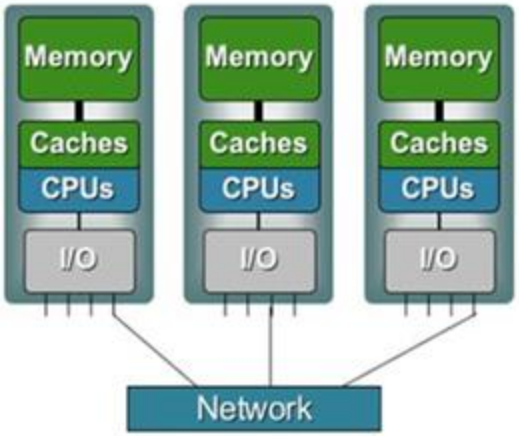
Programação Paralela com Memória Compartilhada: OpenMP

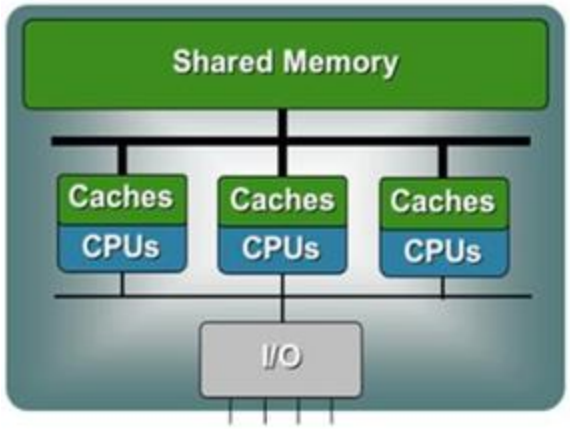
Prof. Marcelo Veiga Neves

marcelo.neves@pucrs.br

Modelos de Programação Paralela



Memória Distribuída (*Cluster*)



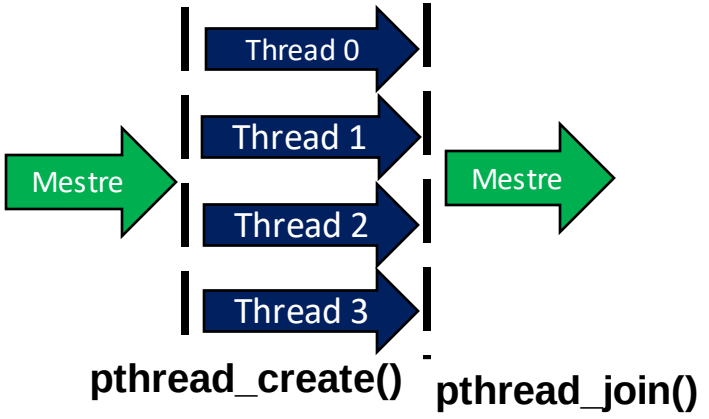
Memória Compartilhada (*Multicore, Maycore*)

Interfaces de Programação Paralela

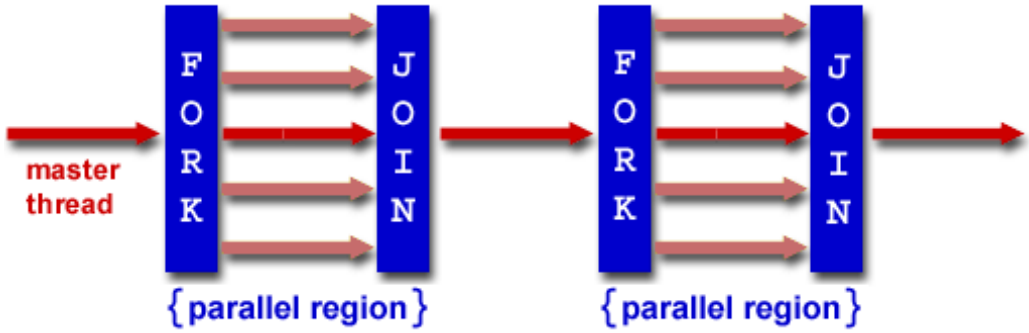
- Memória Compartilhada
  - Baseado em pragmas e diretivas:
    - OpenMP
    - Cilk++
  - Baseado em funções:
    - POSIX Threads
    - Intel TBB

