo *k-nearest*. Nesse mapeamento, existem quatro nós NUMA para os quais as páginas são distribuídas. O mapeamento foi ineficaz para o algoritmo *k-means*, pois a maior parte do tempo de execução desse algoritmo é devido a entrada e saída.

Os resultados mostraram que, para algoritmos de *machine learning*, políticas que focam no balanceamento de carga são melhores. Outro ponto é que utilizar a memória rápida (MCDRAM) do Xeon Phi apresenta resultados melhores.

Referências

[Cruz et al. 2016] Cruz, E. H., Diener, M., Alves, M. A., Pilla, L. L., and Navaux, P. O. (2016). Lapt: A locality-aware page table for thread and data mapping. *Parallel Computing*, 54:59–71.

[Witten et al. 2016] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[You et al. 2017] You, Y., Buluc, A., and Demmel, J. (2017). Scaling deep learning on gpu and knights landing clusters. *arXiv preprint arXiv:1708.02983*.

Paralelização do Algoritmo de Otimização por Enxame de Partículas para Combinação de Descritores de Imagem

Handrey E. Galon[1], **Roberto S. U. Rosso Jr.**[1], **Rafael S. Parpinelli**[1]

[1]Programa de Pós-Graduação em Computação Aplicada – PPGCA
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

handrey.galon@edu.udesc.br, roberto.rosso@udesc.br, rafael.parpinelli@udesc.br

Resumo. *A Otimização por Enxame de Partículas é um algoritmo de otimização para problemas complexos de domínio contínuo. Apresenta-se neste trabalho a versão deste algoritmo implementado de forma paralela utilizando o pacote multiprocessing do Python para realizar a combinação de descritores de imagem que, quando comparado com sua versão sequencial, apresenta um speedup de médio de 1,5x.*

A Otimização por Enxame de Partículas (*Particle Swarm Optimization - PSO*) proposta por [Kennedy and Eberhart 1995] é um algoritmo de otimização para problemas complexos de domínio contínuo. Este algoritmo é um dos mais utilizados nos que se encaixam na categoria de *Swarm Intelligence*, porém, para problemas de grande porte, o elevado número de avaliações da função *fitness* restringe a sua aplicação. Por outro lado, a computação paralela é uma alternativa atraente para sua utilização, devido ao fato de ser facilmente paralelizado. Neste trabalho, são desenvolvidas uma versão serial e outra paralela de diferentes abordagens do PSO. O algoritmo paralelo é implementado utilizando o pacote *multiprocessing* do Python, que suporta a criação de processos usando uma API semelhante ao módulo de *threading*. O pacote oferece simultaneidade local e remota usando subprocessos em vez de *threads*. Devido a isso, o módulo de *multiprocessing* permite que o programador aproveite completamente os vários núcleos de uma determinada máquina. Dessa forma, este trabalho apresenta os resultados obtidos através da implementação das seguintes abordagens do algoritmo PSO: (1) PSO em sua versão canônica; (2) PSOw, proposto por [Shi and Eberhart 1998], que é uma versão do PSO com a adição do parâmetro de peso de inércia. Nesta abordagem foram utilizadas uma versão com peso de inércia fixo durante as iterações (PSOw *Offline*), e com peso de inércia adaptativo (PSOw *Online*); e, por último (3) PSOx, proposto por [Clerc and Kennedy 2002], que possui o parâmetro referente a um fator de constrição do enxame de partículas.

O objetivo do trabalho é realizar a combinação de descritores de imagem, atribuindo pesos para cada um, visando melhorar a taxa de classificação das imagens da base ETH-80, que conta com 3.280 imagens distribuídas uniformemente entre oito classes. A Equação 1 apresenta como será feita a combinação dos descritores.

$$\delta_D(I) = \sum_{1}^{n} w_i \cdot d_i(I) \quad 0 \leq w_i \leq 1 \tag{1}$$

Onde δ_D resulta na medida de similaridade entre as imagens da base, w_i é o peso atribuído à cada descritor e $d_i(I)$ são os próprios descritores. Quanto menor o valor de

w_i, menor será a influência do descritor na classificação das imagens, e quanto maior o valor de w_i, maior será sua influência. Os descritores de imagem utilizados neste trabalho são: ACC (*Auto Color Correlation*), BIC (*border/interior pixel classification*), JAC (*Joint auto-correlogram*), SID (*Invariant Steerable Pyramid Decomposition*), HTD (*Homogeneous Texture Descriptor*), SMS (*Steerable Mean Standard-deviation*), Fourier, MSF (*MultiScale Fractal*) e TSDIZ (*Tensor Scale Descriptor with Influence Zones*), os quais possuem uma descrição mais detalhada em [da Silva 2011].

O resultado de cada descritor de imagem é uma matriz com dimensões de 3.280×3.280 que representa o vetor de características de cada imagem da base. A paralelização ocorre na etapa de realizar o cálculo da matriz resultante de um descritor multiplicando pelo respectivo peso atribuído pelo PSO. Esses cálculos são utilizados para obter o valor do *fitness* das partículas, que é a própria taxa de acerto de classificação das imagens.

A execução dos algoritmos foi realizada 12 vezes para cada abordagem do PSO em uma máquina com processador Intel Core i7-4771, 3,50 GHz com 16 GB de memória RAM e com o sistema operacional Linux Ubuntu 17.10 (64 bits). A taxa de acerto na classificação das imagens da base, após a combinação dos descritores, utilizando as abordagens citadas do PSO foram de 92% em média, apresentando uma melhora quando comparado ao método de combinação de descritores utilizando a técnica de voto majoritário, que apresentou uma taxa de acerto de 89% em média. A Tabela 1 apresenta os tempos médios de processamento obtidos no processo de otimização. Verifica-se um speedup não muito expressivo dadas as características do problema e tamanho da base de dados.

Tabela 1. Resultados das abordagens testadas.

Abordagem	**Sequencial**	**Paralelo**	***Speedup***
PSO Canônico	15h49min (±0,11)	10h22min (±0,08)	1,52x
PSOw *Offline*	16h07min (±0,24)	10h38min (±0,09)	1,51x
PSOw *Online*	16h09min (±0,14)	10h43min (±0,08)	1,50x
PSOx	16h32min (±0,33)	10h42min (±0,10)	1,45x

Trabalhos futuros envolvem testar outros métodos de paralelização, como por exemplo, MPI4Py (*Message Passing Interface for Python*), visando melhorar o *speedup*.

Referências

Clerc, M. and Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.

da Silva, A. T. (2011). *Recuperação de imagens por conteúdo baseada em realimentação de relevância e classificador por floresta de caminhos ótimos*. PhD thesis, Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948.

Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. *IEEE International Conference on Evolutionary Computation Proceedings*, pages 69–73.

Paralelização do Algoritmo Evolução Diferencial em GPU com Uso de Gerador Cíclico de Índices

Mateus Boiani[1], **Rafael Stubs Parpinelli**[1]

[1] Programa de Pós-Graduação em Computação Aplicada – PPGCA
Universidade do Estado de Santa Catarina (UDESC) – Joinville – SC – Brasil

mateus.boiani@edu.udesc.br, rafael.parpinelli@udesc.br

Resumo. *A Evolução Diferencial é um algoritmo de otimização para problemas complexos de domínio contínuo. Apresenta-se aqui uma versão deste algoritmo utilizando a plataforma de computação paralela NVIDIA CUDA, chamado gDE. O gDE utiliza um gerador cíclico de índices que o diferencia das abordagens tradicionais e, quando comparado com sua versão em CPU (cDE), um speedup médio de 5× foi obtido.*

1. Introdução

A Evolução Diferencial (*Differential Evolution* - DE) foi proposta por Storn e Price [Das and Suganthan 2011] e é um dos algoritmos populacionais mais utilizados em problemas complexos de domínio contínuo. Sendo um algoritmo evolutivo, este possui como característica dois níveis de paralelismo, implícito e explícito. O primeiro é resultado da utilização de múltiplas soluções durante o processo de otimização e o segundo permitindo que a execução do algoritmo seja modelada em arquiteturas paralelas. Outros trabalhos já exploraram o uso da DE em GPU (*Graphics Processing Unit*) [Wong et al. 2015]. Um dos desafios ao executar este algoritmo em GPU é a gerar os índices aleatórios necessários nas etapas de *crossover* e mutação. Alguns autores geram todos os índices em CPU e os copiam para GPU utilizando técnicas de tentativa-e-erro para garantir que sejam diferentes entre si, e outros executam um *kernel* de tentativa-e-erro em GPU. Porém, é sabido que geração de números aleatórios em GPU possuem limitações quanto a quantidade e qualidade de amostragem. Desta forma, este trabalho apresenta uma versão do DE em GPU que utiliza uma rotina otimizada para geração de índices aleatórios, chamada de gDE. Esta rotina consiste de um vetor circular de índices, onde são sorteados K deslocamentos, o primeiro é um valor entre 0 e o tamanho da população e para os demais um valor entre 1 e o tamanho da população divido por K. Para determinar os índices é feita a soma parcial dos deslocamentos (ex: para $K = 3$, o segundo índice é obtido através da soma do 1º deslocamento com o 2º e assim sucessivamente). O algoritmo foi construído utilizando a plataforma de computação paralela NVIDIA CUDA e a versão implementada é a DE/rand/1/bin.

A primeira etapa da execução efetua a alocação da memória, inicializa os vetores de solução e os parâmetros de controle. Posteriormente, as informações são copiadas para memória global da GPU e é dado início ao processo de otimização. O gDE divide-se em 3 *kernels*, o primeiro é o gerador de índices aleatórios, que deve garantir índices diferentes entre si e diferentes do indivíduos que os usará. O segundo *kernel* do algoritmo DE efetua a avaliação dos vetores *trial* gerados. Por último, o *kernel* de substituição das melhores proles, responsável por comparar e substituir indivíduos *trial* melhor adaptados.

A Equação 1 descreve a função *Shifted Griewank* que foi escolhida para realizar a otimização. Os vetores iniciais de solução possuem tamanho D de acordo com a quantidade de dimensões a serem otimizadas. Cada valor do vetor está distribuído no intervalo de $[-600.0, +600.0]$. Ao calcular o valor da otimização é necessário aplicar o deslocamento, de modo a obter $z = x - o$, onde o é o vetor de deslocamento. $f\_bias = -180.0$ é o ponto ótimo deslocado. Ambas abordagens (cDE e gDE) utilizam taxa de 30.0% para o *crossover*, fator de mutação 0,5 e 5000 gerações. Ambas utilizam o gerador de índices proposto neste trabalho, porém, o algoritmo em DE em CPU (cDE) não é paralelizado.

$$F_1(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} cos(\frac{z_i}{\sqrt{i}}) + 1 + f\_bias \quad (1)$$

A Tabela 1 apresenta os resultados obtidos no processo de otimização para o tempo de processamento em segundos. Os testes foram executados em uma máquina com processador Intel i7-4790K, GeForce GTX 970 utilizando Linux Ubuntu 16.04. O tamanho da população utilizado é descrito por η, a quantidade de dimensões está representada por D, $\bar{x}$ é a média entre 10 execuções e σ é o desvio-padrão. Para $\eta = 1000$ o algoritmo gDE utiliza 25 blocos com 25 *threads* cada. O teste de Wilcoxon foi utilizado para analisar se existe diferença estatística entre os tempos de processamento considerando 5% de significância (*p-value* indicado). Ambas abordagens atingiram o valor ótimo da função em todos os casos mostrando a eficiência do algoritmo DE em domínios contínuos.

		cDE		**gDE**		
η	D	Tempo (s)		Tempo (s)		*p-value*
		$\bar{x}$	σ	$\bar{x}$	σ	
1000	10	35,864	0,632	7,234	0,016	0,001
	50	43,591	0,368	8,409	0,058	0,001
	100	54,240	0,617	9,962	0,023	0,001

Tabela 1. Comparação dos resultados obtidos pelo DE em CPU e em GPU.

2. Conclusão

Dos resultados obtidos é possível observar que o gDE obteve tempos computacionais inferiores quando comparado com a abordagem em CPU, mantendo a qualidade da otimização (*speedup* médio de 5×). Um dos grandes diferenciais do gDE está na forma como o gerador de índices aleatórios funciona, não existindo a necessidade de utilizar métodos de tentativa-e-erro, como é feito tradicionalmente. Em trabalhos futuros, pretende-se explorar outras funções *benchmark* com altas dimensionalidades, incluir rotinas de auto-ajuste de parâmetros e implementar o gDE em um modelo co-evolutivo.

Referências

Das, S. and Suganthan, P. N. (2011). Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31.

Wong, T. H., Qin, A. K., Wang, S., and Shi, Y. (2015). *cuSaDE: A CUDA-Based Parallel Self-adaptive Differential Evolution Algorithm*, pages 375–388. Springer International Publishing, Cham.

Performance Prediction of Stencil Applications on Accelerator Architectures

Víctor Martínez[1], **Philippe Navaux**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{victor.martinez,navaux}@inf.ufrgs.br

Resumo. *As aplicações stencil são comuns na solução de problemas relacionados com as equações diferenciais parciais, por exemplo nas simulações de geofísica. Consistem em um padrão que vai calculando a mesma operação em múltiples dados. Este trabalho apresenta um modelo para a predição do tempo de execução das aplicações stencils, baseado em Machine Learning. Os resultados mostram que é possível treinar o modelo e conseguir uma alta precisão.*

1. Introduction

The performance of HPC applications depends on many factors: architecture, code optimization, compiler and runtime frameworks. For example, [Dupros et al. 2015, de la Cruz and Araya-Polo 2015] presents cache-efficient algorithms for stencil computations on HPC architectures. On the other hand, Machine Learning (ML) is a comprehensive methodology for optimization. Recently, ML algorithms have been used on HPC systems. In [Martínez et al. 2017], the authors introduce a ML model to predict the performance of stencil computations on multicore architectures.

Stencil computations consist in using the neighboring points to evaluate a current point. The algorithm then moves to the next point applying the same computation pattern until the entire domain has been traversed. In this works, we study three well-known stencil kernels: the 7-point Jacobi for heat transfer and the seismic wave propagation explained in [Martínez et al. 2017], and the isotropic acoustic wave propagation explained in [Martínez et al. 2018]

2. Machine Learning Methodology

2.1. Testbed

We used Intel Xeon Phi (*Knights Landing*) to carry out the experiments. The detailed configuration are shown in Table 1. Based on this platform, Table 2 details all the configurations available. As it can be noted, a brute force approach would be unfeasible due to the large number of simulations required, because some of these executions can take many hours (or days).

2.2. Prediction Model

The proposed ML model is based on Support Vector Machines (SVM) and was built on top of three consecutive layers. The input layer contains the configuration values from the input vector presented in table 2. The hidden layer contains two SVMs that take values from the input vector to simulate the hardware counters, measured by PAPI library: L2 total cache misses and total cycles. Finally, the output layer contains one SVM to obtain the execution time value. SVMs were implemented in R language.

