Paralelização
Introdução a vetorização, OpenMP e MPI

3 – OpenMP

Paulo Penteado
IAG / USP

pp.penteado@gmail.com

Esta apresentação: http://www.ppenteado.net/ast/pp_para_on/pp_para_on_3.pdf
Arquivos do curso: http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf

Programa

1 – Conceitos

- Motivação
- Formas de paralelização
  - Paralelismo de dados
  - Paralelismo de tarefas
- Principais arquiteturas paralelas atuais
  - Com recursos compartilhados
  - Independentes
- Paradigmas discutidos neste curso:
  - Vetorização
  - OpenMP
  - MPI
- Escolhas de forma de vetorização
- Algumas referências
- Exercícios – testes de software a usar no curso

Slides em http://www.ppenteado.net/ast/pp_para_on_1.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

2 – Vetorização

- Motivação
- Arrays – conceitos
- Organização multidimensional
- Arrays – uso básico
- Arrays – row major x column major
- Operações vetoriais
- Vetorização avançada
  - Operações multidimensionais
  - Redimensionamento
  - Buscas
  - Inversão de índices
- Algumas referências
- Exercícios - vetorização

Slides em http://www.ppenteado.net/ast/pp_para_on_2.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

3 – OpenMP

- Motivação
- Características
  - Diretrizes
  - Estruturação
- Construções
  - parallel
  - loop
  - section
  - workshare
- Cláusulas
  - Acesso a dados
  - Controle de execução
- Sincronização
  - Condições de corrida
- Exercícios - OpenMP

Slides em http://www.ppenteado.net/ast/pp_para_on_3.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

4 – MPI

- Motivação
- Características
- Estruturação
- Formas de comunicação
- Principais funções
  - Controle
  - Informações
  - Comunicação
- Boost.MPI
- Sincronização
  - Deadlocks
- Exercícios - MPI

Slides em http://www.ppenteado.net/ast/pp_para_on_4.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf

pp.penteado@gmail.com

OpenMP - Open Multi-Processing (OMP)

É o padrão melhor estabelecido para programação em paralelo com **memória compartilhada (múltiplos *threads*).**

Antes do OpenMP:

- Não havia um padrão: quem escrevia cada compilador inventava sua forma de criar e usar threads.
- Para mudar de linguagem, sistema operacional, computador, ou simplesmente compilador, era necessário aprender uma nova forma, e reimplementar o código de acordo.

OpenMP:

- Engloba uma biblioteca que provê a funcionalidade, e uma API (*Application Programming Interface*) que especifica a forma de a usar.
- **Independente de linguagem, compilador, sistema operacional e hardware.**
- Padrão aberto, e em contínua evolução, decidido pelo *Architecture Review Board* (ARB), que inclui os principais desenvolvedores de hardware e software.
- 1.0 em 1997, 2.0 em 2002, 3.0 em 2008, 3.1 previsto para 2011.

OpenMP - Características

Paralelismo apenas por memória compartilhada:

- **Cada unidade de execução é um thread.**
- Todos os threads têm acesso a uma mesma memória (global, pública).
- Cada thread tem também sua memória independente dos outros (local, privada), para seu uso interno.
- Geralmente muito mais simples de programar que com memória distribuída (ex: MPI).

Com mais freqüência é usado para paralelismo de dados (e tem maior variedade de ferramentas para isso), mas também faz paralelismo de tarefas.

Tradicionalmente suportado por compiladores de C, C++ e Fortran:

- A funcionalidade e a semântica (significado das construções) são as mesmas, dentro dos limites de cada linguagem.
- A sintaxe (forma de expressar) é semelhante (não é idêntica por as linguagens terem sintaxes diferentes).

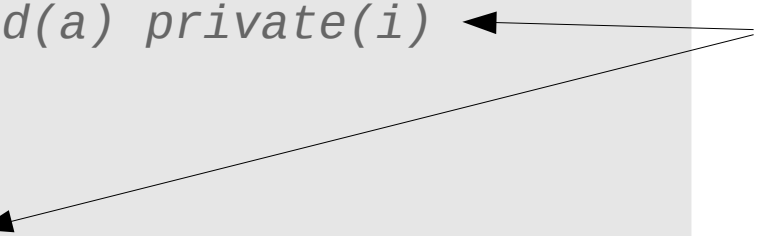
OpenMP - Características

Implementado através de **diretrizes de compilação** (instruções que serão executadas pelo compilador, não pelo programa produzido):

- **Diretrizes são vistas como comentários** por compiladores que não conheçam OMP (ou quando OMP é desabilitado na hora de compilar).
- Mantém o código funcional como programa serial.
- A paralelização de um código serial pode ser feita gradualmente: **não é necessário reestruturar o programa apenas para incluir os usos de OMP** (mas pode ser necessário reestruturar para o tornar paralelizável).