POSIX Threads

```
#include <pthreads.h>


void printHello(){
  printf("Paralelo %d\n", idThread);
}

int main(){
 printf("Sequencial\n");
 pthread_t threads[numThreads];

 for(int i=0;i<numThreads; i++)
   pthread_create(...);
 for(int i=0;i<numThreads; i++)
   pthread_join(...);

 printf("Sequencial\n");
 return 0;
}
```

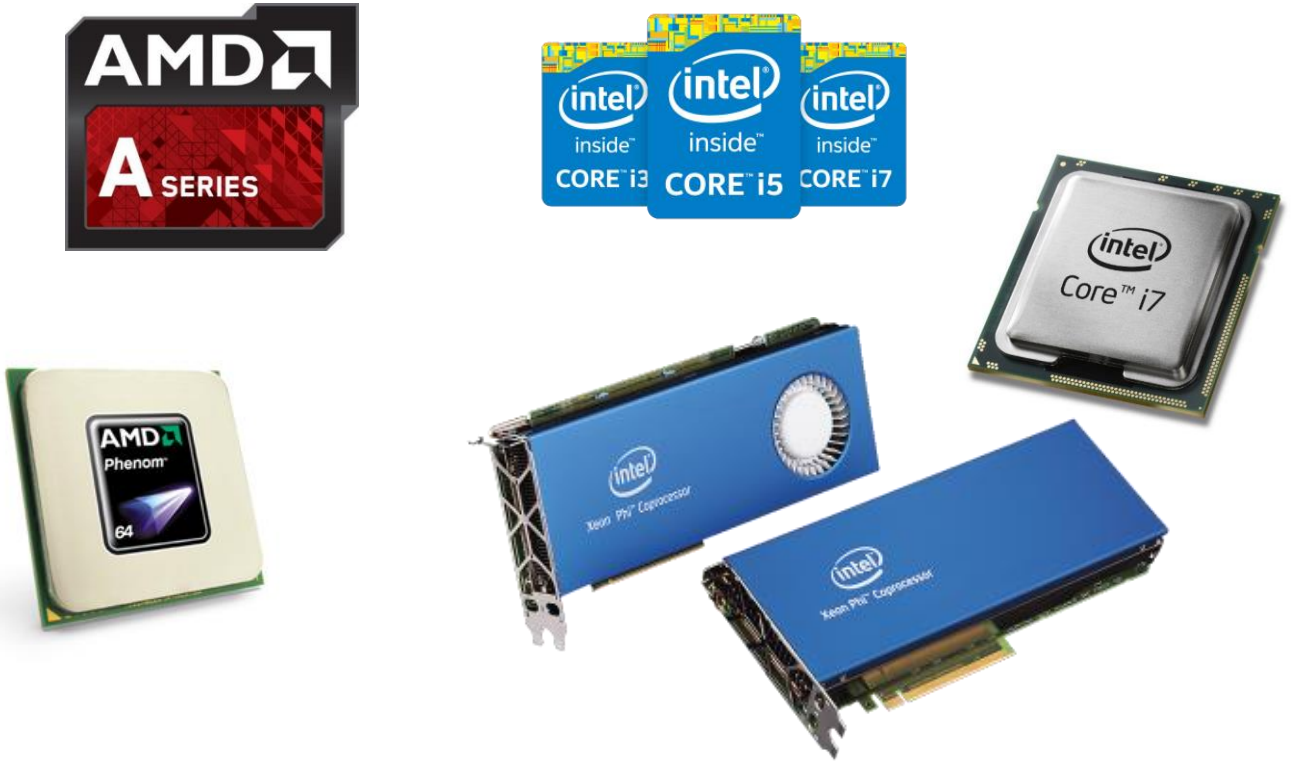


OpenMP



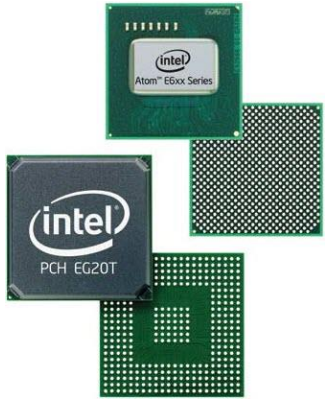

```
#include <omp.h>
int main(){
 printf("Sequencial\n");

 #pragma omp parallel
 {
   printf("Paralelo %d\n", idThread);
 }

 printf("Sequencial\n");
 return 0;
}
```

Onde utilizar POSIX Threads e OpenMP?



Onde utilizar POSIX Threads e OpenMP?

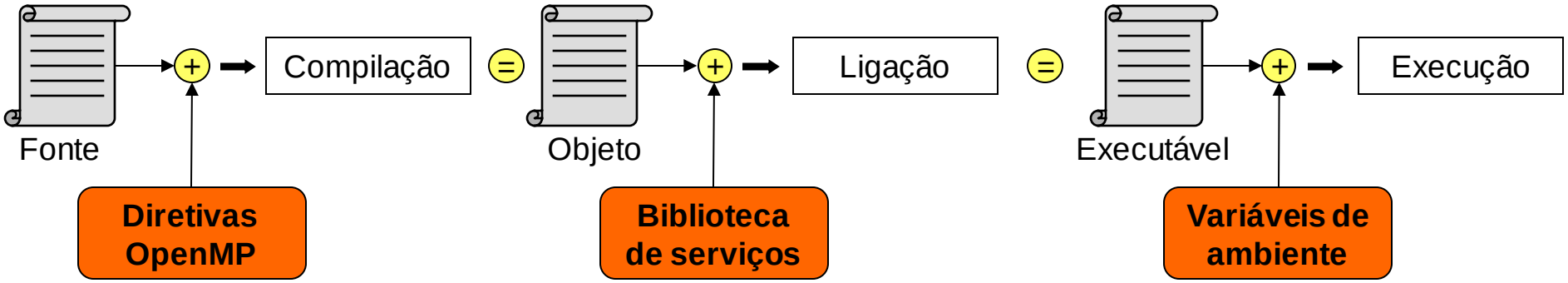
Onde utilizar POSIX Threads e OpenMP?



Onde utilizar POSIX Threads e OpenMP?

OpenMP

- Interface de programação (API) para desenvolvimento de aplicações multithread em C/C++ e Fortran
  - Define
    - Diretivas de compilação
    - Biblioteca de funções/serviços
    - Variáveis de ambiente



OpenMP

- Interface de programação (API) para desenvolvimento de aplicações multithread em C/C++ e Fortran
  - Define
    - Diretivas de compilação
    - Biblioteca de funções/serviços
    - Variáveis de ambiente
  - Busca estabelecer um padrão
    - ANSI X3H5 como primeira tentativa (1994)
  - Esforço iniciado de um grupo de fabricantes de hardware e desenvolvedores de software (Fujitsu, HP, IBM, Intel, Sun, DoE ASC, Compunity, EPCC, KSL, NASA Ames, NEC, SGI, ST Micro/PG, ST Micro/Portland Group, entre outros)

OpenMP

- API para C/C++
  - Variáveis de ambiente

```
OMP_NOME
```

  - Diretivas de compilação

```
#pragma omp diretiva [cláusula]
```

  - Biblioteca de serviços

```
omp_serviço( ... );
```

OpenMP

- **API para C/C++**
  - **Variáveis de ambiente**

OMP_*NOME*

  - Identifica o número de tarefas que serão executadas em paralelo

    `OMP_NUM_THREADS (default: número de processadores)`
  - Indica se o número de tarefas a serem executadas em paralelo deve ou não ser ajustado dinamicamente