Table 1. Testbed

Processor	Intel Xeon Phi 7520
Clock(GHz)	1.40
Cores	68
Sockets	1
Threads	272
L2 cache size (MB)	32

Table 2. Configuration Domain

Optimization	Parameters	Total configurations
Number of threads	1	272
Scheduling policy	1	3
Chunk size	1	512
Total	**3**	**417,792**

2.3. Training and Validation

We created a training set by randomly selecting a subset from the configuration set presented in Table 2. Then, for each experiment we measured the hardware counters and execution time. A random testing set was used to calculate new execution time values. Table 3 presents the total number of experiments that were performed to obtain the training and validation sets. After that, we measured the accuracy of the model using the coefficient of determination (R-square). R-square ranges from zero to one, equal to one indicates a perfect fit of data prediction. As it can be noted in Table 4, the R-square is close to 99%, then we get a highly accurate regression.

Table 3. Experiment sets

	Jacobi	**Seismic**	**Isotropic**
Training set	103	163	159
Testing set	929	1469	1436
Total	1032	1632	1595

Table 4. Prediction Accuracy

	R-square
Jacobi	0.9949
Seismic	0.9993
Isotropic	0.9610

3. Conclusion

In this paper, we introduced a predictive performance modeling strategy for stencil applications on accelerator architectures. We showed that performance can be predicted with a high accuracy (96-99%). Our model is not restricted to accelerators platforms and can also be implemented into architectures with the available hardware counters to obtain the cache-related measurements. Future work is oriented to unsupervised algorithms to avoid training stage.

References

de la Cruz, R. and Araya-Polo, M. (2015). *Modeling Stencil Computations on Modern HPC Architectures*, pages 149–171. Springer International Publishing, Cham.

Dupros, F., Boulahya, F., Aochi, H., and Thierry, P. (2015). Communication-avoiding seismic numerical kernels on multicore processors. In *International Conference on High Performance Computing and Communications (HPCC)*, pages 330–335.

Martínez, V., Dupros, F., Castro, M., and Navaux, P. (2017). Performance improvement of stencil computations for multi-core architectures based on machine learning. *Procedia Computer Science*, 108:305 – 314. International Conference on Computational Science, {ICCS} 2017, 12-14 June 2017, Zurich, Switzerland.

Martínez, V., Serpa, M., Dupros, F., Padoin, E. L., and Navaux, P. (2018). *Performance Prediction of Acoustic Wave Numerical Kernel on Intel Xeon Phi Processor*, pages 101–110. Springer International Publishing, Cham.

plenUS4.0: Uma Proposta de Uso da IIoT na Embrapa Clima Temperado Explorando Programação Baseada em Fluxos

Verônica M. Tabim[1], Leonardo S. João[3], Huberto Kayser Filho[3]
João L. Lopes[2], Adenauer C. Yamin[1]

[1]Universidade Católica de Pelotas (UCPel) – Pelotas – RS
[2]Instituto Federal Sul-rio-grandense (IFSul) – Pelotas – RS
[3]Universidade Federal de Pelotas (UFPel) – Pelotas – RS

veronica.tabim@sou.ucpel.edu.br, ldrsjoao@inf.ufpel.edu.br
joaolopes@cavg.ifsul.edu.br, rodrigosouza@pelotas.ifsul.edu.br
adenauer.yamin@ucpel.edu.br

Resumo. *Este trabalho explora as premissas da IIoT (Industrial Internet of Things) no projeto PlenUS4.0, em desenvolvimento para os laboratórios da Embrapa Clima Temperado. O objetivo da pesquisa é promover um maior controle, por parte dos usuários, dos diferentes equipamentos aonde são realizadas análises, garantindo melhores níveis de precisão e confiabilidade. Os testes até o momento com as tecnologias previstas apresentaram resultados promissores.*

1. Visão Geral

O projeto plenUS4.0 expande o atual sistema de monitoramento desenvolvido para Embrapa Clima Temperado. O plenUS4.0 tem como premissa o emprego do *middleware* EXEHDA [Lopes et al. 2014] com o objetivo de disponibilizar Sistemas Ubíquos, empregando a infraestrutura provida pela Internet. A premissa é explorar relações pró-ativas entre usuários, softwares e equipamentos, promovendo assim soluções computacionais que contribuam de forma sinérgica no atendimento das atividades de pesquisa da Embrapa Clima Temperado.

Os mecanismos concebidos para o plenUS4.0 permitem um registro histórico dos estados contextuais (temperatura, umidade, etc.) em que se encontram os equipamentos dos diferentes laboratórios da Embrapa. Outrossim, está prevista a geração de alertas em decorrência do tratamento destas informações contextuais registradas.

O objetivo central do trabalho de pesquisa em desenvolvimento é explorar as premissas da IIoT (*Industrial Internet of Things*) no plenUS4.0, em particular empregando a Programação Baseada em Fluxos enquanto estratégia para processamento contextual.

A IIoT possibilita a instrumentação de processos industriais, como aqueles que acontecem na Embrapa, explorando recursos de sensoriamento, processamento, análise e conectividade, permitindo que objetos sejam monitorados através de uma infraestrutura composta por dispositivos que interoperam através de uma rede de computadores [Domova and Dagnino 2017].

A questão de pesquisa consiste em viabilizar a especificação de regras para tratamento das informações sensoriadas que qualifiquem a geração dos alertas. Nesse sentido, a combinação das estratégias da IIoT com a Programação Baseada em Fluxos pode possibilitar um monitoramento pró-ativo e autônomo dos ambientes da Embrapa, bem como um suporte à tomada de decisões gerenciais.

2. Explorando Programação Baseada em Fluxos na IIoT

Para provimento de ciência de contexto, o *middleware* EXEHDA tem sua arquitetura constituída por dois tipos de servidores: (i) Servidor de Borda que se destina a gerenciar a interação com o meio físico através de gateways; e (ii) Servidor de Contexto que atua no armazenamento e no processamento das informações contextuais, integrando dados históricos e dados provenientes de diferentes Servidores de Borda.

Os gateways são utilizados então, para tratar os diversos tipos de protocolos físicos inerentes a dispositivos de sensoriamento e/ou atuação, bem como garantir que dispositivos com capacidade restrita, tanto computacional como energética, possam se comunicar com o Servidor de Borda via TCP/IP.

No plenUS4.0 é proposto o uso do Node-RED como ferramenta para viabilizar a criação de regras diretamente pelos usuários, as quais seriam executadas no Servidor de Contexto. Na perspectiva da Programação Baseada em Fluxos, o Node-RED possibilita a conexão de dispositivos e APIs através de uma interface Web, concebidas considerando seu emprego na área da Internet das Coisas. A ferramenta é implementada na linguagem JavaScript, utilizando o *framework* Node.js [Blackstock and Lea 2014].

3. Considerações Finais

Dentre os laboratórios que serão atendidos na Embrapa temos o Laboratório de Análise de Sementes Oficial (LASO). Conforme exigência do Ministério da Agricultura, os Laboratórios de Análise de Sementes necessitam operacionalizar um sistema de qualidade baseado na norma ABNT NBR ISO/IEC 17025:2005.

Considerando as demandas decorrentes da implementação de sistema de qualidade no LASO, este trabalho explora as premissas da Industrial IoT. A expectativa é atender à exigência de maior controle, por parte dos usuários, dos equipamentos aonde serão realizadas as análises, garantindo melhores níveis de precisão e confiabilidade nos resultados.

A pesquisa está em andamento sendo feitas no momento diversas atualizações tecnológicas na infraestrutura existente na Embrapa. Após a disponibilização da interface de programação de regras empregando o Node-RED, será feita uma avaliação do plenUS4.0 empregando o método TAM (*Technology Acceptance Model*) [Yoon and Kim 2007].

Referências

Blackstock, M. and Lea, R. (2014). Towards a distributed data flow platform for the web of things. *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39.

Domova, V. and Dagnino, A. (2017). Towards intelligent alarm management in the age of iiot. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–5.

Lopes, J., Souza, R., Geyer, C., Costa, C., Barbosa, J., Pernas, A., and Yamin, A. (2014). A middleware architecture for dynamic adaptation in ubiquitous computing. *Journal of Universal Computer Science*, 20(9):1327–1351.

Yoon, C. and Kim, S. (2007). Convenience and TAM in a ubiquitous computing environment: The case of wireless LAN. *Electronic Commerce Research and Applications*, 6(1):102–112.

Processamento do fluxo de dados da rede baseado na arquitetura lambda

Alexsander Haas[1], João V.F. Lima[1]

[1]Universidade Federal de Santa Maria (UFSM)
Caixa Postal 97105-900 – Santa Maria – RS – Brasil

{ahass,jvlima}@inf.ufsm.br

Resumo. *O presente trabalho visa aplicar a arquitetura lambda na implementação de um sistema para o processamento do fluxo de dados da rede em tempo real. Realizando uma integração entre softwares open source para executar as etapas de geração dos dados, coleta, processamento e armazenamento.*

1. Introdução

Foram gerados mais de 30.000 g*igabytes* de dados por segundo na última década, essa taxa de criação continua a crescer [Marz et al. 2015]. A quantidade de dados gerados é diversa, desde postagens de usuários em redes sociais até as informações coletadas e transmitidas por dispositivos conectados à internet, também nomeada *Internet of Things* (IoT), esse conceito que eleva significativamente o tráfego de rede, pois cada vez mais dispositivos estão gerando e enviando informações. Em conjunto com esta expansão se encontra o problema de como realizar um monitoramento efetivo da rede, para que seja possível realizar o processamento do seu fluxo de dados e obter os resultados de maneira mais rápida e executar ações sobre eles.

Com base nisto o objetivo deste trabalho é aplicar a arquitetura lambda para o desenvolvimento de um sistema capaz de realizar o processamento do fluxo de dados em tempo real.

2. Sistema proposto

As abordagens de processamento de *Big Data*, geralmente utilizam técnicas de processamento em lote, utilizando *clusters* que executam os processos de modo paralelo, a técnica de execução por lote possui um tempo de resposta superior a 30 segundos, e algumas aplicações requerem respostas da ordem de sub-segundo [Rychly et al. 2014].

A arquitetura lambda é voltada para o processamento de uma carga massiva dados, ela utiliza o processamento em lote e por fluxo, trazendo uma visão em tempo real dos dados processados [Vögler et al. 2016]. Para o desenvolvimento do ambiente serão utilizados *softwares open source*, que serão integrados para que seja possível realizar o processamento por lote e em tempo real.

A arquitetura lambda é formada por três camadas [Marz et al. 2015]:

- **Lote:** que ingere e armazena uma grande quantidade de dados, realizando o processamento sobre eles.

- **Velocidade:** que ingere e processa os dados incrementando informações sobre eles, esta operação sendo realizada em uma forma de transmissão de baixa latência.
- **Serviço:** que tem como objetivo agregar as informações dos resultados obtidos da primeira e segunda camada, criando assim uma visão dos dados que foram processados pelas duas camadas anteriores.

O sistema proposto para o realizar o processamento é baseado na arquitetura lamba, e realiza a integração entre alguns *softwares open source*, destacados a seguir:

- **Bro:** ferramenta responsável pelo monitoramento em tempo real do tráfego de rede, gerando logs de informações.
- **Flume:** serviço distribuído utilizado para agregar e mover grandes quantidades de dados, utilizado como um canal para o transporte de dados.
- **Kafka:** serviço de mensagens distribuído, utilizado como um mediador de mensagens (*broker*) para abstrair todo o fluxo de dados em tópicos e por meio destes um terceiro consumir as informações.
- **Spark:** mecanismo utilizado para o processamento de dados em larga escala de forma paralela e distribuída. Realiza também processamento de fluxo em tempo real.
- **HBase:** base de dados distribuída, escalável e não relacional, executado no ecossistema do Hadoop.

Para executar as camadas da arquitetura lambda, foi realizado a seguinte integração entre os *softwares*. O Bro é responsável por gerar os logs com informações sobre o monitoramento da rede, e para o transporte destes logs é utilizado o Flume como um canal para envia-los para o Kafka, este agindo como mediador de mensagens criando tópicos sobre estas informações recebidas, para serem consumidos pelo Spark e por meio deste armazenar os dados no HBase.

4. Conclusão

A aplicação do sistema foi realizada inicialmente para o monitoramento da rede, processando em tempo real as informações obtidas e realizando o enriquecimento dos dados, para posteriormente armazena-los. Como sugestão de continuidade da implementação deste sistema, é utiliza-lo em uma análise do tráfego de rede para a detecção de intrusões por meio de algoritmos supervisionados de rede neural e árvore de decisão.

Referências

Vögler, M., Schleicher, J.M., Inzinger, C., and Dustdar, S. (2016). Ahab: A cloud-based distributed big data analytics framework for the Internet of Things. In software-practice & experience, pages 443-454.

Marz, N. and Warren, J. (2015). Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications, 1th edition.

Rychly, M., Koda, P. e Smrz, P. (2014). Scheduling decisions in stream processing on heterogeneous clusters. In Eighth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pages 614-619.

Proposta de Provisionamento Elástico de Recursos com MPI-2 para a DSL SPar

Cassiano Rista[1], **Luiz Gustavo L. Fernandes**[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Grupo de Modelagem de Aplicações Paralelas (GMAP), Porto Alegre – RS – Brasil

luis.rista@acad.pucrs.br,luiz.fernandes@pucrs.br

Resumo. *Este artigo apresenta uma proposta para desenvolvimento de um módulo de provisionamento elástico e autônomo a ser integrado em uma linguagem especifica de domínio (DSL) voltada para o paralelismo de stream. O módulo deverá explorar a elasticidade como uso de MPI-2 em um ambiente de cluster de computadores, permitindo a criação de processos em tempo de execução, serialização, ordenamento e balanceamento de carga.*

1. Introdução

O paradigma de computação em *cluster* se tornou um recurso computacional usual em muitas organizações, universidades e centros de pesquisa para atingir requisitos de computação de alto desempenho [Dantas and Rista 2005]. No entanto, tem-se verificado atualmente que a utilização desses ambientes para a execução de aplicações de *stream*, que demandam alta capacidade de processamento paralelo, tem sido evitada. Isso ocorre, devido a dificuldade de aproveitar a elasticidade, visto que esses ambientes são usualmente projetados para um número fixo de processos.

No cenário atual, aplicações que permitem o processamento de fluxos contínuos de dados, como aplicações de *stream*, vem tornando-se cada vez mais relevantes. Somado a isso, a elasticidade possibilita ainda a aquisição de recursos de forma dinâmica com base na variação da carga de trabalho do sistema. Dessa forma, explorar a elasticidade para aplicações de *stream* a partir de uma linguagem especifica de domínio (DSL) em um ambiente de *cluster* pode ser um diferencial de pesquisa. Nesse sentido, o artigo tem como objetivo principal a implementação de um módulo de elasticidade autônomo para a DSL SPar [Griebler and Fernandes 2017], permitindo o provisionamento dinâmico de recursos para aplicações de *stream* paralelas com uso de MPI-2, a partir dos modelos de *pipeline* e *farm*.

