Introdução à programação paralela em Fortran usando OpenMP e MPI

Henrique Gavioli Flores,Alex Lima de Mello,
Marcelo Trindade Rebonatto

Universidade de Passo Fundo

5 de Abril de 2018

Sumário

Introdução

- FORmula TRANslation
- Uma das primeiras linguages de Alto Nível (HLL)
  - A de maior sucesso
  - Mais prática e com desempenho similar a Assembly
- Evolução dos computadores e das linguagens de programação
  - Processamento Paralelo
  - Programação Paralela
- OpenMP: interface de programação para multiprocessamento
- MPI: padrão de trocas de mensagens

Fortran

- Desenvolvida na IBM entre os anos de 1954 e 1957
  - John W. Bacus
  - + pesquisadores e desenvolvedores
- Adotada por cientistas para a escrita de programas com base matemática
- Sobrevive a mais de 60 anos
  - Diversas versões ao longo do tempo
  - Versões para processamento de Alto Desempenho
- Fortran é largamente utilizada até hoje.
  - Aplicações com grande necessidade de operações matemáticas
  - Alta complexidade
- Eta, DSSAT

Compiladores Fortran

- Utilizado para pesquisa científica
- Aplicações de alto desempenho
  - OpenMP,MPI,CUDA,OpenACC
- Alguns compiladores mais utilizados:
  - gfortran(GNU)
  - pgfortran(PGI)
  - ifort(Intel)
  - XL Fortran(IBM)
  - Cray(Cray)

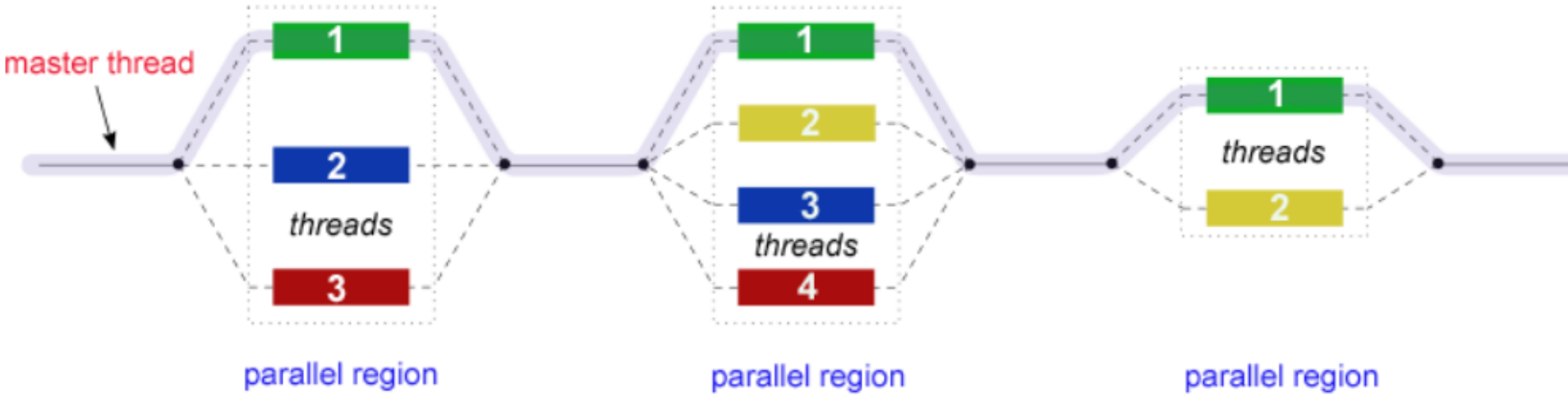
Compiladores Fortran com OpenMP e MPI

- OpenMP
  - Incluido por padrão
  - Necessário informar a flag durante a compilação do código
- MPI
  - Necessita que uma implementação do MPI seja instalada
  - Exemplo: MPICH2, OpenMPI
- Com SO Linux Ubuntu
  - libopenmpi-dev
  - openmpi-bin
  - gfortran

OpenMP

- API para programação paralela
- Memória compartilhada
- Proposta em 1997 na SuperComputing Conference
- Disponível pra C/C++ e Fortran
- Utiliza o modelo Fork/Join

OpenMP


master thread
1
2
threads
3
parallel region
1
2
3
threads
4
parallel region
1
threads
2
parallel region

Programação com OpenMP

- Compilando em Fortran com OpenMP

```
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

Programação com OpenMP

- Pragma
  - definido em Fortran por "!$"
  - compiladores não compatíveis tratam como comentário
  - serve para definir o início de uma diretiva OpenMP.

- Funções básicas
  - OMP_GET_NUM_THREADS()
    - Retorna o número de threads utilizadas em uma seção paralela
  - OMP_GET_MAX_THREADS()
    - Retorna a quantidade máxima de threads a serem usadas
  - OMP_SET_NUM_THREADS(nthreads)
    - Define a quantidade de threads a serem usadas pelas seções paralelas
  - OMP_GET_THREAD_NUM()
    - Retorna o número identificador(ID) da thread, variando de 0(thread master) até OMP_GET_NUM_THREADS() -1.