Ex (Fortran):

```
program pp_omp_ex1
!Generic example of the look of an OMP program
implicit none
integer, parameter :: n=1000
integer :: i,a(n)
!$omp parallel do shared(a) private(i)
do i=1,n
  a(i)=i
enddo
!$omp end parallel do
end program pp_omp_ex1
```



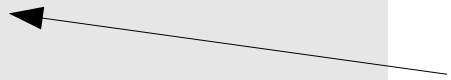
OpenMP - Características

Implementado através de **diretrizes de compilação** (instruções que serão executadas pelo compilador, não pelo programa produzido):

- **Diretrizes são vistas como comentários** por compiladores que não conheçam OMP (ou quando OMP é desabilitado na hora de compilar).
- Mantém o código funcional como programa serial.
- A paralelização de um código serial pode ser feita gradualmente: **não é necessário reestruturar o programa apenas para incluir os usos de OMP** (mas pode ser necessário reestruturar para o tornar paralelizável).

Ex (C/C++):

```
int main(int argc, char *argv[]) {
/*Generic example of the look of an OMP program*/
int n=1000;
int a[n];
#pragma omp parallel for shared(a,n)
for (int i=0; i<n; i++) {
  a[i]=i;
}
return 0;
}
```



OpenMP - Características

Estrutura típica de um programa OMP (Chapman et al. 2007):

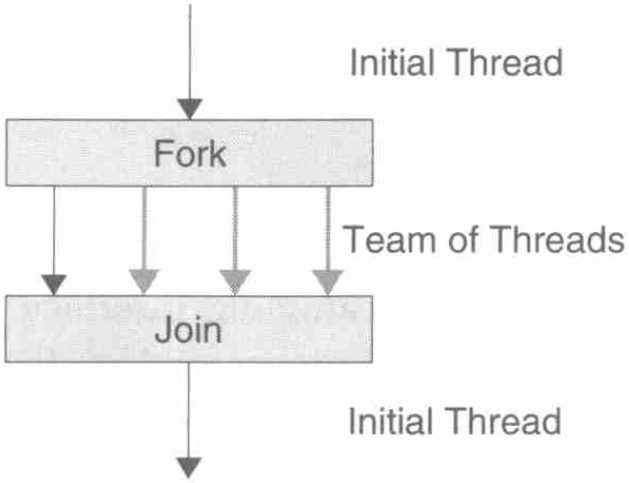


Figure 2.1: **The fork-join programming model supported by OpenMP** – The program starts as a single thread of execution, the initial thread. A team of threads is forked at the beginning of a parallel region and joined at the end.

O thread master (inicial, 0) é onde a execução do programa se inicia e termina.

O master em algum ponto cria um time (conjunto) de outros threads, que vão executar algo em paralelo: ***fork***.

Quando todos os threads de um time chegam ao final da região paralela, todos menos o master terminam: **join**.

É possível haver várias regiões paralelas, com seriais entre elas.

OpenMP - Características

Em alguns casos, é possível também ter vários níveis de paralelização (*nesting*):