2. Trabalhos Relacionados

O interesse por pesquisas voltadas para o desenvolvimento de DSLs para a abstração de paralelismo tem aumentado consideravelmente nos últimos anos. Nesse sentido, o Grupo de Modelagem de Aplicações Paralelas (GMAP), desenvolveu originalmente uma DSL em C++ projetada para fornecer abstrações de alto nível para paralelismo de *stream* em arquiteturas *multi-core*. Recentemente, Griebler et al. [Griebler and Fernandes 2017] propôs em seu trabalho, o desenvolvimento de pesquisas direcionadas ao suporte de programação paralela distribuída para ambientes de *cluster* na DSL SPar. O objetivo é preservar a semântica original, enquanto a transformação de código *source-to-source* resultante

deverá gerar código paralelo baseado na interface de troca de mensagens (MPI). Os resultados preliminares dos experimentos demonstram uma melhor programabilidade sem significativa perda de desempenho.

Por sua vez, Righi et al. [d. R. Righi et al. 2016] apresenta neste artigo o AutoElastic, um modelo de elasticidade em nível de plataforma como serviço (PaaS) para aplicações de alto desempenho na nuvem. O diferencial desta abordagem consiste em proporcionar elasticidade para aplicações de alto desempenho sem intervenção do usuário ou modificação do código fonte. Além disso, o modelo AutoElastic assemelha-se com o estilo de programação baseado em soquetes que é oferecido pelo MPI-2, em que os novos processos podem ser facilmente criados ou removidos, a partir da computação paralela. A avaliação do protótipo usando o *middleware* OpenNebula mostrou ganhos de até 26% de desempenho no tempo de execução de uma aplicação com o gerenciador AutoElastic.

3. Proposta

Este artigo propõe o desenvolvimento de um módulo que seja capaz de habilitar o suporte a elasticidade na DSL SPar [Griebler and Fernandes 2017], permitindo assim, o provisionamento dinâmico de recursos para aplicações de *stream* paralelas, a partir de modelos como *pipeline* e *farm*. Para isso, considerou-se nessa fase do projeto de pesquisa uma estratégia de elasticidade autônoma em nível de *cluster*, permitindo assim uma maior flexibilidade para a integração junto a DSL SPar. O módulo proposto tem como objetivo assegurar que os nodos de computação do *cluster* recebam recursos suficientes para manter a execução de aplicações paralelas de *stream* dentro de um determinado nível de qualidade de serviço (QoS). Esse nível de QoS é determinado por um controlador principal de modo autônomo, não havendo necessidade de definição de *thresholds* por parte dos usuários.

Por fim, propõe-se a criação e remoção de processos em tempo de execução, serialização, ordenamento e balanceamento de carga, preservando a semântica original e gerando transformações de código *source-to-source* resultantes em MPI-2, devido a sua característica dinâmica. Como resultado, pretende-se analisar o impacto que a estratégia de elasticidade autônoma pode trazer sobre métricas como desempenho, portabilidade e eficiência, quando do processamento paralelo de aplicações de *stream*.

Referências

[d. R. Righi et al. 2016] d. R. Righi, R., Rodrigues, V. F., da Costa, C. A., Galante, G., de Bona, L. C. E., and Ferreto, T. (2016). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Transactions on Cloud Computing*, 4(1):6–19.

[Dantas and Rista 2005] Dantas, M. A. R. and Rista, C. (2005). A wireless monitoring approach for a ha-oscar cluster environment. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 302–306.

[Griebler and Fernandes 2017] Griebler, D. and Fernandes, L. G. (2017). Towards Distributed Parallel Programming Support for the SPar DSL. In *Parallel Computing: On the Road to Exascale, Proceedings of the International Conference on Parallel Computing*, ParCo'17, Bologna, Italy. IOS Press.

Proposta de um algoritmo de escalonamento de jobs baseado no consumo de energia para clusters heterogêneos

Fernando Emilio Puntel[1], **Andrea Schwertner Charão** [1], **Adriano Petry** [2]

[1] Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Santa Maria
Santa Maria, RS, Brasil

[2]Centro Regional Sul de Pesquisas Espaciais
Instituto Nacional de Pesquisas Espaciais
Santa Maria, RS, Brasil

{fepuntel, andrea}@inf.ufsm.br, adriano.petry@inpe.br

Resumo. *Com as constantes pesquisas e implementações para reduzir o consumo de energia em sistemas computacionais, o desenvolvimento de algoritmos que levem este fator em conta tem sido um ponto importante para investigação. Neste trabalho, propõe-se desenvolver uma nova estratégia de escalonamento de jobs para um sistema gerenciador de recursos em cluster heterogêneo, baseado no consumo energético.*

1. Introdução

A área de eficiência energética em HPC (*High Performance Computing*) vem sendo muito investigada atualmente. Dentre seus objetivos, está implementar e aplicar modelos e técnicas que ofereçam poder de processamento em cluster e supercomputadores e ao mesmo tempo uso moderado de recursos energéticos.

Estudos recentes descrevem técnicas para economia de energia em HPC nas mais diferentes áreas e aplicações, como na utilização de algoritmos de escalonamento e balanceamento de carga para *jobs* além de *frameworks* integrados ao Sistema Gerenciador de Recursos (SGR) em um cluster com a intenção de economizar energia. Neste contexto, o estudo de [Chiesi et al. 2015] propõe um algoritmo de escalonamento de *jobs* onde cada nó de execução possui um orçamento de energia para execução e o consumo de energia de cada nó é medido utilizando sensores físicos nos componentes do nó. A principal intenção do algoritmo do estudo é realizar o escalonamento de *jobs* com requisições heterogêneas de maneira eficiente em um cluster homogêneo. O estudo de [Georgiou et al. 2015] combina o algoritmo de escalonamento com desligamento ou alteração da frequência dos nós de processamento para redução do consumo de energia. O algoritmo de escalonamento proposto pelo estudo é avaliado com requisições de *jobs* heterogêneas de diferentes usuários e sempre deve levar em consideração o consumo de energia restante de cada nó.

Em vários ambientes utiliza-se cluster com recursos homogêneos, porém, em muitos casos os clusters são compostos por nós de processamento com diferentes configurações de hardware, o que torna um desafio maior para o escalonamento e gerenciamento. Com isso, este estudo tem como objetivo propor a implementação de uma nova estratégia de escalonamento baseado no consumo de energia para *jobs*, voltado para clusters com recursos heterogêneos. Este algoritmo tomará decisões de escalonamento

baseado no conhecimento anterior do consumo de energia dos *jobs* em cada um dos nós de processamento.

2. A proposta

A estratégia de escalonamento proposta tem como objetivo realizar o escalonamento de *jobs* em um cluster heterogêneo. O algoritmo levará em consideração o conhecimento de quanto cada *job* consome de energia em cada um dos nós de processamento do cluster. Para gerenciamento do cluster e submissão dos *jobs* será utilizado o SLURM como sistema gerenciador de recursos.

Basicamente, o algoritmo levará em consideração o conhecimento do consumo de energia de cada *job*, obtido através de execuções anteriores. Com isso, será possível realizar o escalonamento dos *jobs* de maneira eficiente, de forma que os nós que consumirem menos energia receberão mais *jobs* para processamento, enquanto que, os nós que executarem o mesmo *job* e consumirem mais energia receberão uma menor carga de trabalho reduzida.Em alguns casos, é possível que alguns nós não recebam nenhum *job* para execução, neste caso, o algoritmo identificou que a melhor estratégia para redução do consumo de energia é deixar o este nó sem nenhum *job* para execução.

A medição do consumo de energia será realizada utilizando uma ferramenta de medição de consumo energético via software, experimentos iniciais foram realizados utilizando a biblioteca Likwid, onde todos os *jobs* serão executados em todos os tipos de nós. Esta informação será atualizada pela média das últimas execuções do nó em questão.

Com o resultado da medição do consumo de energia, o algoritmo poderá realizar o escalonamento dos *jobs* no cluster heterogêneo. A fórmula a seguir apresenta como será a divisão proporcional ao consumo de energia dos *jobs* por nó, onde "A"corresponde ao nó que receberá os *jobs*, "Ea"representa o total de energia consumida para aquele *job* no nó "A"em *Joule*, e "Ea", "Eb"... "En"representam os totais de consumo de energia nos nós "a", "b"... "n", respectivamente:

$$A = \frac{\frac{1}{Ea}}{\frac{1}{Ea} + \frac{1}{Eb} + ... + \frac{1}{En}}$$

Posteriormente aos experimentos em um cluster físico, espera-se avaliar o algoritmo em um cluster simulado, a fim de avaliar a estratégia de escalonamento em grande escala.

Referências

Chiesi, M., Vanzolini, L., Mucci, C., Scarselli, E. F., and Guerrieri, R. (2015). Power-aware job scheduling on heterogeneous multicore architectures. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):868–877.

Georgiou, Y., Glesser, D., and Trystram, D. (2015). Adaptive Resource and Job Management for Limited Power Consumption. *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015*, pages 863–870.

Proposta de um escalonador de transações para o Glasgow Haskell Compiler

Rodrigo M. Duarte[1], **André R. Du Bois**[1], **Gerson G. H. Cavalheiro**[1]

[1]CDTec – Universidade Federal de Pelotas - UFPEL
LUPS - Laboratory of Ubiquitous and Parallel Systems
Gomes Carneiro 1 – 96010-610 – Pelotas – RS – Brazil

{rmduarte,dubois,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Este artigo apresenta como proposta a implementação de um escalonador de transações no Run Time do Glasgow Haskell Compiler, com o intuito de prover melhor desempenho a aplicações concorrentes que utilizem STM-Haskell como modelo de sincronização.*

1. Introdução

Memória Transacional (MT) é um modelo de sincronização entre *threads* que facilita a programação concorrente. Neste as regiões criticas do código são tratadas como transações, parecida com as presente em banco de dados [Harris et al. 2005].

Apesar do elevado nível de abstração apresentado pelas MT, as mesmas ainda sofrem degradação no desempenho quando o programa apresenta elevada contenção de memória. Isso acaba fazendo com que as transações gerem muitos conflitos, levando a re-execução consecutiva das transações e deseperdício de recursos computacionais. [Yoo and Lee 2008]. Com o intuito de resolver este problema, o escalonamento de transações vem sendo aplicado como alternativa [Maldonado et al. 2010]. A ideia principal é executar transações conflitantes em alguma ordem sequencial a fim de evitar novos conflitos.

STM Haskell é uma extensão da linguagem funcional Haskell que fornece a abstração de MT para a programação concorrente. Nela um novo tipo de variável transacional e definida (TVar) [Harris et al. 2005]. Essa variável só pode ser modificada por ações de leitura e escrita dentro de um ação transacional, ou seja, o sistema de tipos de Haskell evita que uma variável transacional seja modificada fora de uma transação. Isso fornece segurança e facilidade no desenvolvimento de programas concorrente.

Glagow Haskell Compiler (GHC) é um compilador e máquina virtual para a linguagem Haskell. Por ser a ferramenta mais atualizada e contendo em seu RTS uma implementação robusta para a concorrência, o GHC é o estado da arte [Peyton-Jones et al. 2017]. Contudo o GHC não possuiu escalonador específico para transações, levando os programas concorrentes, que possuem alta contenção de memória e que usam STM-Haskell para sincronização, apresentarem perda de desempenho.

Alguns trabalhos tem apresentado a utilização de escalonadores de transações para reduzir a taxa de conflitos e tentar aumentar o desempenho de aplicações usando MT. Entre estes trabalhos esta o CAR-STM [Dolev et al. 2008], que se utiliza de instrumentação sobre as transações, para decidir quando deve executar as transações em sequência.

LUTS [Nicácio et al. 2013] que usa heurística e instrumentação para decidir quando as transações devem ser serializadas e até trabalhos como em [Di Sanzo et al. 2016], que utilizam de cadeias de markov para decidir quando as transações devem ser executadas de forma concorrente e quando as mesmas devem ser serializadas, com o escalonador operando dinamicamente sobre o sistema transacional. Os trabalhos apresentados anteriormente são todos escalonadores implementados para bibliotecas de MT em C.

2. Proposta

Este trabalho traz como proposta a modificação do RTS (Runtime System) do GHC, para a inserção de um escalonador especifico para transações. O objetivo principal é fornecer maior desempenho a programas usando STM-Haskell. Foi escolhido o GHC devido o mesmo possuir código fonte aberto e ser o mais atualizado. A ideia inicial e fornecer um escalonador de transações simples, que execute de forma sequencial todas as transações conflitantes e depois, com o domínio do código do RTS, experimentar escalonadores mais avançados como o utilizando cadeias de Markov. Outro objetivo também e fornecer uma interface de alto nível para que o programador posso desenvolver seus próprios modelos de escalonamento.

3. Agradecimentos

O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

Referências

Di Sanzo, P., Sannicandro, M., Ciciani, B., and Quaglia, F. (2016). Markov chain-based adaptive scheduling in software transactional memory. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 373–382. IEEE.

Dolev, S., Hendler, D., and Suissa, A. (2008). Car-stm: scheduling-based collision avoidance and resolution for software transactional memory. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 125–134. ACM.

Harris, T., Marlow, S., Peyton-Jones, S., and Herlihy, M. (2005). Composable memory transactions. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 48–60. ACM.

Maldonado, W., Marlier, P., Felber, P., Suissa, A., Hendler, D., Fedorova, A., Lawall, J. L., and Muller, G. (2010). Scheduling support for transactional memory contention management. In *ACM Sigplan Notices*, volume 45, pages 79–90. ACM.

Nicácio, D., Baldassin, A., and Araújo, G. (2013). Transaction scheduling using dynamic conflict avoidance. *International Journal of Parallel Programming*, pages 1–22.

Peyton-Jones, S., Marlow, S., et al. (2017). Glasgow haskell compiler. Disponivel em https://www.haskell.org/ghc/. Acesso em: Dezembro de 2017.

Yoo, R. M. and Lee, H.-H. S. (2008). Adaptive transaction scheduling for transactional memory systems. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 169–178. ACM.

Proposta de validação de uma arquitetura para checkpoint dinâmico no Apache Hadoop

Paulo V. M. Cardoso, Patrícia Pitthan Barcelos

Pós-Graduação em Ciência da Computação (PGCC)
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

pcardoso@inf.ufsm.br, pitthan@inf.ufsm.br

Resumo. *A análise da eficiência de recuperação de um sistema em determinados cenários de falha é essencial para validar a viabilidade dos mecanismos de tolerância a falhas. Este trabalho propõe uma alternativa dinâmica para o estabelecimento de checkpoints no Apache Hadoop e define um cenário para sua validação com falhas no nó mestre da ferramenta.*

1. Introdução

Sistemas de alto desempenho devem evitar a ocorrência de erros, tornando-se necessário o emprego de técnicas de tolerância a falhas. O *Checkpoint and Recovery* (CR) consiste em uma técnica reativa de recuperação de erros por retorno cujo objetivo é conduzir o sistema a uma condição consistente, através do salvamento periódico do seu estado em momentos anteriores a falhas (*checkpoint*). A recuperação acontece após uma falha e se resume em recuperar o andamento normal do sistema.