```
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)

- !$OMP PARALLEL (!$OMP END PARALLEL)
  - Marca um bloco a ser processado em paralelo
  - variáveis: Shared, private, firstprivate e lastprivate

```
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
```

Programação com OpenMP

- !$OMP MASTER (!$OMP END MASTER)
  - Delimita um bloco de código a ser executado apenas pela thread master(OMP_GET_THREAD_NUM() == 0)
- !$OMP SINGLE (!$OMP END SINGLE)
  - Delimita um bloco de código a ser executado apenas pela primeira thread que encontrar a diretiva, intependente de sua ID
- !$OMP CRITICAL (!$OMP END CRITICAL)
  - Indica que o código entre as diretivas será executado por uma thread de cada vez
  - pode ser indicado um nume $OMP CRITICAL (¡nome¿)
  - seções críticas sem nome são tratadas como se tivessem o mesmo nome

- !$OMP DO (!$OMP END DO)
  - usado para delimitar um trecho de código composto da iteração do loop DO.
  - Os indices do loop são divididos igualmente entre as threads.
  - A variável de controle do loop (step) é defida por padrão como PRIVATE.
  - Pode ser utilizado !$OMP PARALLEL DO (e o END equivalente) para iniciar uma seção paralela e o laço de repetição juntos.
- !$OMP SECTIONS (!$OMP END SECTIONS)
  - Utilizado para executar tarefas direrentes em paralelo, cada tarefa delimitada entre um par de diretivas !$OMP SECTION.

- Exemplo do uso de !$OMP DO e !$OMP CRITICAL

```
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)

(b)

```
$ gfortran -fopenmp pi.f90 -o pi
$ ./pi
 3.14144874
```

MPI

- Padrão para troca de mensagens
- Memória distribuída
- Proposta em 1992 em um workshop no Centro de Pesquisa em Computação Paralela de Williamsburg
- Utiliza troca de mensagens para comunicação entre processos

Programação com MPI

- Compilando em Fortran com MPI

```
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
```

- Conceitos MPI
  - Processos MPI: *rank*
  - Organizados em grupos
  - Dados e Envelope
    - dados = endereço de memória a ser enviado/recebido
    - envelope = identificação do processo enviado/recebido + o assunto da mensagem + comunicador associado.

<table>
<tr><th colspan="6">Mensagem</th></tr>
<tr><th colspan="3">Envelope</th><th colspan="3">dado</th></tr>
<tr><td>origem
destino</td><td>tag</td><td>comunicador</td><td>endereço</td><td>quantidade</td><td>tipo</td></tr>
</table>

Programação com MPI

- 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.

Programação com 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)
```

Enviando e recebendo mensagens

- MPI_RECV(recebe, 1, MPI_INTEGER, i, tag, MPI_COMM_WORLD, status, ierr)
- MPI_SEND(envia, 1, MPI_INTEGER,i, tag, MPI_COMM_WORLD, ierr)

- legenda:
  - envia/recebe = posição inicial do buffer
  - 1 = tamanho do buffer
  - MPI_INTEGER = tipo de dado a ser enviado
  - i = rank que irá receber/mandar a mensagem
  - tag = identificador da mensagem
  - MPI_COMM_WORLD = comunicador
  - ierr = var. auxiliar (sucesso/falha)

Enviando e recebendo mensagens - Exemplo

```
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)

(b)

```
$ mpif90 segundo.f90 -o segund
$ mpiexec -n 4 segundo
Received  123  from 1
Received  246  from 2
Received  369  from 3
```

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	

Usando coringas

- MPI_RECV(recebe, 1, MPI_INTEGER, **MPI_ANY_SOURCE**, **MPI_ANY_TAG**, MPI_COMM_WORLD, status, ierr)
- status(MPI_SOURCE)
- status(MPI_TAG)

- legenda:
  - MPI_ANY_SOURCE: pode ser qualquer origem
  - MPI_ANY_TAG: pode ser qualquer tag
  - status(MPI_SOURCE): identifica a orgigem da mensagem recebida
  - status(MPI_TAG) : identifica o assunto da mensagem recebida

Comunicando conjuntos de dados

- MPI_SEND(vet(ind), parte, MPI_INTEGER, MPI_ANY_SOURCE, tag, MPI_COMM_WORLD, ierr)
- MPI_RECV(vet, parte, MPI_INTEGER, i, tag, MPI_COMM_WORLD, status, ierr)

- legenda:
  - vet(ind)= posição inicial do vetor
  - parte = elementos a serem comunicados
  - MPI_INTEGER = tipo de dado a ser enviado
  - i = rank que irá receber/mandar a mensagem
  - tag = identificador da mensagem
  - MPI_COMM_WORLD = comunicador
  - ierr = var. auxiliar (sucesso/falha)
  - vet = posição de recebimento (ind = 0)

Comunicando conjuntos de dados - Exemplo

```
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)

Considerações Finais

- Uso de OpenMP e MPI em Fortan é simples e direto
- Programadores podem 'facilmente' explorar o paralelismo em seus programas
- Estudos adicionais
  - Operações de redução global em OpenMP e MPI
  - Comunicação de grupo e assincronas em MPI
  - ...