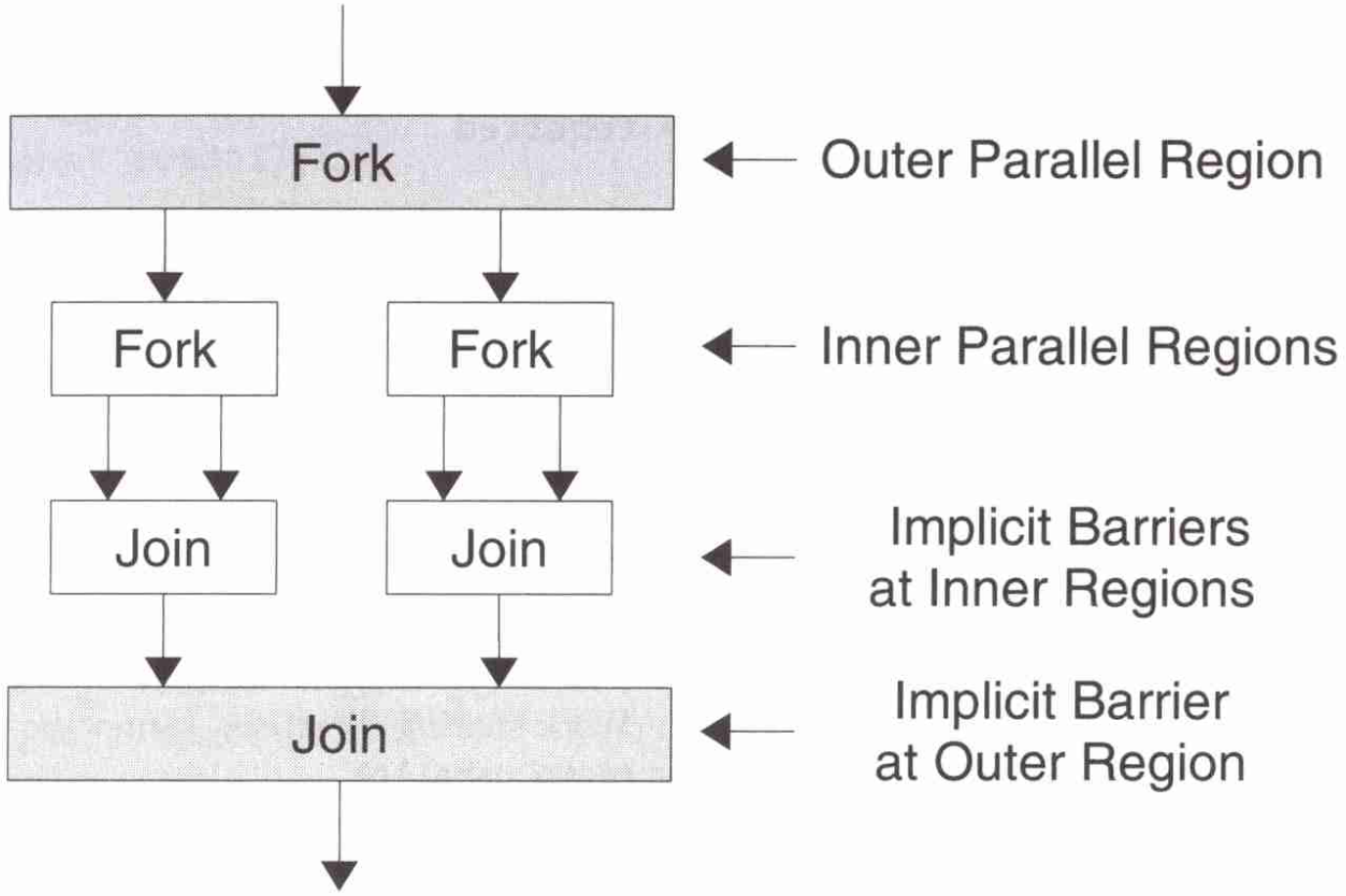


Figure 6.20: **Nested OpenMP parallel regions** – Inner parallel regions introduce implicit synchronization points in addition to the barrier synchronization points at the outer parallel regions.

(Chapman et al. 2007)

OpenMP - Construções

Trechos de código são paralelizados em OMP por **construções** (*constructs*).

Cada construção é especificada por diretrizes OMP (“comentários”), que delimitam a região de código a ser paralelizada, e como ela será paralelizada.

Fora das construções, o código é serial, inalterado pelo OMP.

Construções

- Base para todas as outras, especificando as seções paralelas:
  - `parallel`

- Dividem dados/tarefas entre os threads (*worksharing constructs*):
  - `loop`
  - `section`
  - `workshare` (só em Fortran)
  - `single`
  - `task` (só a partir do 3.0)

- Controlam sincronização (de dados e outros recursos):
  - `barrier`
  - `master`
  - `critical`
  - `atomic`
  - `locks`

OpenMP - `parallel`

Delimita uma região paralela.

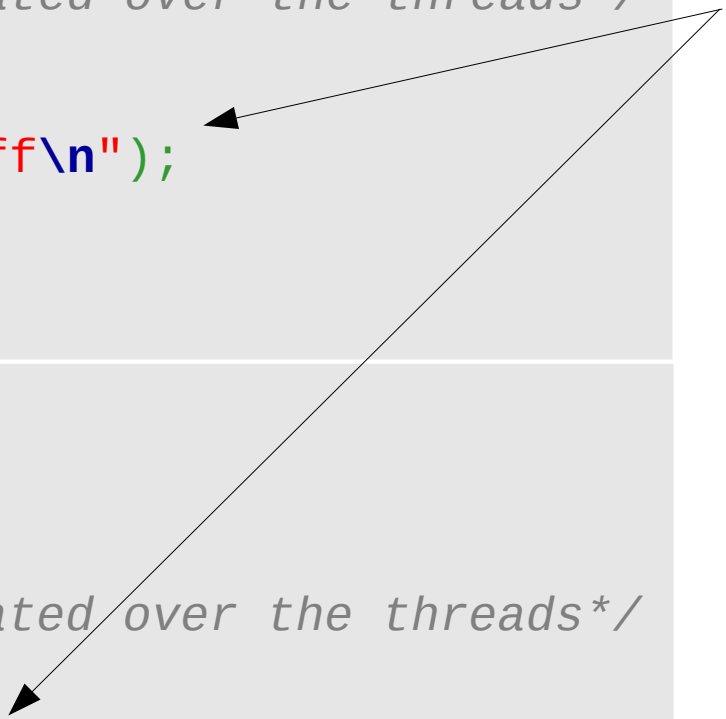
Sozinha, apenas especifica que a região deve ser executada por todos os threads, sem especificar o que será diferente de um thread para outro. Ex:

C

```c
#include <stdio.h>
int main() {
/*Example of work replicated over the threads*/
#pragma omp parallel
{
  printf("Doing some stuff\n");
}
return 0;
}
```

C++

```cpp
#include <iostream>
int main() {
/*Example of work replicated over the threads*/
#pragma omp parallel
{
  std::cout<<"Doing some stuff"<<std::endl;
}
}
```



Bloco paralelo: executado em **OMP_NUM_THREADS** (variável de sistema) threads simultaneamente.

OpenMP - `parallel`

Delimita uma região paralela.

Sozinha, apenas especifica que a região deve ser executada por todos os threads, sem especificar o que será diferente de um thread para outro. Ex:

Fortran

```
program pp_omp_ex2
!Example of work replicated over the threads
Implicit none
!$omp parallel
write (6,*) 'Doing some stuff'
!$omp end parallel
end program pp_omp_ex2
```

Região paralela: executada em **OMP_NUM_THREADS** (variável de sistema) threads simultaneamente.

Quando executado, este programa apenas replica o trabalho, idêntico:

```
[user@computer dir]$ export OMP_NUM_THREADS=1

[user@computer dir]$ ./pp_omp_ex2.out
Doing some stuff

[user@computer dir]$ export OMP_NUM_THREADS=2

[user@computer dir]$ ./pp_omp_ex2.out
Doing some stuff
Doing some stuff
```

OpenMP - `parallel`

Como executar algo útil (diferente entre threads) só com `parallel`?

Fazendo uso do número do thread. Ex:

Fortran

```
program pp_omp_ex3
!Example of how to use thread numbers
!$ use omp_lib
implicit none
integer :: tid
!$omp parallel
tid=0 !Dummy thread number for the case OMP is not enabled
!$ tid=omp_get_thread_num()
write (6,*) 'Doing some stuff in thread ',tid
!$omp end parallel
end program pp_omp_ex3
```

Sentinelas (*sentinels*):

Linhas iniciadas com **`!$`** só são compiladas por compiladores que sabem OMP, se OMP estiver habilitado. Caso contrário, são só comentários, preservando o código serial.

A função **`omp_get_thread_num()`**, parte da biblioteca OMP, retorna o número do thread sendo executado. Pode ser usado para a decisão de o que cada thread faz.

Paralelização em OMP usando o índice do thread é semelhante a paralelização com CUDA (GPUs Nvidia).

OpenMP - `parallel`

Como executar algo útil (diferente entre threads) só com `parallel`?

Fazendo uso do número do thread. Ex:

Diretrizes para compilação condicional:

Só com compiladores que sabem OMP, se OMP estiver habilitado, _OPENMP é definido. Caso contrário, estes trechos são ignorados, deixando apenas o código serial.

C++

```
#include <iostream>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of how to use thread numbers*/
#pragma omp parallel
{
  int tid=0; /*Dummy thread number for the case OMP is not enabled*/
  #ifdef _OPENMP
    tid=omp_get_thread_num();
  #endif
  std::cout << "Doing some stuff in thread "
   << tid << std::endl;
}
}
```

OpenMP - `parallel`

Como executar algo útil (diferente entre threads) só com `parallel`?

Fazendo uso do número do thread. Ex:

C

```c
#include <stdio.h>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of how to use thread numbers*/
#pragma omp parallel
{
  int tid=0; /*Dummy thread number for the case OMP is not enabled*/
  #ifdef _OPENMP
    tid=omp_get_thread_num();
  #endif
  printf("Doing some stuff"
   " in thread %i\n",tid);
}
return 0;
}
```

Diretrizes para compilação condicional:

Só com compiladores que sabem OMP, se OMP estiver habilitado, _OPENMP é definido. Caso contrário, estes trechos são ignorados, deixando apenas o código serial.