No *framework* Apache Hadoop [White 2015], projetado para armazenar e processar grandes quantidades de dados, o *checkpoint* é usado pelo seu sistema de arquivos distribuído (HDFS - *Hadoop Distributed File System*) para alcançar tolerância a falhas. Neste sentido, o *checkpoint* é importante na manutenção do NameNode, já que este é um elemento essencial no que se refere à disponibilidade.

Contudo, o *checkpoint* no Hadoop possui configuração estática, pois não permite que o período entre salvamentos seja alterado sem que os serviços do *framework* sejam interrompidos. A partir dessa observação, este trabalho apresenta um mecanismo de *checkpoint* dinâmico para o Hadoop, a fim de garantir que o mesmo possa adaptar-se ao contexto de uso dos seus recursos. Para a validação, é definido um cenário de falhas transientes no NameNode a partir de modificações no gerenciador de aplicações do Hadoop.

2. Checkpoint Dinâmico

o Hadoop possui atributos de configuração de *checkpoint* estáticos, que são definidos antes da execução do *framework*. Por isso, a confiabilidade do sistema depende, dentre outros fatores, da definição de um período ideal entre *checkpoints*. Nesse caso, *checkpoints* mais frequentes aumentam o nível de confiabilidade mas comprometem o desempenho, enquanto *checkpoints* menos frequentes elevam o tempo de recuperação.

O mecanismo de *checkpoint* dinâmico, proposto em [Cardoso and Barcelos 2015], tem como objetivo tornar a configuração do CR no Hadoop adaptável para tornar este processo menos intrusivo, visando não comprometer a confiabilidade e o desempenho do

HDFS. A partir de eventuais mudanças na periodicidade dos *checkpoints*, o mecanismo dinâmico pode oferecer soluções adequadas de acordo com o contexto de uso do sistema.

A arquitetura proposta é constituída por alterações no HDFS, monitores independentes, além de um coordenador. Em cada elemento que seja necessário acompanhamento, um processo monitor é iniciado para identificar alterações no comportamento do nó (trocas de contexto). A partir de métricas de desempenho, o monitor envia uma mensagem ao supervisor – módulo central de monitoramento –, que indica um novo período de *checkpoint*. O coordenador do mecanismo, implementado com o auxílio do *framework* Zookeeper, armazena o novo atributo e invoca um alerta de mudança ao HDFS.

O objetivo é garantir um intervalo ideal para cada situação identificada, para que o *checkpoint* do Hadoop esteja sempre adaptado ao ambiente. Isto é, de acordo com o nível de utilização dos recursos e das métricas definidas, o atributo é atualizado e alterado sem a necessidade de interromper qualquer serviço da ferramenta. Com um intervalo adaptado em tempo de execução, o ambiente pode ser aproveitado de forma mais eficiente.

3. Validação

A validação do mecanismo define um cenário geral de falhas transientes no NameNode para analisar a eficácia da recuperação do Hadoop com os mecanismos estático e dinâmico para o *checkpoint*. A falha transiente será induzida em momentos específicos da execução de aplicações, com o comando $kill$ do Linux. O NameNode deve ser reiniciado instantes depois da falha para que se faça uma análise de sua recuperação.

Como uma falha no NameNode é caracterizada como um ponto único de falha (SPOF), os serviços que dependem do nó mestre são finalizados com erro. Assim, qualquer aplicação que use o HDFS sofre essa restrição. Para que uma falha transiente possa ser emulada, a validação propõe um mecanismo de espera no ApplicationMaster (AM) de cada aplicação. O AM é responsável pela negociação de recursos para determinada aplicação, além da coordenação dos *containers* de execução do Hadoop.

Para que os *containers* não sejam afetados pela falha no NameNode, as modificações propostas no AM incluem um *timeout* máximo de espera por uma possível recuperação do NameNode, além do tratamento da etapa de recuperação. Desta forma, a dinamicidade do mecanismo proposto pode ser validada perante um cenário de falhas no elemento mestre do HDFS, definido como meta de tolerância a falhas do *checkpoint*.

4. Próximos Passos

Com a definição dos cenários de falha, serão executados testes de desempenho com objetivo de analisar o tempo de recuperação do HDFS. Além da validação do mecanismo dinâmico nos cenários definidos, será feita uma comparação com o mecanismo de *High Availability* do Hadoop, que define uma arquitetura para eliminar o SPOF (*Single Point of Failure*) do NameNode.

Referências

Cardoso, P. V. and Barcelos, P. P. (2015). Avaliação de uma arquitetura de configuração dinâmica para o checkpoint no hdfs. *XII Encontro Anual de Tecnologia da Informação*.

White, T. (2015). *Hadoop: The Definitive Guide, 4th Edition*. "O'Reilly Media, Inc.".

Redução de sobrecarga gerada pelo uso de contadores de desempenho em ambientes virtualizados

Pedro F. Popiolek[1], **Karina S. Machado**[1], **Odorico M. Mendizabal**[1]

[1]Centro de Ciências Computacionais (C3) – Mestrado em Engenharia de Computação (PPGComp) – Grupo de Sistemas Digitais e Embarcados (GSDE)
Universidade Federal do Rio Grande - FURG
Caixa Postal 474 – 96.201-90 – Rio Grande – RS – Brasil

{p.f.popiolek,karina.machado,odoricomendizabal}@furg.br

Resumo. *Provedores de serviços em nuvem gerenciam de forma eficiente seus recursos. Contadores de desempenho auxiliam no monitoramento da saúde desses sistemas. No entanto, os contadores e as aplicações em execução competem pelos recursos disponíveis. Este trabalho investiga a sobrecarga gerada pela utilização de contadores de desempenho em ambientes virtualizados. Em adição, propõe uma abordagem para reduzir a sobrecarga produzida.*

1. Introdução

Sistemas computacionais destinados a oferecer serviços com requisitos de desempenho, e que lidam com demandas variáveis ao longo do tempo, beneficiam-se da avaliação de dados fornecidos por contadores de desempenho (CDs). A análise desses dados possibilita detectar situações que requeiram a alocação dinâmica de recursos computacionais. Isso permite aos provedores firmar níveis de acordo de serviços com os clientes, e proporcionar um uso eficiente da infraestrutura computacional [Vazquez 2015].

Esse cenário está presente em nuvens computacionais. Provedores adotam estratégias de consolidação de servidores para alcançar um gerenciamento eficiente de recursos. No entanto, aplicações podem sofrer interferência no desempenho por estarem competindo pelos mesmos recursos [Rameshan 2016]. CDs provêem dados base para a detecção ou previsão de degradação de desempenho no sistema. Contudo, sua utilização acarreta em contenção de recursos que poderiam estar disponíveis às aplicações.

Com essa perspectiva, o trabalho investiga a sobrecarga gerada por CDs em ambientes virtualizados. Além do mais explora a possibilidade de reduzir a sobrecarga gerada pela utilização de CDs. Esse estudo é baseado na redução do conjunto de CDs utilizado para o monitoramento do sistema. O foco está em reduzir a geração de dados visando não perder informações sobre o estado do sistema. Outro resultado esperado é eliminar a subjetividade no processo de seleção de CDs para o monitoramento de sistemas.

2. Metodologia

O desenvolvimento desse trabalho é baseado no estudo de dados gerados em um ambiente de experimentação controlado. O ambiente de experimentação é constituído por máquinas virtuais (MVs) em um servidor hospedeiro (SH). A quantidade de MVs é fixa para cada experimento. Cada MV possui um sistema operacional (SO) que, durante os experimentos, coletam dados de utilização de recursos do sistema enquanto executam uma

determinada carga de trabalho. Como carga de trabalho são utilizados *micro-benchmarks* (*CPU-intensive*: cálculo do número π. *IO-intensive*: operações de leitura e/ou escrita de blocos em disco. *Memory-intensive*: operações de leitura e escrita em memória.) com a capacidade de variar a intensidade de carga ao longo do experimento. Essa característica é necessária para estimular variações nos valores obtidos pelos CDs.

Os dados experimentais obtidos pelos CDs são analisados utilizando a técnica de mineração de dados de agrupamento [Torgo 2016]. Essa técnica foi aplicada com o objetivo de agrupar CDs com alto grau de similaridade, segundo o coeficiente linear de Pearson e, sem fazer distinção do tipo de correlação linear envolvida.

A partir de cada agrupamento é extraído um CD para compor o novo conjunto de contadores (reduzido). Obtido esse subconjunto para cada carga de trabalho, são realizados experimentos com as características: sem monitoramento (SM); com monitoramento utilizando todos os contadores (CM); e com monitoramento utilizando o conjunto reduzido de contadores (CMR). Assim, com o histórico de $operações/s$ (OPS) realizadas pelo *benchmark*, é possível calcular a sobrecarga gerada pelo conjunto completo e pelo conjunto reduzido de CDs, segundo: a Equação (1) para experimentos de mesma carga.

$$Sobrecarga(\%) = \left[\frac{(\textit{média de OPS CM ou CMR, exclusive}) \times 100}{\textit{média de OPS SM}}\right] - 100 \quad (1)$$

Para a realização dos experimentos foi utilizado o SO Windows Server 2012 R2 no SH e nas MVs. E como *hypervisor*, o VMware Workstation 12 Player. Foram utilizadas até duas MVs em execução concomitante, com 1 CPU e 1 GB de RAM cada. O SH utilizado possui processador Intel Xeon E3-1240V5 e 16 GB de RAM. Os experimentos foram repetidos 7 vezes para averiguação dos resultados obtidos. Os CDs utilizados foram os disponíveis por padrão na ferramenta nativa do SO utilizado, *Performance Monitor*.

3. Resultados e próximas etapas

A sobrecarga avaliada para a utilização de todos os CDs (9,108 contadores), com dados coletados a cada segundo, variou aproximadamente entre 30% a 5%. As sobrecargas mais elevadas devido a utilização de CDs foram observadas em experimentos *IO-Bound* – escritas aleatórias em disco e, principalmente, leituras aleatórias em disco. A utilização de conjuntos de CDs reduzidos, encontrados pela técnica proposta, acarretou numa redução da sobrecarga observada, ultrapassando 65% de redução. Os CDs utilizados foram reduzidos para: 384 em leituras aleatórias em disco; e 316 em escritas aleatórias em disco.

Em sequência a esse estudo pretende-se investigar os CDs que permanecem relevantes para diferentes perfis de carga de trabalho. E aqueles que são relevantes para cenários específicos de utilização de recursos. Também serão abordados: a utilização de *macro-benchmarks* nos experimentos; e a avaliação de relações não lineares entre os CDs.

Referências

Rameshan, N. (2016). On the role of performance interference in consolidated environments. In *IEEE/USENIX ICAC*. KTH Royal Institute of Technology.

Torgo, L. (2016). *Data mining with R: learning with case studies*. CRC press.

Vazquez, C. (2015). *Time series forecasting of cloud data center workloads for dynamic resource provisioning*. The University of Texas at San Antonio.

Selecionando Provedores de Computação em Nuvem via um Algoritmo Genético e baseado em Indicadores de Desempenho

Lucas Borges de Moraes[1], **Adriano Fiorese**[1], **Rafael Stubs Parpinelli**[1]

[1]Departamento de Ciências da Computação – (DCC)
Universidade do Estado de Santa Catarina – (UDESC)
Caixa Postal 631 – 89.219-710 – Joinville – SC – Brasil

lucasborges1292@gmail.com, {adriano.fiorese, rafael.parpinelli}@udesc.br

Resumo. *Esse trabalho visa especificar uma modelagem de um Algoritmo Genético (AG) capaz de selecionar o menor conjunto de provedores de nuvem que maximiza o atendimento da requisição do cliente, com o menor custo possível. A modelagem é baseada na análise dos valores e tipos de cada indicador de desempenho para cada provedor em função do requisitado pelo cliente.*

1. Introdução

O sucesso do modelo de computação em nuvem motivou o surgimento de um grande número de novos Provedores de Nuvem (PNs), tornando a tarefa de seleção do mais adequado, para cada cliente, um processo complexo. O problema da seleção de PNs já foi abordado em diferentes trabalhos [Garg et al. 2013][Baranwal and Vidyarthi 2014]. O diferencial do modelo proposto é sua capacidade de encontrar respostas mais satisfatórias para casos complexos onde o uso conjugado de mais de um PN seja necessário para atender o cliente. A medição da qualidade de um PN pode ser feita através dos seus indicadores de desempenho (PIs – *Performance Indicators*). Os PIs são classificados em quantitativos e qualitativos. Para os quantitativos, há três classificações conforme a sua utilidade de acordo com o seu valor numérico: HB ou *Higher is Better* (maior é melhor), LB ou *Lower is Better* (menor é melhor) e NB ou *Nominal is Best* (nominal é o melhor).

Encontrar o menor conjunto de PNs que maximiza o atendimento da requisição do cliente é uma operação exponencial ao número de PNs candidatos.Para resolver essa limitação pode-se aplicar uma técnica meta-heurística de busca como um Algoritmo Genético (AG).Esse trabalho apresenta a ideia de uma modelagem básica necessária para desenvolver um AG binário que resolve o problema da seleção de PNs baseada em PIs. Esse modelo possui alta abrangência e generalidade para modelagem de diversos cenários e requisições, sendo inclusive capaz de lidar com dados qualitativos.

2. Modelagem do Problema e da Solução Proposta

Dado um conjunto inicial finito não vazio P, com n distintos PNs, cada um com M distintos PIs associados, mais o custo, o problema é escolher o menor subconjunto de PNs de P, tal que $P' \subset P$, de forma a maximizar o atendimento da requisição do cliente com o menor custo envolvido. A base de dados (uma extensa lista de PNs com seus respectivos PIs conhecidos, alimentada direta ou indiretamente por corretores e/ou auditores de computação em nuvem) contém todos os PIs cadastrados, com seus tipos (HB/LB/NB) e valores, para cada PN. Os PIs são diversos: quantidade de memória (RAM, disco), processamento, disponibilidade, nível de segurança, etc. Cada PI deve estar na mesma unidade

de medida para todos os PNs (ex: PI custo em dólar por hora). A requisição representa as necessidades computacionais do cliente e deve informar todos os m PIs de interesse, com o respectivo valor desejável (X_j) e o peso daquele PI para com os outros (w_j).