OpenMP - `parallel`

Neste exemplo, ao executar o programa compilado sem OMP:

```
[user@computer dir]$ g++ pp_omp_ex3.cpp -o pp_omp_ex3.out

[user@computer dir]$ ./pp_omp_ex3.out
Doing some stuff in thread 0
```

Com OMP:

```
[user@computer dir]$ g++ pp_omp_ex3.cpp -o pp_omp_ex3.out -fopenmp

[user@computer dir]$ export OMP_NUM_THREADS=1

[user@computer dir]$ ./pp_omp_ex3.out
Doing some stuff in thread 0

[user@computer dir]$ export OMP_NUM_THREADS=2

[user@computer dir]$ ./pp_omp_ex3.out
Doing some stuff in thread 1
Doing some stuff in thread 0
```

A ordem da saída é aleatória: depende de que thread por acaso termine primeiro.

OpenMP – *worksharing constructs*

Apenas `parallel` provê controle de muito baixo nível sobre os threads.

As construções de divisão de trabalho (*worksharing*) são formas prontas de paralelizar alguns tipos de tarefas:

- `loop` (**do** em Fortran, **`for`** em C/C++):
  Cada thread executa uma fração das iterações do loop.

- `section`
  Cada `section` specifica uma tarefa a ser executada por um thread.

- `workshare` (só em Fortran)
  Especifica que as operações vetoriais contidas na seção devem ser distribuídas entre os threads.

- `single`
  Especifica que uma seção do trecho paralelizado deve ser executado por apenas um dos threads (importante para problemas de sincronia ou acesso a recursos).

- `task` (só a partir do OMP 3.0)
  Semelhante a `section`, cada uma especifica uma tarefa a ser dada a um thread, mas mais elaboradas, incluindo criação dinâmica de tasks.

OpenMP – loop

do em Fortran, **`for`** em C/C++

Cada thread executa uma fração das iterações do loop. Ex (Fortran):

```
program pp_omp_ex4
!Example of a parallel loop
!$ use omp_lib
implicit none
integer,parameter :: n=7
integer :: i,a(n),t(n),tid
!$omp parallel shared(a,t) private(i,tid)
tid=0 !Dummy thread number, for the case OMP is not enabled
!$ tid=omp_get_thread_num()
!$omp do
do i=1,n
  a(i)=2*i
  t(i)=tid !Record which thread did which iteration
enddo
!$omp end do
!$omp end parallel
!Show what was produced
write(6,*) 'i,a(i),thread number'
do i=1,n
  write(6,*) i,a(i),t(i)
enddo
end program pp_omp_ex4
```

OpenMP – loop (C++)

```
#include <iostream>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of a parallel loop*/
int n=7;
int a[n],t[n];
#pragma omp parallel shared(a,t,n)
{
  int tid=0; /*Dummy thread number for the case OMP is not enabled*/
  #ifdef _OPENMP
    tid=omp_get_thread_num();
  #endif
  #pragma omp for
  for (int i=0; i<n; i++) {
    a[i]=2*i;
    t[i]=tid; /*Record which thread did which iteration*/
  }
}
/*Show what was produced*/
std::cout << "i,a(i),thread number" << std::endl;
for (int i=0; i<n; i++) {
  std::cout << i << " " << a[i] << " " << t[i] << std::endl;
}
}
```

OpenMP – loop (C)

```c
#include <stdio.h>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of a parallel loop*/
int n=7;
int a[n],t[n];
#pragma omp parallel shared(a,t,n)
{
  int tid=0; /*Dummy thread number for the case OMP is not enabled*/
  #ifdef _OPENMP
    tid=omp_get_thread_num();
  #endif
  #pragma omp for
  for (int i=0; i<n; i++) {
    a[i]=2*i;
    t[i]=tid; /*Record which thread did which iteration*/
  }
}
/*Show what was produced*/
printf("i,a(i),thread number\n");
for (int i=0; i<n; i++) {
  printf("%i %i %i \n",i,a[i],t[i]);
}
return 0;
}
```

OpenMP – loop

Um resultado possível deste exemplo, quando executado:

```
[user@computer dir]$ gfortran pp_omp_ex4.f90 -o pp_omp_ex4.out

[user@computer dir]$ ./pp_omp_ex4.out
 i,a(i),thread number
           1           2           0
           2           4           0
           3           6           0
           4           8           0
           5          10           0
           6          12           0
           7          14           0

[user@computer dir]$ gfortran pp_omp_ex4.f90 -o pp_omp_ex4.out -fopenmp

[user@computer dir]$ export OMP_NUM_THREADS=2

[user@computer dir]$ ./pp_omp_ex4.out
 i,a(i),thread number
           1           2           0
           2           4           0
           3           6           0
           4           8           0
           5          10           1
           6          12           1
           7          14           1
```