Para o AG proposto é utilizado um indivíduo com codificação binária, onde a quantidade de variáveis codificadas é igual à quantidade total de PNs cadastrados na base de dados. Cada variável terá um único bit, indicando se aquele PN pertence ao conjunto solução ou não. Assim, seja $I = \{bit(0|1)_{P_1}, bit(0|1)_{P_2}, ..., bit(0|1)_{P_n}\}$ um indivíduo com um único vetor binário com n bits, o bit 1 na posição i desse vetor, indica que o i-ésimo PN de P pertence ao conjunto solução codificado por aquele indivíduo. Similarmente, o bit 0 informa que o i-ésimo PN não pertence ao conjunto solução. Logo, o espaço de busca é discreto e multimodal, de tamanho $2^n - 1$ (codificação toda preenchida com 0 não é uma solução válida). O custo total do indivíduo é igual a soma do custo de todos os PNs que pertencem ao seu conjunto solução codificado. O fitness do indivíduo é calculado conforme Equação 1, proporcional a minimização de duas componentes: a média da pontuação correspondente ao número de PNs com a pontuação do custo total daquele indivíduo; e a razão da quantidade de PIs da requisição não atendidos, exceto o custo, ponderado pelo peso de cada PI(informado na requisição). Ambas componentes estão normalizadas entre 0 e 1. O valor $n1_I$ é a quantidade de bits $1s$ do indivíduo. Os operadores genéticos utilizados são o crossover binário de um ponto de corte (prob. 95%) e mutação "bit-flip" (prob. 2%). A rotina de seleção é o torneio estocástico de tamanho 5. Parâmetros: tamanho da população de 50 com 1000 iterações.

$$fitness_I = 1 - \frac{\frac{n1_I - 1}{n-1} + \frac{\sum_{i=1}^{n} y_i \, [\, I_i = \text{bit(1)}\,]}{\sum_{j=1}^{n} y_j}}{2} - \frac{\sum_{j=1}^{m} w_j * \begin{cases} 0, \text{se } x_{ij} \geq X_j \text{ e } PI_j \in HB \\ 0, \text{se } x_{ij} \leq X_j \text{ e } PI_j \in LB \\ 0, \text{se } x_{ij} = X_j \text{ e } PI_j \in NB \\ 1, \text{caso contrario} \end{cases}}{\sum_{j=1}^{m} w_j} \tag{1}$$

3. Considerações Finais

Esse trabalho apresentou um modelo capaz de selecionar o menor conjunto de PNs que maximiza o atendimento da requisição do cliente com o menor custo possível utilizando um AG. Para tal, usa-se como base o valor e tipo de cada PI, contido na requisição e na base de dados. O modelo foi submetido a bases hipotéticas com 10, 50, 100 PNs e 5 PIs mais o custo e 5 requisições com variados níveis de dificuldade. A assertividade foi superior a 70% para os casos mais difíceis (30 execuções cada), com baixo desvio padrão e tempo de execução inferior a um segundo. Como trabalhos futuros pretende-se estender a modelagem para PIs qualitativos e aplicar a seleção para cenários com dados reais.

Referências

Baranwal, G. and Vidyarthi, D. P. (2014). A framework for selection of best cloud service provider using ranked voting method. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 831–837.

Garg, S. K., Versteeg, S., and Buyya, R. (2013). A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29:1012–1023.

SRCGreen Escalonador Verde para Grades Computacionais

Tathiana Duarte do Amarante[1], **Maurício Aronne Pillon**[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brasil

tathiduarte@gmail.com, Mauricio.Pillon@udesc.br

Resumo. *A rápida ascensão das tecnologias computacionais traz uma série de possibilidades em pesquisas voltada a Computação de Alto Desempenho. Com advento da Computacão Verde a redução do consumo de energia passou a ser uma preocupação, por outro, o incremento na quantidade de serviços ofertados, aumentou o consumo. O objetivo deste trabalho é propor um escalonador aplicado em grades computacionais buscando maximizar a eficiência energética.*

1. Introdução

Com a evolução das tecnologias computacionais voltada a *High Performance Computing* (HPC), que por seu alto teor de processamento, necessita de um maior consumo de energia para execução de suas tarefas. Desta forma a preocupação maior é no impacto negativo de consumo de energia elétrica ao meio ambiente e o alto custo financeiro [Mämmelä et al. 2012]..

As alterações climáticas que estão ocorrendo e o aquecimento global estão ocasionando o esgotamento de reservas energéticas, desta maneira a *Computação Verde* tem tomado espaço para buscar reduzir o consumo energético para atender o crescimento de empresas de Tecnologia da Informação sem gastos excessivos de energia elétrica [Scaramella and Healey 2007]. Desta maneira Ambientes HPC, tornaram-se alvo para pesquisas científicas já que possui um alto consumo de energia elétrica [Teodoro et al. 2013], [Chawla and Saluja 2016] .

Este trabalho tem por objetivo propor um escalonador de recursos em grades computacionais com o intuito de maximizar a eficiência energética em HPC. Este escalonador baseia-se em verificar a necessidade real do usuário e alocar em outros processadores, caso seja possível. Desta maneira minimiza-se o número de processadores subutilizados.

2. Escalonador SRCGreen proposto

A proposta para o trabalho consiste no desenvolvimento de um escalonador *(Scheduler Resource Computational Green)* aplicado em grades computacionais buscando a redução de consumo de energia elétrica. Nestes ambientes de grades computacionais, existe uma grande quantidade de jobs em execução, portanto é relevante que o escalonador proposto garanta que todos os processadores estejam realmente ocupados para que a eficiência energética seja atingida.

Os objetivos centrais para o desenvolvimento do escalonador são: a criação de um mecanismo que analise as tarefas recebidas, verifique a necessidade real do usuário e aloque em outros processadores subutilizado, desta maneira não se faz necessário deixar máquinas ligadas sem necessidade.

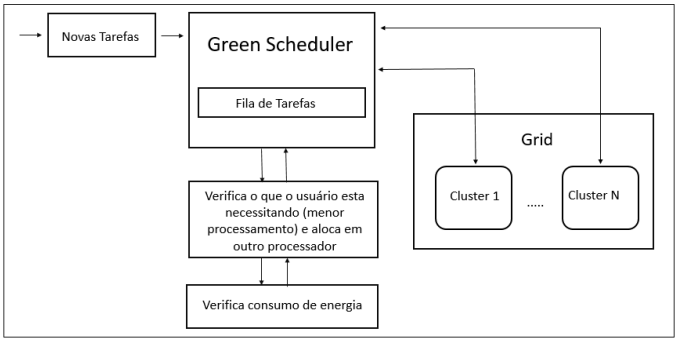


Figura 1. Modelo do escalonador Proposto

Conforme apresentado na Figura 1, o módulo *Verifica o que usuário está necessitando e aloca em outro processador* é responsável por fazer a averiguação prévia de utilização e alocar em outro processador que esteja sendo utilizado por outro usuário, não necessitando alocar em um novo processador.

Para o desenvolvimento do trabalho o ambiente utilizado será simulado, pois é preciso utilizar um grande número de testes, envolvendo assim muitos recursos tornando-se de alto custo econômico se aplicado em uma plataforma real. Para verificar o módulo *Verificar consumo de energia* será utilizado como base o [Teodoro et al. 2013] que calcula o consumo de energia baseado no consumo do host para executar determinada tarefa.

3. Considerações

Este trabalho terá como objetivo principal a redução de consumo de energia elétrica em computação em grade, buscando uma melhor eficiência energética sem causar muito impacto no desempenho. Tomando como base os escalonadores utilizados para grades computacionais, observou-se que não existia uma preocupação em minimizar o consumo energético. O presente artigo baseia-se em princípios de alguns trabalhos que buscam redução energética maximizando a eficiência da grade computacional. Em trabalhos futuros pretende-se descrever o escalonador em detalhes e dispor um protótipo para avaliação a redução de energia proporcionada pelo uso do Escalonador desenvolvido.

Referências

Chawla, S. and Saluja, K. (2016). Enhanced job scheduling algorithm with budget constraints in computational grids. In *Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on*, pages 515–520. IEEE.

Mämmelä, O., Majanen, M., Basmadjian, R., De Meer, H., Giesler, A., and Homberg, W. (2012). Energy-aware job scheduler for high-performance computing. *Computer Science-Research and Development*, pages 1–11.

Scaramella, J. and Healey, M. (2007). Service-based approaches to improving data center thermal and power efficiencies. *IDC White Paper*.

Teodoro, S., do Carmo, A. B., and Fernandes, L. G. (2013). Energy efficiency management in computational grids through energy-aware scheduling. In *Proceedings Annual ACM Symposium on Applied Computing*, pages 1163–1168. ACM.

Suporte para Computação Autonômica com Elasticidade Vertical para a DSL SPar

Gildomiro Bairros[1], **Luiz Gustavo Fernandes**[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

{gildomiro.bairros, luiz.fernandes@pucrs.br

Resumo. *O objetivo deste trabalho é propor uma solução para o suporte de elasticidade vertical em aplicações de processamento de stream desenvolvidas com a SPar. Trata-se de uma linguagem específica de domínio para expressar paralelismo de stream em alto nível. Nossa fornece suporte a elasticidade automática para ambientes em Linux Contêineres e rotinas que abstraem detalhes da infraestrutura de nuvem através da VEL (Vertical Elasticity Library).*

1. Introdução

Nos últimos anos, a computação em nuvem ganhou crescente atenção do ambiente acadêmico e corporativo. Ela é uma arquitetura que permite acesso a recursos computacionais configuráveis que podem ser provisionados com pouco esforço [Vogel et al. 2016]. Além disso, a elasticidade é considerada uma das principais propriedades para possibilitar a implementação desse modelo, onde usuário pode pagar somente pelo o que ele realmente usa. Elasticidade pode ser definida como a capacidade de dimensionar de forma dinâmica os recursos do ambiente computacional, a fim de atender as demandas da aplicação[Righi et al. 2015]. Isso possibilita que os recursos do ambiente virtual possam ser adicionados ou removidos em tempo de execução, sem interrupções de serviço.

Na sua maioria, as aplicações que demandam alto desempenho para processamento paralelo de *stream* necessitam usufruir da elasticidade, exigindo baixa latência no seu processamento ou alta vazão[Wu and Tan 2015, Griebler 2016]. Isso porque a carga varia de diferentes formas. Exemplos são: processamento de transações de log nos mercados financeiros; detecção de tendências em mídias sociais; e monitoramento de ataques maliciosos em redes de telecomunicações [Wu and Tan 2015].

Pensando no modelo de desenvolvimento de aplicações de processamento paralelo de *stream*, Griebler [Griebler et al. 2017] desenvolveu SPar. Ela é uma Linguagem Específica de Domínio que oferece uma interface intuitiva e estruturada para programação paralela, atingindo um grupo mais amplo e mais diversificado de programadores e pesquisadores. No entanto, a SPar não contempla plataformas para computação em nuvem [Griebler 2016]. Desta forma, a proposta deste trabalho é desenvolver um módulo que possa ser integrado ao compilador da SPar, permitindo a exploração de elasticidade vertical em nível de aplicação, sem a necessidade de intervenção no código.

2. Proposta

A DSL SPar desenvolvida por [Griebler et al. 2017] traz como grande benefício a facilidade em expressar o paralelismo de *stream* e a aderência ao padrão de desenvolvimento

em C++. Para que ela possa se beneficiar das plataformas de nuvem, é necessário mais do que uma infraestrutura elástica. Ela também tem que ter a capacidade de se adaptar dinamicamente de acordo com sua carga de trabalho e a sua configuração de hardware existente. Assim, ela poderá extrair o melhor desempenho do ambiente de forma transparante para o desenvolvedor.

Para resolver este problema de pesquisa, está sendo desenvolvido um módulo chamado de VEL (*Vertical Elasticity Library*). A Figura 1 apresenta a arquitetura do módulo de elasticidade desenvolvido. Este módulo trabalha com o modelo de elasticidade vertical e sua arquitetura é composta por duas camadas. Uma camada é o *VEL_Monitor_Manager* que fica na infraestrutura de nuvem, sendo responsável por receber as solicitações dos Containers e fazer o redimensionamento conforme a demanda da aplicação. A outra camada é a ***VEL_Client_API*** que fica no cliente para ser utilizada pela aplicação paralela. A camada ***VEL_Client_API*** é inserida no código durante o processo de compilação, sendo responsável por identificar qual virtualizador que a VM está utilizando e se comunicar com o ***VEL_Monitor_Manager***. O módulo VEL está sendo desenvolvido como uma biblioteca, dessa forma, é possível acoplá-lo à SPar e em outras aplicações.

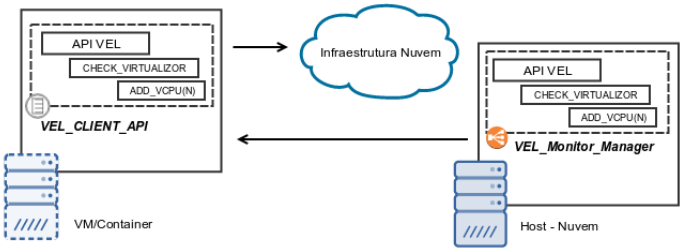


Figura 1. Arquitetura Geral da VEL (*Vertical Elasticity Library*)

Referências

[Griebler 2016] Griebler, D. (2016). *Domain-Specific Language & Support Tool for High-Level Stream Parallelism*. PhD thesis, Faculdade de Informática - PPGCC - PUCRS, Porto Alegre, Brazil.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):1740005.

[Righi et al. 2015] Righi, R., Rodrigues, V., Andre daCosta, C., Galante, G., Bona, L., and Ferreto, T. (2015). Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *Cloud Computing, IEEE Transactions on*, PP(99):1–1.

[Vogel et al. 2016] Vogel, A., Griebler, D., Maron, C. A. F., Schepke, C., and Fernandes, L. G. (2016). Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack. In *24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 672–679, Greece. IEEE.

[Wu and Tan 2015] Wu, Y. and Tan, K.-L. (2015). Chronostream: Elastic stateful stream computation in the cloud. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 723–734.

Um Estudo sobre Ferramentas de Monitoramento de Nuvens

Julio Machado[1], João Vítor V. T. de Oliveira[1], Vitor A. Ataídes[1], Maurício L. Pilla[1], Laércio L. Pilla[2]

[1]Centro de Desenvolvimento Tecnológico – UFPel
Pelotas – RS – Brasil

[2]Departamento de Informática e Estatística – UFSC
Florianópolis – SC – Brasil

{jmdsneto,jvvtdoliveira,vataides,pilla}@inf.ufpel.br, laercio.pilla@ufsc.br

Resumo. *O monitoramento de Nuvens é uma tarefa de crucial importância tanto para provedores de Nuvem quanto para clientes. Por um lado, permite o controle de como o hardware e software estão sendo utilizados. Por outro lado, provê informações de desempenho e indicações de possíveis comportamentos em plataformas e aplicações. Este trabalho apresenta um estudo sobre ferramentas de monitoramento da Nuvem.*

1. Introdução

A Computação em Nuvem [Mell and Grance 2011] se tornou o paradigma mais adotado no que diz respeito a entrega de serviços pela internet. A quantidade de serviços baseados em Computação em Nuvem cresce a cada dia, e cresce também a infraestrutura responsável por esses serviços. Para gerenciar e operar estas infraestruturas, cada vez maiores e mais complexas, é necessário um monitoramento rápido, eficiente e constante. O objetivo deste trabalho é realizar um estudo sobre as principais ferramentas de monitoramento da Nuvem.