OpenMP – section

Cada `section` é executada por um thread (paralelismo de tarefas). Ex (Fortran):

```
program pp_omp_ex5
!Example of sections
!$ use omp_lib
implicit none
integer,parameter :: n=7
integer :: i,a(n),ta(n),b(n),tb(n),tid
!$omp parallel shared(a,ta,b,tb) private(i,tid)
!$omp sections
!$omp section
 tid=0 !Dummy thread number, for the case OMP is not enabled
 !$ tid=omp_get_thread_num()
 do i=1,n
  a(i)=2*i
  ta(i)=tid !Record which thread did which iteration
 enddo
!$omp section
 tid=0 !Dummy thread number, for the case OMP is not enabled
 !$ tid=omp_get_thread_num()
 do i=1,n
  b(i)=3*i
  tb(i)=tid !Record which thread did which iteration
 enddo
!$omp end sections
!$omp end parallel
!Show what was produced
write(6,*) 'i,a(i),thread that made a(i),b(i), thread that made b(i)'
do i=1,n
 write(6,*) i,a(i),ta(i),b(i),tb(i)
enddo
end program pp_omp_ex5
```

OpenMP – section

Cada `section` é executada por um thread (paralelismo de tarefas). Ex (C++):

```
#include <iostream>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of sections*/
int n=7;
int a[n],ta[n],b[n],tb[n];
```

(continua na próxima página)

```
#pragma omp parallel shared(a,ta,b,tb,n)
{
  #pragma omp sections
  {
    #pragma omp section
    {
      int tid=0; /*Dummy thread number for the case OMP is not enabled*/
      #ifdef _OPENMP
        tid=omp_get_thread_num();
      #endif
      for (int i=0; i<n; i++) {
        a[i]=2*i;
        ta[i]=tid; /*Record which thread did which iteration*/
      }
    }
    #pragma omp section
    {
      int tid=0; /*Dummy thread number for the case OMP is not enabled*/
      #ifdef _OPENMP
        tid=omp_get_thread_num();
      #endif
      for (int i=0; i<n; i++) {
        b[i]=3*i;
        tb[i]=tid; /*Record which thread did which iteration*/
      }
    }
  }
}
/*Show what was produced*/
std::cout<<"i,a(i),thread that made a(i),b(i), thread that made b(i)"<<std::endl;
for (int i=0; i<n; i++) {
  std::cout<<i<<" "<<a[i]<<" "<<ta[i]<<" "<<b[i]<<" "<<tb[i]<<std::endl;
}
}
```

OpenMP – section

Cada `section` é executada por um thread (paralelismo de tarefas). Ex (C):

```
#include <stdio.h>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of sections*/
int n=7;
int a[n],ta[n],b[n],tb[n];
```

(continua na próxima página)

```
#pragma omp parallel shared(a,ta,b,tb,n)
{
  #pragma omp sections
  {
    #pragma omp section
    {
      int tid=0; /*Dummy thread number for the case OMP is not enabled*/
      #ifdef _OPENMP
        tid=omp_get_thread_num();
      #endif
      for (int i=0; i<n; i++) {
        a[i]=2*i;
        ta[i]=tid; /*Record which thread did which iteration*/
      }
    }
    #pragma omp section
    {
      int tid=0; /*Dummy thread number for the case OMP is not enabled*/
      #ifdef _OPENMP
        tid=omp_get_thread_num();
      #endif
      for (int i=0; i<n; i++) {
        b[i]=3*i;
        tb[i]=tid; /*Record which thread did which iteration*/
      }
    }
  }
}
/*Show what was produced*/
printf("i,a(i),thread that made a(i),b(i), thread that made b(i)\n");
for (int i=0; i<n; i++) printf("%i %i %i %i %i\n",i,a[i],ta[i],b[i],tb[i]);
return 0;
}
```

OpenMP – workshare

Especifica que as operações vetoriais devem ser executadas em paralelo, com os elementos divididos entre os threads.

Apenas para Fortran.

Ex:

```
program pp_omp_ex6
!Example of workshare
!$ use omp_lib
implicit none
integer,parameter :: n=7
integer :: i,j,a(n),b(n),c(n)
!$omp parallel shared(a,b,c) private(i,j)
!$omp workshare
forall (i=1:n) a(i)=i
forall (j=1:n) b(j)=2*j
!OMP makes sure that all results are available before the next line
c(1:n)=a(1:n)+b(1:n)
!$omp end workshare
!$omp end parallel
end program pp_omp_ex6
```

OpenMP – cláusulas

Especificam ou modificam aspectos do funcionamento das construções:

Acesso a dados:

- `shared` - Especifica variáveis a serem compartilhadas entre todos os threads.
- `private` - Especifica variáveis a serem privadas (locais): uma cópia existe para cada thread.
- `lastprivate` - Como `private`, mas mantém o último valor local após a seção paralela.
- `Firstprivate` - Como `private`, mas inicializa a variável com o valor que tinha antes da seção paralela.
- `default` - Determina se por default uma variável é `shared`, `private` ou `none`.