2. Ferramentas de Monitoramento da Nuvem

O monitoramento de Nuvens é uma tarefa de crucial importância tanto para provedores de Nuvem quanto para clientes. Por um lado, permite o controle de como o *hardware* e *software* estão sendo utilizados. Por outro lado, provê informações de desempenho e indicações de possíveis comportamentos em plataformas e aplicações. Atualmente existem diversas ferramentas que monitoram nuvens, elas se diferenciam por seus objetivos e focos. Os cinco monitores estudados neste trabalho são: Data Dog, Logic Monitor, AppDynamics, New Relic e Ganglia.

DataDog [Dat 2014]**:** é um monitor que tem como principal finalidade o monitoramento escalar de nuvens e o monitoramento de aplicações de alta performace. Ele está integrado atualmente em mais de 200 plataformas, como Amazon Auto Scaling, Apache, BitBucket e etc. Ele é *open-source* e a documentação e integração das plataformas é feita pela comunidade. Sua interface permite visualizar os dados de diversas formas.

Logic Monitor [Log 2011]**:** é um monitor SaaS de código fechado utilizado por empresas como: Adidas, Siemens, Sophos e outros. O foco do Logic Monitor é prover não só o monitoramento das plataformas em camadas, permitindo o controle de fácil acesso, como também em segurança, possibilitando que o acesso as informações monitoradas seja obtido com confidencialidade.

AppDynamics [Cis 2009]**:** é um *software* pertencente a Cisco que tem como principal funcionalidade monitorar aplicações de alto desempenho. Ele é constituído por alguns componentes configuráveis: controller, MySQL database, *events service*, e opcionalmente *end user monitoring* (EUM) *server*. O AppDynamics está disponível para os sistemas Linux (RHEL 6 e 7, CentOS 6 e 7 e Ubuntu 14 e 16) e Windows (Server 2008 R2, 2012, 2012 R2 e 2016).

New Relic [New 2014] **:** é um sistema de monitoramento dinâmico e flexível. Com foco em coleta de dados em tempo real. Assim, um alerta é enviado ao usuário logo que algo pré espeficicado pelo próprio usuário em um filtro der errado.

Ganglia [Massie et al. 2004]**:** é um sistema de monitoramento distribuído voltado a computação de alta performace, como *clusters* e *grids*. Utiliza XML para representação dos dados, XDR para compactação e transferência de dados e RRDtool para armazenamento e visualização. Possui licença BSD e foi desenvolvido pela Universidade da Califórnia, sendo portável a um extenso conjunto de sistemas operacionais.

Após estudo das cinco ferramentas, observou-se que três apresentaram como um dos focos a flexibilidade, apresentando portabilidade para várias plataformas: DataDog, AppDynamics e New Relic. Enquanto o LogicMonitor apresentou como principal finalidade a segurança e confidencialidade. A ferramenta Ganglia tem como prioridade a computação de alta perfomace e a sua utilização é voltada a clusters e grids. Das cinco ferramentas, duas são *open-source*: DataDog e Ganglia.

3. Conclusão

O monitoramento da Nuvem está envolvido em praticamente todas as tarefas que caracterizam a Computação em Nuvem. Neste trabalho foram apresentadas cinco ferramentas que monitoram Nuvens, onde os objetivos variam entre flexíbilidade, segurança e performace. Na continuidade deste trabalho será feita uma análise de consumo de *cpu* e memória destas ferramentas. Em seguida será desenvolvido uma ferramenta de monitoramento de Nuvens de baixo consumo.

Referências

(2009). AppDynamics sistema de monitoramento. `https://www.appdynamics.com/`. Acessado: 30-11-2017.

(2011). Logic Monitor sistema de monitoramento. `https://www.logicmonitor.com/`. Acessado: 04-12-2017.

(2014). DataDog sistema de monitoramento. `https://www.datadoghq.com/`. Acessado: 27-11-2017.

(2014). New Relic sistema de monitoramento. `https://newrelic.com/`. Acessado: 01-12-2017.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 840.

Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States.

Um Modelo Taxonômico de Notificações e Alertas Aplicado à Privacidade de Dados

Luís Augusto Silva[1], **Valderi R. Q. Leithardt**[12], **Rudimar S. Dazzi**[1], **Jorge M. Sá Silva**[2]

[1] Laboratório de Sistemas Embarcados e Distribuídos - LEDS
Universidade do Vale do Itajaí (UNIVALI)
Caixa Postal 360 – CEP 88302-202 – Itajaí – SC – Brasil

[2]Departamento de Engenharia Informatica - Universidade Coimbra (DEI-UC)
Polo II - Pinhal de Marrocos - Coimbra - Portugal

luis.silva@edu.univali.br, {valderi, rudimar}@univali.br,

sasilva@dei.uc.pt

Resumo. *Com o crescente número de dispositivos móveis que recebem notificações diariamente, é necessário gerenciar a variedade de informações produzidas. Com isso, se faz necessário o desenvolvimento de gerenciamento e controles de informações. Este artigo propõe um modelo taxonômico para controle e gerenciamento de alertas e notificações focado em ambientes e perfil de usuários, aplicando critérios de privacidade dos dados.*

1. Introdução

A partir do crescente avanço tecnológico, buscam-se novas técnicas com finalidade de simplificar a integração entre cidadãos e a rede de sensores presentes em cidades inteligentes, prevalecendo-se de conceitos como Internet das Coisas (IoT) [Garcia et al. 2017]. A IoT pode ser explicada em três diferentes paradigmas [Ahlgren et al. 2016]: Orientado a Internet, Sensores e Conhecimento. Mantendo seu principal objetivo, garantir a conexão do grande volume de dispositivos e objetos à Internet. Servindo como apoio para a computação ubíqua, permitindo assim, o avanço e desenvolvimento de ambientes inteligentes.

Com base nas funcionalidades e com o intuito de garantir a comunição entre os usuários e dispositivos presentes em ambientes inteligentes, é usual adaptar requisitos e parâmetros predefinidos. Para tanto, são considerados os seguintes critérios para ambientes inteligentes e IoT: i) a redução de custos, ii) a melhoria da utilização, iii) a comunicação relacionada ao usuário ou outro dispositivo inserido no ambiente, necessitando um modelo taxonômico para o gerenciamento dos dados.

2. Modelo Proposto

A partir dos requisitos necessários para o funcionamento do conjunto de regras e a sua composição foi implementado um modelo taxonômico com base nas necessidades, considerando assim os requisitos necessários para uma taxonomia [Ahmed et al. 2016]. Estes requisitos formam um componente, voltados à serviços de comunicação e disseminação de informações com foco em alertas e notificações voltado para IoT. O modelo é dividido em categorias específicas, sendo empregadas conforme a necessidade.

Portanto, a contribuição deste trabalho é o desenvolvimento de uma solução para o problema do controle e gerenciamento de alertas e de notificações, levando em consideração a localização em que o usuário se encontra, suas preferências e os meios para o envio de notificações (SMS, *Push Notification* ou Aplicativos de Mensagens).

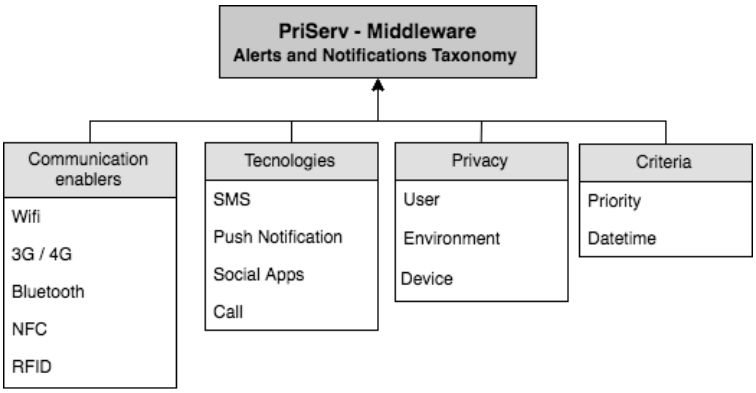


Figura 1. Taxonomia de Alertas/Notificações. Adaptado de [Leithardt et al. 2013]

O modelo taxônomico é fundamentado nas tecnologias de comunicação, conforme Figura 1. Com isso, foi possível representar os requisitos de conectividade, e também, os demais componentes necessários para envio de alertas para usuário em ambientes inteligentes. Para tanto, fundamentou-se na utilização dos critérios para o gerenciamento do envio de mensagens de forma dinâmica, de acordo com a situação e requisitos de privacidade relacionados a localização e demais parâmetros e requisitos.

3. Considerações Finais

Ao longo deste trabalho, foi possível identificar a importância em utilizar os critérios mencionados, levando em consideração a hierarquia do usuário, critérios de privacidade de acordo com o ambiente e a hierarquia atribuída. Como consequência, elaboramos o modelo de taxonomia direcionado para alertas. Pretendemos implementar mecanismos de controle e gerenciamento, relacionando ao modelo desenvolvido. Gerenciando assim, ambientes heterogêneos e tratando múltiplos tipos de notificações conforme a localização.

Referências

Ahlgren, B., Hidell, M., and Ngai, E. C. H. (2016). Internet of things for smart cities: Interoperability and open data. *IEEE Internet Computing*, 20(6):52–56.

Ahmed, E., Yaqoob, I., Gani, A., Imran, M., and Guizani, M. (2016). Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. *IEEE Wireless Communications*, 23(5):10–16.

Garcia, C., Fernandes, P., Davet, P., Lopes, J. a. L., Yamin, A., and Geyer, C. (2017). A proposal based on iot for social inclusion of people with visual impairment. In *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, WebMedia '17, pages 489–495, New York, NY, USA. ACM.

Leithardt, V. R. Q., Borges, G. A., de Morais Rossetto, A. G., Rolim, C. O., Geyer, C. F. R., Correia, L. H. A., Nunes, D., and Sa, J. (2013). A privacy taxonomy for the management of ubiquitous environments. *Journal of Communication and Computer*, 10(12).

Uma Proposta de Benchmark Paralelo para Arquiteturas Multicore

Adriano Marques Garcia[1], **Claudio Schepke**[1]

[1]Laboratório de Estudos Avançados em Informática (LEA)
Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete
Av. Tiarajú 810, Ibirapuitã – 97.546-550 – Alegrete – RS – Brazil

adriano1mg@gmail.com,claudioschepke@unipampa.edu.br

Resumo. *Este trabalho analisa um conjunto de aplicações paralelas com o objetivo de mostrar que elas possuem características distintas o suficiente para serem usadas como um benchmark. Essas aplicações foram paralelizadas utilizando: OpenMP, Pthreads, MPI-1 e MPI-2. As treze aplicações mostraram um comportamento diversificado, de tal forma que vários cenários podem ser avaliados, podendo serem utilizadas como um benchmark.*

1. Introdução

Nos últimos anos, o aumento da complexidade das aplicações para sistemas embarcados exigiu uma maior eficiência computacional e energética. Além disso, a chegada da computação *exascale*, com poder de processamento 1000 vezes maior do que os processadores *petascale* [Cappello et al. 2009], traz consigo a necessidade de aumentar o desempenho com menor impacto possível sobre o consumo de energia.

O objetivo da computação paralela é usar múltiplos processadores para executar simultaneamente diferentes partes do mesmo programa [Rauber and Rünger 2010]. No entanto, os processadores devem poder trocar informações em um determinado ponto na execução. Embora o paralelismo possibilite aumentar o desempenho, a comunicação entre processadores pode levar a um gargalo de desempenho e de consumo de energia.

O paralelismo pode ser explorado com diferentes Interfaces de Programação Paralela (IPP). Cada uma delas com suas peculiaridades em termos de sincronização e comunicação, o que pode impactar no ganho de desempenho e o consumo de energia. Se uma aplicação faz uso mais intensivo de CPU ou de memória, isso também influenciará esses fatores. Portanto, embora o paralelismo permita ganhos de desempenho, isso pode levar a um maior consumo de energia. No entanto, não há ainda um *benchmark* que ofereça um bom conjunto de aplicações, completamente paralelizadas com múltiplas IPPs e com diferentes modelos de comunicação entre tarefas.

Este trabalho reúne um conjunto de 13 aplicações desenvolvidas com o objetivo de avaliar o desempenho e o consumo de energia em arquiteturas *multi-core*. O objetivo deste trabalho é realizar uma série de testes nestas aplicações para mostrar que elas possuem características suficientes para serem usadas como *benchmark* para avaliar o desempenho e o consumo de energia de diferentes IPPs em arquiteturas *multi-core*.

2. Benchmark Proposto

Foi realizado um estudo dos trabalhos relacionados, onde foram analisados diversos outros *benchmarks* paralelos. Entretanto nenhum deles possui um bom conjunto de

aplicações completamente paralelizado e utilizando diversas IPPs. O *benchmark* NAS, por exemplo, possui algumas aplicações paralelizadas em MPI e outras em OpenMP, porém ainda é limitado quanto necessita-se realizar uma comparação de uma aplicação com várias IPPs. Dessa forma, torna-se interessante haver um *benchmark* que apresente uma mesma aplicação paralelizada de quatro maneiras distintas.

As aplicações que constituem o *benchmark* são as sequintes: Integração Numérica (NI), Cálculo do PI (PI), Produto Escalar (DP), Série Harmônica (HA), Ordenação Par-Ímpar (OE), Transformada Discreta de Fourier (DFT), Similaridade de Histogramas (HS), Jogo da Vida (GL), Turing Ring (TR), Dijkstra (DJ), Método de Jacobi (JA), Multiplicação de Matrizes (MM) e Gram-Schmidt (GS).

3. Resultados e Conclusões

A Figura 1 apresenta os resultados preliminares de uso de CPU e memória de cada aplicação, divididos por IPP. Esses resultados mostram que o uso dos recursos é bastante diversificado entre as aplicações. Além disso, as IPPs também apresentam comportamento distinto. Em média, Pthreads e OpenMP utilizaram mais memória e menos CPU. Isso faz sentido se considerarmos que *threads* se comportam como processos mais "leves" para o sistema operacional.