Controle de execução:

- `nowait` - Suprime barreiras implícitas (como ao final de loops).
- `schedule` - Determina como iterações de loops são divididas entre threads.

OpenMP – sincronização

Em algumas situações, é necessário controlar o acesso ou a ordem de operações entre threads. Ex. (Fortran):

```
program pp_omp_ex7
!Example of a race condition
!$ use omp_lib
implicit none
integer :: i,n,a
n=1000000
a=0
!$omp parallel shared(a,n) private(i)
!$omp do
do i=1,n
  !When one thread writes to a, the value of a
  !it had read might not be current anymore
  a=a+i
enddo
!$omp end do
!$omp end parallel
!Show what was produced
write(6,*) 'a=',a
end program pp_omp_ex7
```

OpenMP – sincronização

Em algumas situações, é necessário controlar o acesso ou a ordem de operações entre threads. Ex. (C++):

```
#include <iostream>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of a race condition*/
int n=1000000,a=0;
#pragma omp parallel shared(a,n)
{
  #pragma omp for
  for (int i=0; i<n; i++) {
    /*When one thread writes to a, the
value of a
    it had read might not be current
anymore*/
    a+=i;
  }
}
/*Show what was produced*/
std::cout << "a=" << a << std::endl;
}
```

OpenMP – sincronização

Em algumas situações, é necessário controlar o acesso ou a ordem de operações entre threads. Ex. (C):

```c
#include <stdio.h>
#ifdef _OPENMP
  #include<omp.h>
#endif
int main() {
/*Example of a race condition*/
int n=1000000,a=0;
#pragma omp parallel shared(a,n)
{
  #pragma omp for
  for (int i=0; i<n; i++) {
    /*When one thread writes to a, the value of a
    it had read might not be current anymore*/
    a+=i;
  }
}
/*Show what was produced*/
printf("a= %i\n",a);
return 0;
}
```

OpenMP – sincronização

Condição de corrida (*race condition*)
O código do exemplo anterior tem comportamento indeterminado.
O resultado depende de que thread por acaso chega antes ao trecho que altera a variável compartilhada entre threads.
Exemplos de resultados de o executar:

```
[user@computer dir]$ gcc -std=c99 pp_omp_ex7.c -o pp_omp_ex7.out

[user@computer dir]$ ./pp_omp_ex7.out
 A=  1784293664

[user@computer dir]$ gcc -std=c99 pp_omp_ex7.c -o pp_omp_ex7.out -fopenmp

[user@computer dir]$ export OMP_NUM_THREADS=20

[user@computer dir]$ ./pp_omp_ex7.out
 a=  1340951740

[user@computer dir]$ ./pp_omp_ex7.out
 a=   482130990

[user@computer dir]$ ./pp_omp_ex7.out
 a=  1846303706

[user@computer dir]$ ./pp_omp_ex7.out
 a=  1784293664

[user@computer dir]$ ./pp_omp_ex7.out
 a=   822224612
```

Exercícios – vetorização, OpenMP

Escrever uma função para multiplicação de matrizes.

Exemplo artificial: toda linguagem tem uma função (ou operador) para isso em alguma biblioteca (padrão ou não) .

Equivalente a (ex. Fortran):

```
module pp_para_on_exe_1
implicit none
interface pp_para_on_exe_1_f
  module procedure pp_para_on_exe_1_ff
end interface
contains
function pp_para_on_exe_1_ff(mat1,mat2) result(mat12)
implicit none
integer :: i,j,k,sz1(2),sz2(2)
double precision :: mat1(:,:),mat2(:,:)
double precision,allocatable :: mat12(:,:)
sz1=(/size(mat1,dim=1),size(mat1,dim=2)/)
sz2=(/size(mat2,dim=1),size(mat2,dim=2)/)
if (sz1(2) .ne. sz2(1)) then
  print *,'Second dimension of second array must match first dimension
of first array'
  return
endif
```

Exercícios – vetorização, OpenMP

(continuação)

```
allocate(mat12(sz1(1),sz2(2)))
do i=1,sz2(2)
  do j=1,sz1(1)
    do k=1,sz2(1)
      mat12(j,i)=mat12(j,i)+mat1(j,k)*mat2(k,i)
    enddo
  enddo
enddo
return
end function pp_para_on_exe_1_ff
```

Uma solução em http://www.ppenteado.net/ast/pp_para_on/pp_para_on_sol_4.pdf

OpenMP – sincronização

Construções para evitar condições de corrida, ou controlar a execução de um trecho do código:

- `barrier` – barreira explícita: execução só continua quando todos os threads chegam à barreira.
- `ordered` – força que um trecho seja executado pelos threads na ordem do loop.
- `master` – código a ser executado só pelo thread master (como semelhante a `single`, mas sem barreiras implícitas).
- `critical` – especifica um trecho que pode ser executado por apenas um thread de cada vez.
- `atomic` – Alternativa mais eficiente (mas mais restrita) a `critical`, especifica uma operação atômica, para impedir que múltiplos threads a executem ao mesmo tempo.
- `locks` – Semelhante a semáforos (semaphores), define uma variável de trava (`lock`): apenas um thread pode estar com a trava de cada vez; os outros têm que esperar até que ela seja liberada. Pode provocar *deadlocks* (adiante).

Algumas referências - OpenMP

- Site oficial
  http://openmp.org/
- *OpenMP Forum*
  http://openmp.org/forum/
- The Community of OpenMP Users, Researchers, Tool Developers and Providers
  http://www.compunity.org/
- *Using OpenMP: Portable Shared Memory Parallel Programming* (2007, só até OMP 2.5)
  Chapman, Jost, van der Pas
  http://www.amazon.com/Using-OpenMP-Programming-Engineering-Computation/dp/0262533022/
- *C++ and OpenMP*
  http://www.compunity.org/events/pastevents/parco07/parco_cpp_openmp.pdf
- *OpenMP C and C++ Application Program Interface*
  http://www.openmp.org/mp-documents/cspec20.pdf
- *Parallel Program in Fortran 95 Using OpenMP*
  http://www.openmp.org/presentations/miguel/F95_OpenMPv1_v2.pdf
- *Complete Specifications - (May, 2008)*
  http://www.openmp.org/mp-documents/spec30.pdf
- *Version 3.0 Summary Card C/C++*
  http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf
- *Version 3.0 Summary Card Fortran*
  http://www.openmp.org/mp-documents/OpenMP3.0-FortranCard.pdf

Algumas referências

Gerais (incluindo OpenMP e MPI e outros)

- *Parallel Programming: for Multicore and Cluster Systems* (2010)
  Rauber e Rünger
  http://www.amazon.com/Parallel-Programming-Multicore-Cluster-Systems/dp/364204817X/

- *An Introduction to Parallel Programming* (2011)
  Peter Pacheco
  http://www.amazon.com/Introduction-Parallel-Programming-Peter-Pacheco/dp/0123742609/

- *An Introduction to Parallel Programming with OpenMP, PThreads and MPI* (2011)
  Robert Cook
  http://www.amazon.com/Introduction-Parallel-Programming-PThreads-ebook/dp/B004I6D6BM/

- *Introduction to High Performance Computing for Scientists and Engineers* (2010)
  Hager e Wellein
  http://www.amazon.com/Introduction-Performance-Computing-Scientists-Computational/dp/143981192X/

Programa

1 – Conceitos

- Motivação
- Formas de paralelização
  - Paralelismo de dados
  - Paralelismo de tarefas
- Principais arquiteturas paralelas atuais
  - Com recursos compartilhados
  - Independentes
- Paradigmas discutidos neste curso:
  - Vetorização
  - OpenMP
  - MPI
- Escolhas de forma de vetorização
- Algumas referências
- Exercícios – testes de software a usar no curso

Slides em http://www.ppenteado.net/ast/pp_para_on_1.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

2 – Vetorização

- Motivação
- Arrays – conceitos
- Organização multidimensional
- Arrays – uso básico
- Arrays – row major x column major
- Operações vetoriais
- Vetorização avançada
  - Operações multidimensionais
  - Redimensionamento
  - Buscas
  - Inversão de índices
- Algumas referências
- Exercícios - vetorização

Slides em http://www.ppenteado.net/ast/pp_para_on_2.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

3 – OpenMP

- Motivação
- Características
  - Diretrizes
  - Estruturação
- Construções
  - parallel
  - loop
  - section
  - workshare
- Cláusulas
  - Acesso a dados
  - Controle de execução
- Sincronização
  - Condições de corrida
- Exercícios - OpenMP

Slides em http://www.ppenteado.net/ast/pp_para_on_3.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf
pp.penteado@gmail.com

Programa

4 – MPI

- Motivação
- Características
- Estruturação
- Formas de comunicação
- Principais funções
  - Controle
  - Informações
  - Comunicação
- Boost.MPI
- Sincronização
  - Deadlocks
- Exercícios - MPI

Slides em http://www.ppenteado.net/ast/pp_para_on_4.pdf

Exemplos em http://www.ppenteado.net/ast/pp_para_on/
Artigo relacionado: http://www.ppenteado.net/papers/iwcca/iwcca_pfp.pdf

pp.penteado@gmail.com