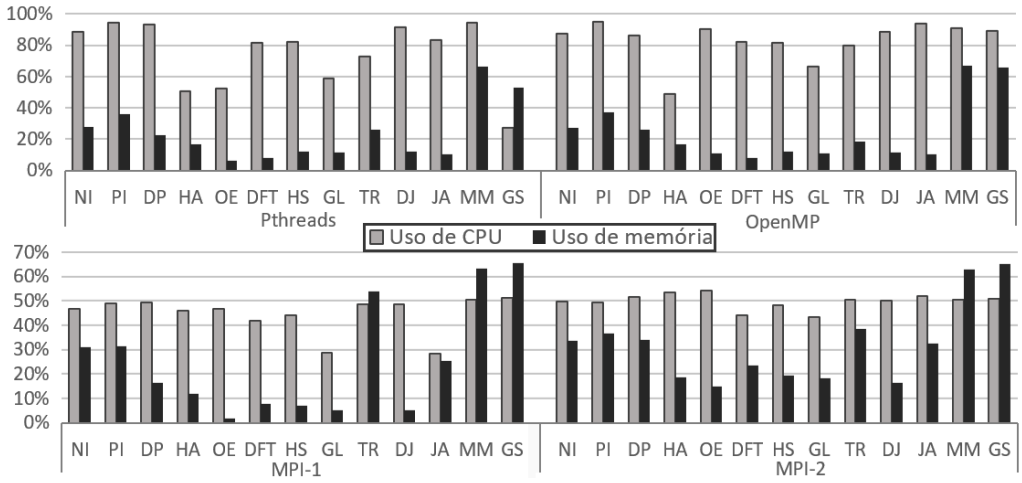


Figura 1. Uso de CPU e memória por aplicação.

Como trabalhos futuros, além de disponibilizar o *benchmark* para a comunidade, pretendemos analisar a relação entre desempenho e consumo de energia dessas aplicações. Também estudamos a possibilidade de expandir as arquiteturas alvo e implementar novas IPPs, como Intel TBT, Cilk etc.

Referências

Cappello, F., Geist, A., Gropp, B., Kale, L., Kramer, B., and Snir, M. (2009). Toward exascale resilience. *International Journal of High Performance Computing Applications*.

Rauber, T. and Rünger, G. (2010). *Parallel programming: For multicore and cluster systems*. Springer Science & Business Media.

Uma Proposta Multicelular Hierárquica para a Localização de Recursos na IoT

Huberto Kaiser Filho[1], **Renado Dilli**[1], **Ana Marilza Pernas Fleischmann**[1], **Adenauer Yamin**[1]

[1]CDTec – Universidade Federal de Pelotas (UFPEL)
R. Gomes Carneiro, 1 – 96010-610 – Pelotas – RS – Brazil

Resumo. *Os ambientes computacionais modernos providos pela IoT se caracterizam por elevados níveis de escalabilidade e heterogeneidade, em um cenário organizacional caracterizado por grande dinamicidade quanto a entrada e saída de recursos. Considerando isto, decorre a necessidade de encontrar os recursos mais qualificados para as demandas de cada usuário. Neste sentido, este artigo apresenta uma proposta de arquitetura multicelular hierárquica para a localização de recursos a ser incorporada ao middleware EXEHDA.*

1. Introdução

O contexto atual contabiliza mais de seis bilhões de coisas conectadas a IoT (*Internet of Things*), com uma previsão de crescimento para 50 bilhões até 2020 [Evans 2011] [Says 2015], cada coisa podendo disponibilizar mais de um serviço às aplicações dos usuários. A combinação de coisas e seus serviços constituem recursos a serem explorados pelos usuários. Assim, a presença desta grande quantidade de recursos na IoT torna imprescindível o uso de mecanismos para descoberta e seleção daqueles mais adequados para cada usuário.

Deste modo, o objetivo central deste trabalho é contribuir com o Subsistema de Execução do EXEHDA, qualificando o Serviço de Descoberta de Recursos para localizar os recursos mais apropriados as demandas do usuário, em um cenário de elevada escalabilidade.

2. Middleware EXEHDA

O EXEHDA [Lopes et al. 2014] é um *middleware* baseado em serviços responsável por gerenciar o ambiente ubíquo constituído por dispositivos embarcados, bem como promover a execução de aplicações cientes de contexto sobre este ambiente. Para prover suporte à tomada de decisões por parte destas aplicações, é necessário que o EXEHDA disponibilize informações provenientes destes recursos.

O EXEHDA organiza o ambiente ubíquo através de células. As células (Figura 1) são compostas por dois componentes principais: a EXEHDA Base responsável pela gerência da célula, oferecendo suporte aos diferentes serviços do middleware, dentre estes o de localização de recursos, e os EXEHDA Nodos, que são responsáveis pelas computações distribuídas. A comunicação entre células acontece de modo Peer-to-Peer, ou seja, de maneira descentralizada, caracterizando uma arquitetura fortemente distribuída.

3. Modelo Proposto

A contribuição central deste trabalho consiste em uma arquitetura hierárquica para localização de recursos, que permita um ranqueamento dos mesmos considerando os interesses do usuário.

Na proposta concebida, as células do EXEHDA que antes eram independentes, seriam organizadas de maneira hierárquica, em uma camada de visão superior as mesmas. A organização prevista comporta uma Estrutura em Árvore, onde a célula mãe mais ao topo (raiz) possui as referências e descrições de suas células filhas. Uma vez que a célula mais ao topo possui conhecimento dos recursos descobertos e disponíveis, a busca pode acontecer de maneira otimizada, onde as informações contextuais estariam referenciadas e reunidas, capacitando a célula a efetuar buscas e classificação de recursos,

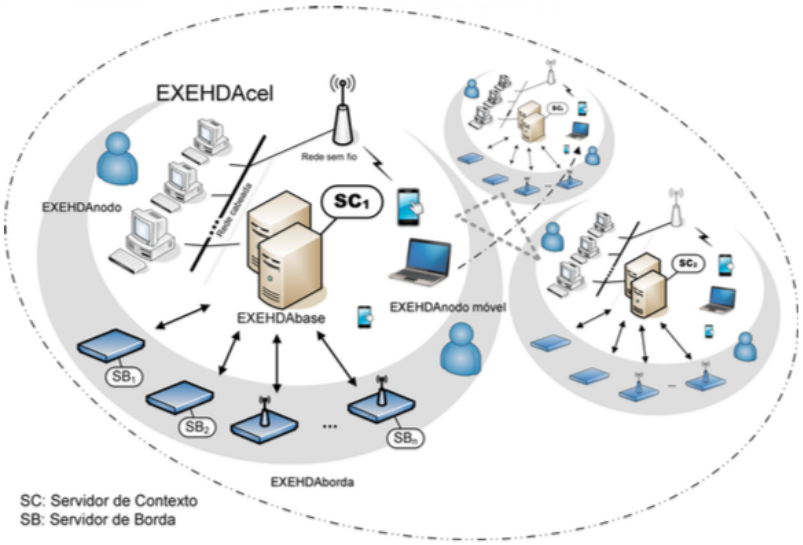


Figura 1. Célula do EXEHDA

4. Considerações Finais

A principal contribuição do trabalho quando comparado ao modelo existente é o suporte ao ranqueamento, bem como a otimização dos esforços de busca de recursos. Os componentes de software para suporte à arquitetura estão em fase de prototipação, e como Trabalhos Futuros pretende-se realizar simulações para avaliar as características pretendidas considerando a escalabilidade típica da IoT.

Referências

Evans, D. (2011). A internet das coisas: como a próxima evolução da internet está mudando tudo. *CISCO IBSG*.

Lopes, J., Souza, R., Geyer, C., Costa, C., Barbosa, J., Pernas, A., and Yamin, A. (2014). A middleware architecture for dynamic adaptation in ubiquitous computing. *j-jucs*, 20(9):1327–1351.

Says, G. (2015). 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. *Gart. Inc*.

Uma proposta para o escalonamento de jobs em ambientes de HPC com constraints monetárias e de energia elétrica *

Raul Leiria[1], **Tiago Ferreto**[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

raul.leiria@acad.pucrs.br, tiago.ferreto@pucrs.br

Resumo. *Em 2016 o supercomputador Santos Dumont do Laboratório Nacional de Computação Científica (LNCC) foi desligado por não haver recursos para pagar sua conta mensal de energia no valor de R$ 500.000,00. Com o desligamento pesquisas foram prejudicadas, incluindo um estudo sobre o Zika vírus. Neste trabalho é proposta uma política de escalonamento com constraints energéticas ajustáveis em tempo de execução para ambientes de HPC.*

1. Introdução

High Performance Computing (HPC) consiste em várias unidades ativas colaborando na resolução de um mesmo problema [Yamin et al. 2006]. Diferente da computação doméstica, a HPC foi planejada para aumentar o desempenho na execução de aplicações de maneira a tornar seu tempo de execução menor. Isso leva a sua adoção para o avanço de pesquisas científicas, busca de petróleo, modelagem climática, medicina e diversas outras áreas que requeiram alto poder computacional.

Supercomputadores fazem parte de ambientes de HPC e permitem a execução paralela de aplicações computacionais. Em 2016 a conta de energia elétrica mensal do supercomputador Santos Dumont custava cerca de R$ 500.000,00, o que corresponde à 80% dos recursos destinados ao Laboratório Nacional de Computação Científica (LNCC) por parte do Governo do Brasil [Sabóia 2016]. Neste trabalho é proposta uma política de escalonamento de *jobs* paralelos que considera *constraints* monetárias e de energia elétrica que podem ser alteradas em tempo de execução na plataforma de HPC.

2. Trabalhos relacionados

Na Tabela 1 é possível visualizar uma versão resumida da revisão bibliográfica realizada neste trabalho. O trabalho mais semelhante ao que está sendo desenvolvido é o de [Dutot et al. 2017]. Entretanto, Dutot não considera em seu algoritmo priorização de *jobs*, variações no preço da energia elétrica e tampouco no *budget* energético disponível. Apenas [Murali et al. 2015] e [Yang et al. 2013] consideram variações no preço da energia elétrica, mas em granularidades fixas (turno ou diária). [Kumbhare et al. 2017] e Dutot não implementaram priorização de *jobs* em seus trabalhos. Contudo, nenhum dos trabalhos elencados nesta seção suporta em tempo de execução variações no *budget* de energia elétrica da plataforma de HPC.

*Este trabalho foi apoiado pelo Programa PDTI, financiado pela Dell Computadores do Brasil Ltda. (Lei 8.248/91)

Autor	Abordagem	Ambiente	Priorização de *jobs*	Preço variável de energia	Budget estático
Kumbhare et al. 2017	Heurística	Real	NA	NA	Energia
Sakamoto et al. 2017	Extensão para o SLURM	Real	Automática	NA	Energia
Dutot et al. 2017	Política de escalonamento	Simulado	NA	NA	Energia
Murali et al. 2015	Política de escalonamento	Simulado	Opcional	Diário	Monetário
Yang et al. 2013	Simulador	Simulado	Perfis energéticos	Turno	Energia

Tabela 1. Trabalhos relacionados

3. Arquitetura

A arquitetura de funcionamento da política proposta pode ser visualizada na Figura 1. A política *Dynamic Easy Backfilling* é uma extensão ao algoritmo de *Easy Backfilling* com o propósito de torná-lo dinâmico em relação à execução prioritária de *jobs* sem ultrapassar um teto máximo de energia estabelecido, que pode ser variável em tempo de execução. As mudanças no *budget* ainda podem ocorrer indiretamente através de alterações no custo da energia elétrica ou no tempo máximo de operação do *cluster* dada a energia disponível.

O desenvolvimento da política está sendo realizado no *framework* SimGrid que realiza a simulação de plataformas de HPC [Dutot et al. 2017]. Em conjunto ao SimGrid estão sendo utilizadas às ferramentas BatSim para a simulação de um *Resource and Job Management System* (RJMS) e o PyBatSim [Dutot et al. 2017] como *wrapper* para o desenvolvimento da política proposta na linguagem de programação Python.

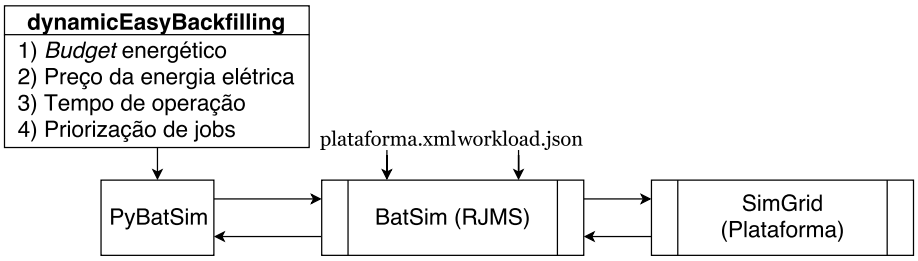


Figura 1. Política de escalonamento proposta

4. Resultados esperados

A principal contribuição deste trabalho é possibilitar que ambientes de HPC possam executar suas aplicações mesmo que hajam restrições na utilização da energia elétrica disponível. A implementação da política *Dynamic Easy Backfilling* é um trabalho em andamento e seu desenvolvimento é modularizado nas seguintes funções. (i) Predição e gerência de energia dos *jobs* e da plataforma; (ii) Reordenação de filas quando houver mudanças nos parâmetros da plataforma; e (iii) Retirada de *jobs* em execução.

Referências

Dutot, P.-F., Georgiou, Y., Glesser, D., Lefevre, L., Poquet, M., and Rais, I. (2017). Towards energy budget control in hpc. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 381–390. IEEE Press.

Sabóia, G. (2016). CBN - Ciência e Saúde - Sem dinheiro para conta de luz, supercomputador é desligado. (Acessado em 09/07/2017).

Yamin, A. C., Rose, C. A. F. D., Cavalheiro, G. G. H., and Diverio, T. A. (2006). *Caderno dos Cursos Permanentes*. ERAD.

Uma proposta para prover elasticidade e melhorar a escalabilidade do Blockchain Privado

Fabricio Reis Furtado[1], Josué Valtair Silva e Silva[1], Márcio Junior Cappellari[1], Claudio Castilhos[1], Rodrigo da Rosa Righi[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo, RS – Brasil

fabricio14@gmail.com, josue.silva@outlook.com, mcappellari@unisinos.br

claudio.seginf@gmail.com, rrrighi@unisinos.br

Resumo. *Considerando as necessidades e desafios identificados com relação a desempenho e escalabilidade na tecnologia Blockchain, será proposto neste trabalho um Gerenciador de Elasticidade para blockchain privado que atuará de forma autônoma e transparente, com a função de alocar e desalocar nós completos conforme a necessidade e fluxo de transações, gerando assim uma economia de recursos no sistema.*

1. Introdução

O blockchain é definido como um livro-razão público, onde se registram todas operações realizadas em ordem cronológica, composto por uma rede *peer-to-peer* e um banco de dados distribuído. No blockchain cada transação realizada é armazenada dentro de um bloco, composto por um conjunto de transações [1]. O blockchain, conforme a sua implementação, é dividido em três tipos: público, consórcio e privado.No blockchain público, todo nó têm o direito de validar uma transação. No blockchain consórcio, somente alguns nós têm poder de validar as transações.No blockchain privado, somente um nó ou conjunto de nós pré-determinados tem o poder de validar as transações. O blockchain consórcio e o blockchain privado tendem a ter uma melhor performance e melhor eficiência energética que o público [3]. O desperdício de poder computacional por mais de um nó para validar a mesma transação é uma das limitações que impactam todos os tipos de blockchain em maior ou menor grau.

2. Blockchain e Elasticidade

Elasticidade é capacidade que um sistema computacional possui de prover recursos automaticamente e rapidamente. Esta provisão de recursos é realizada conforme a demanda da aplicação, sendo alocados recursos quando necessário e desalocados quando não mais necessário, seja de forma manual ou automática. Na alocação automática, o sistema possui regras de ação que conforme a situação do momento ele aloca ou desaloca recursos [2]. Em um ambiente privado, a elasticidade possibilita a economia de energia elétrica, tendo um impacto positivo no viés financeiro e no meio ambiente.

3. Proposta

Utilizando os príncípios do método de revisão sistemática de literatura, foi realizada uma busca nas bases de dados científicas e aplicados filtros para seleção dos trabalhos relevantes que abordam as limitações de arquitetura e desempenho em blockchain. Analisando-os, foi identificado que nenhum abordou os aspectos de alocação dinâmica de recursos e

compatibilidade. Visto isto, optou-se por desenvolver uma arquitetura focada no blockchain privado. Para o desenvolvimento do projeto será desenvolvida uma aplicação blockchain que irá interagir com uma camada de gerenciamento dinâmico de nós completos que utilizará o conceito de elasticidade para validar transações. O objetivo desta aplicação é ser tolerante a falhas e sempre manter um número mínimo de nós completos disponíveis, conforme o aumento ou redução da demanda ir gerenciando dinamicamente outros nós completos, assim gerando economia de recursos. O modelo verifica regularmente o desempenho de CPU de cada um dos nós completos e caso a utilização atinja um limite pré determinado é alocado um novo nó para realizar a validação das transações. A arquitetura base do modelo proposto pode ser visualizada na Figura 7. O modelo possui um gerenciador de elasticidade que atua como uma camada intermediária entre os clientes (nós solicitantes) e os nós completos que validam as transações.

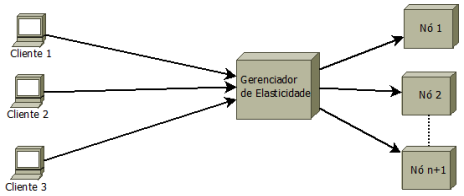


Figura 1. Arquitetura do Modelo Proposto.

O gerenciador irá trabalhar de forma autonôma com o modelo de elasticidade horizontal reativa baseado em limite superior e limite inferior. Alocando e desalocando nós conforme a carga de transações. O gerenciador também atuará como um balanceador de carga, distribuindo a carga de trabalho entre os nós.

4. Conclusão

A escalabilidade é um assunto recorrente na comunidade blockchain, pois é um grande gargalo para a ampliação da utilização da tecnologia. Soluções blockchain ainda possuem uma desvantagem em performance com relação a sistemas centralizados. Há uma dificuldade em melhorar a escalabilidade do blockchain pois uma reparametrização do mesmo pode gerar problemas de segurança. Outro aspecto negativo é o consumo de recursos pois a forma que o mesmo é implementado hoje exige um grande volume de máquinas alocadas para a validação das transações. O blockchain privado é um conceito novo e ainda pouco explorado pelas empresas. Este modelo possui uma melhor performance justamente por alguns gargalos do blockchain público estarem presentes em menor escala ou inexistir.

Referências

[1] Bitcoin.org. Bitcoin developer guide, 2009. Disponível em: ¡https://bitcoin.org/en/developer-guide¿. Acesso em: nov. 2017.

[2] Guilherme Galante and Luis Carlos E De Bona. A survey on cloud computing elasticity. *Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, (1):263–270, October 2012.

[3] Lakshmi Siva Sankar, M. Shindu, and M. Sethumadhavan. Survey of consensus protocols on blockchain applications. *2017 4th International Conference on Advanced Computing and Communication Systems, ICACCS 2017*, 2017.

Uma Suíte de *Benchmarks* Parametrizáveis para o Domínio de Processamento de Stream em Sistemas Multi-Core

Carlos A. F. Maron[1], **Luiz Gustavo Fernandes**[1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Grupo de Modelagem de Aplicações Paralelas (GMAP), Porto Alegre – RS – Brasil

carlos.maron@acad.pucrs.br,luiz.fernandes@pucrs.br

Resumo. *Avaliar o desempenho é importante para computação. Porém, assim como o hardware, o software também deve ser avaliado quando características podem influenciar no seu comportamento. Nestes casos, a suíte de benchmarks parametrizáveis para o processamento de stream serve como uma ferramenta de apoio ao usuário e até programadores.*

1. Introdução

Aplicações do domínio de *stream* estão se destacando nos últimos tempos. Tanto na indústria como na academia, estas aplicações se tornaram cada vez mais recorrentes no processamento de informações. Alguns exemplos destas aplicações estão presentes em nosso cotidiano, como aplicações que processam imagem, vídeo e áudio, redes sociais, casas inteligentes, *Big Data*, entre outras. As aplicações deste domínio processam um *stream*, formado por conjuntos infinitos de elementos (Ex, imagens, *frame*, estrutura, etc.). Tais aplicações têm um comportamento semelhante ao de uma linha de produção automobilística, onde existe uma sequência de postos de trabalhos que realizam uma tarefa sobre cada um dos carros que entram nesta linha. No código de uma aplicação, os postos de trabalhos são os estágios que realizam as operações sobre cada um dos elementos.

Paralelizar estas aplicações pode ser um desafio para encontrar a melhor configuração de desempenho. Cada aplicação tem particularidades ao lidar com o seu fluxo contínuo, pois muitas propriedades são variáveis. Por exemplo, os estágios podem realizar computações diversificadas, com ou sem estado interno, elementos do *stream* ter tamanhos e tipos diferentes. Além disso, algumas aplicações podem exigir diferentes propriedades no desempenho, como vazão ou latência. Desse modo, avaliar o desempenho neste cenário pode ser complicado, já que paralelizar estas aplicações é um desafio [Griebler et al. 2017, Griebler et al. 2018] e algumas propriedades podem impactar no desempenho.

No entanto, ao deparar-se com as clássicas suítes de *benchmarks*, como NAS [Bailey et al. 1991] e PARSEC [Bienia 2011], que possuem aplicações do domínio de *stream*, percebe-se que essas aplicações não refletem totalmente o comportamento e a paralelização de aplicações deste tipo. Além disso, são *benchmarks* usados para mostrar o desempenho da infraestrutura e não abordam as propriedades das aplicações de *stream*, além de que a parametrização nestas suítes é limitada. Basicamente, resumem-se na definição do número de *threads*, no tamanho da entrada ou complexidade do processamento.

2. Proposta

Este trabalho propõem a suíte BenSP - *Benchmark for Stream Parallelism*. Uma suíte de *benchmarks* para o domínio de processamento de *stream* em sistemas *multi-core*.

Com o BenSP as propriedades de *stream* podem ser testadas em uma aplicação por meio de parâmetros. Essa característica da suíte foi inspirada no *microbenchmark* Eigenbench [Hong et al. 2010], que deriva características e comportamentos de sistemas de memória transacional por meio de parâmetros.

A Figura 1 mostra a possibilidade de parametrização da aplicação Dedup na suíte BenSP. Em aplicações como o Dedup, utilizadas para remoção de dados duplicados e compressão de arquivos [Bienia 2011], o processo de fragmentação define o tamanho médio do elemento de *stream* (*chunk*), resultando em diferentes comportamentos como o aumento do uso de memória, tempo de execução, taxa de deduplicação e compressão. Testes realizados com a parametrização mostraram um ganho no tempo de execução de 67%.

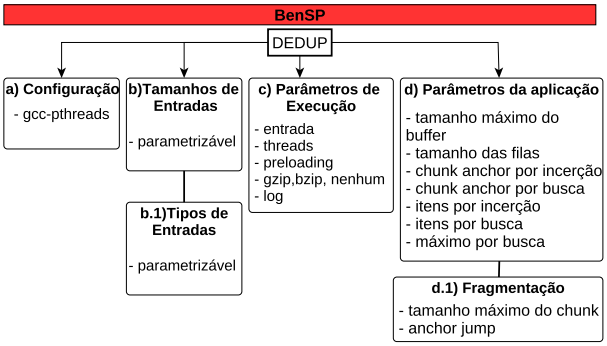


Figura 1. Parametrização Suíte BenSP.

Além do Dedup, aplicações como Ferret e DPI (*Deep Packet Inspection*) também fazem parte da suíte BenSP. Por exemplo, o elemento de *stream* (*frame*) no Ferret é um dos parâmetros que pode ser ajustado. Neste caso, como a aplicação processa a busca de similaridade em imagens, vídeo e áudio, isso influenciará na precisão e no tempo de execução da aplicação. No DPI, o tamanho do elemento de *stream* (pacote de rede) também pode ser ajustado, inferindo no reconhecimento do tráfego de rede, por exemplo.

Referências

[Bailey et al. 1991] Bailey, D. H. et al. (1991). The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73.

[Bienia 2011] Bienia, C. (2011). *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University.

[Griebler et al. 2017] Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):1740005.

[Griebler et al. 2018] Griebler, D., Filho, R. B. H., Danelutto, M., and Fernandes, L. G. (2018). High-Level and Productive Stream Parallelism for Dedup, Ferret, and Bzip2. *International Journal of Parallel Programming*, pages 1–19.

[Hong et al. 2010] Hong, S., Oguntebi, T., Casper, J., Bronson, N., Kozyrakis, C., and Olukotun, K. (2010). Eigenbench: A simple exploration tool for orthogonal tm characteristics. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–11.

Uso de *Threads* para o Planejamento de Segunda Ordem das Redes Geodésicas.

Vinicius Nonnenmacher, Ismael Érique Koch, Fabrício dos Reis Furtado, Rodrigo Righi, Luiz Gonzaga Jr.

[1]Programa de Pós-Graduação em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo, RS – Brasil

vnonnenmacher@gmail.com, isma.koch@gmail.com, fabricio14@gmail.com

rrrighi@unisinos.br, lgonzagajr@gmail.com

Resumo. *O presente trabalho demonstra um estudo feito na paralelização da meta-heurística da colônia artificial de abelhas (ABC) aplicada ao problema de planejamento de segunda ordem de uma rede geodésica planimétrica. A partir de um programa desenvolvido sequencialmente em python, foram realizados experimentos utilizando processamento paralelo de uma das fases do algoritmo.*

1. Introdução

O uso de técnicas para processamento de alto desempenho com fins práticos é utilizado em larga escala [Mattson et al. 2004]. Com o objetivo de verificar a extensão dos benefícios do uso de paralelismo ao algoritmo ABC aplicado ao planejamento de segunda ordem das redes geodésicas, este artigo descreve a implementação e análise do problema utilizando a paralelização de dados através de *threads*.

O planejamento de segunda ordem das redes geodésicas [Grafarend and Sansò 2012] é um problema conhecido na área da geodesia. As redes geodésicas são redes de triângulos medidos com exatidão podendo usar técnicas de levantamento terrestres ou geodesia espacial. Seus vértices são materializados em locais estratégicos a fim de obter melhor precisão e atender a finalidade desejada. Essas redes são utilizadas para serviços baseados em localização como mapeamento, geoinformação, registro de terras etc. Uma rede geodésica estabelecida deve ter alta precisão e confiabilidade. Para isso, no planejamento da mesma, deve-se estimar os melhores pesos para a matriz peso, onde podem ser aplicadas meta-heurísticas de minimização numérica.

O algoritmo Colônia Artificial de Abelhas (Artificial Bee Colony - ABC) foi implementado em *python* para geração dos valores da Matriz Peso a fim de minimizar $d^T * d$. O procedimento para cálculo segue abaixo, sendo P, a Matriz Peso a ter seus valores estimados pela ABC dentro do espaço de busca, e A e Q_X, a Matriz Design e a Matriz Criteria calculada com pesos originais da rede, respectivamente.

$$Q_{XM} = (A^T \cdot P \cdot A)$$
$$Q^{-1}_{R_{XM}} = Q^T_{XM} \cdot (Q_{XM} \cdot Q^T_{XM})^{-1}$$
$$D = Q^{-1}_{R_{XM}} - Q_X$$
$$d = vector(D)$$

$$d^T \cdot d \rightarrow \min$$

Onde, ^{T} é a transposta da respectiva matriz, $^{-1}$ indica a inversão da matriz e $vector$ representa a extração da diagonal de uma matriz e sua transposição em um vetor.

2. Implementação e Experimentos

O ABC é um algoritmo de busca que reproduz N ciclos otimizando a solução através de uma meta-heuristica. O *loop* principal do algoritmo executa diferentes fases, verificando e otimizando o *fitness* das soluções a cada iteração até um número pré-definido de ciclos. A função descrita no capítulo 1 é executada dentro da fase das abelhas trabalhadoras onde o objetivo é verificar e melhorar as soluções em um vetor de soluções.

Com o objetivo de melhorar o tempo de execução da fase das abelhas trabalhadoras, cada *thread* foi programada de forma a receber por parâmetro a posição inicial e final do intervalo o qual a *thread* deve processar. O vetor de soluções é acessado por cada *thread*, que computa e salva os resultados na mesma posição do vetor. Assim, a cada iteração do algoritmo, ao entrar na fase das abelhas trabalhadoras, o programa cria *N threads*, e processa paralelamente o vetor de soluções, salvando as novas soluções no mesmo. Foram realizados experimentos com diferentes números de *threads* e o tempo de execução foi mensurado por meio da função *time* do *python*.

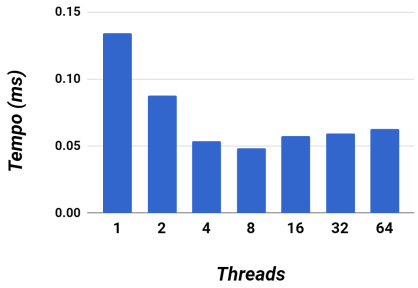


Figura 1. Tempo de execução da função por número de threads.

Os resultados podem ser verificados na Figura 1 que demonstra o tempo de execução da função com diferente número de *threads*.

3. Conclusão

Para otimizar o tempo de execução do algoritmo ABC aplicado as redes geodésicas de segunda ordem, este trabalho propôs a paralelização da fase das abelhas trabalhadoras distribuindo o processamento de um vetor de soluções em *N threads*. Os resultados demonstram que o tempo de execução do método foi reduzido até o uso de 8 *threads* gerando um *speed up* sublinear. Devido ao tempo perdido em comunicação e sincronismo, é possível verificar na prática que não compensa a utilização de mais que 8 *threads*.

Referências

Grafarend, E. and Sansò, F. (2012). *Optimization and Design of Geodetic Networks*. Springer Berlin Heidelberg.

Mattson, T., Sanders, B., and Massingill, B. (2004). *Patterns for Parallel Programming*. Software Patterns Series. Pearson Education.

18ª edição

ERAD 2018

Patrocínio

FAPERGS

TECHDEC
Sua Infraestrutura Inteligente e Segura

SDC

laniaq
scientific computing

ISSN 2177-0085