    `OMP_DYNAMIC (default: FALSE)`
  - Indica se deve ser contemplado ativação de paralelismo aninhado

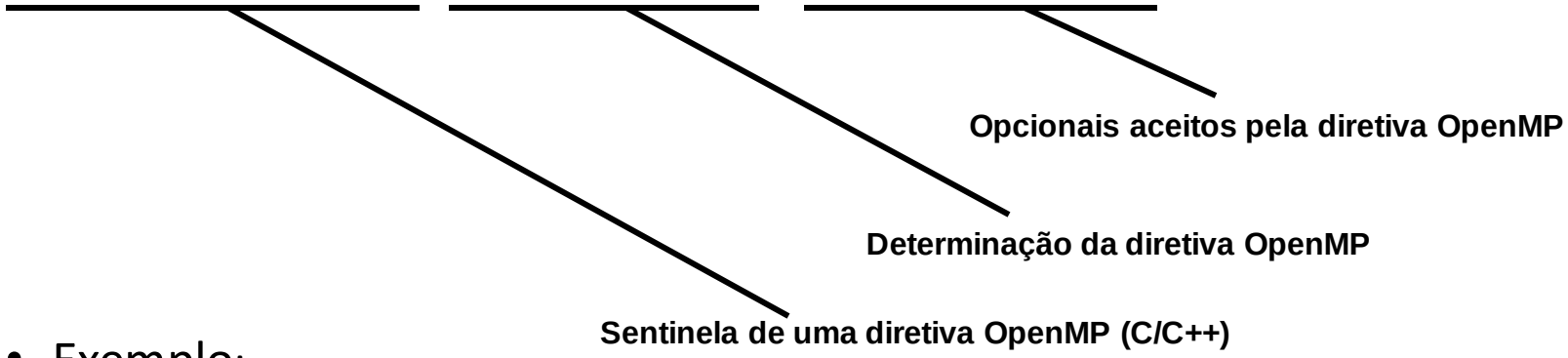
    `OMP_NESTED (default: FALSE)`
  - Define esquema de escalonamento das tarefas paralelas

    `OMP_SCHEDULE (default: estático)`

OpenMP

- **API para C/C++**
  - **Diretivas**
    - Formato:

```
#pragma omp diretiva [cláusula]
```



    - Exemplo:

```
#pragma omp parallel default(shared) private(x,y)
```

OpenMP

- **API para C/C++**
  - **Biblioteca de serviços**
    - Permitem controlar e interagir com o ambiente de execução.

    - Formato:

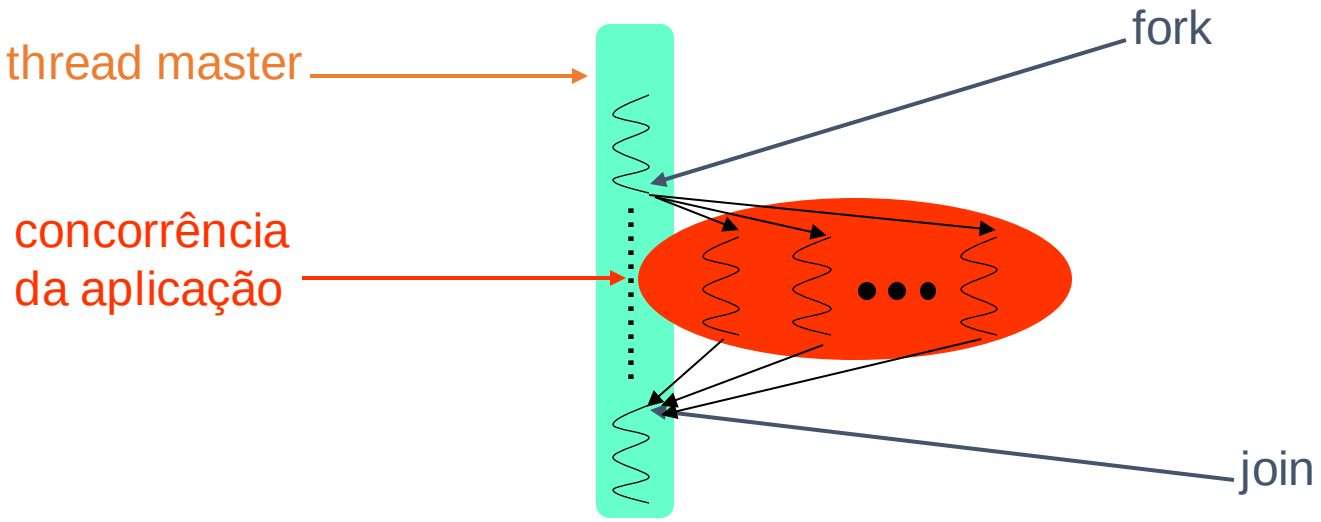
```
retorno omp_serviço ( [parâmetros] );
```

    - Exemplos:

```
void omp_set_num_threads( 10 );
int omp_get_num_threads();
void omp_set_nested( 1 );
int omp_get_num_procs ();
```

OpenMP

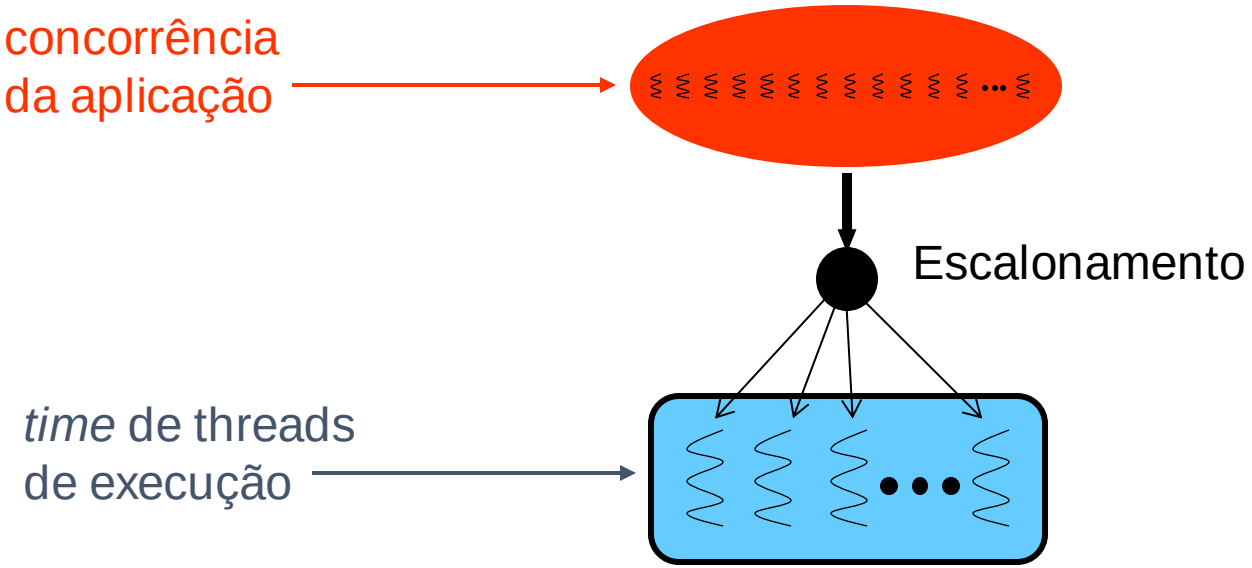
- Modelo básico de execução
  - Fork / Join



  - O programador descreve a concorrência de sua aplicação

OpenMP

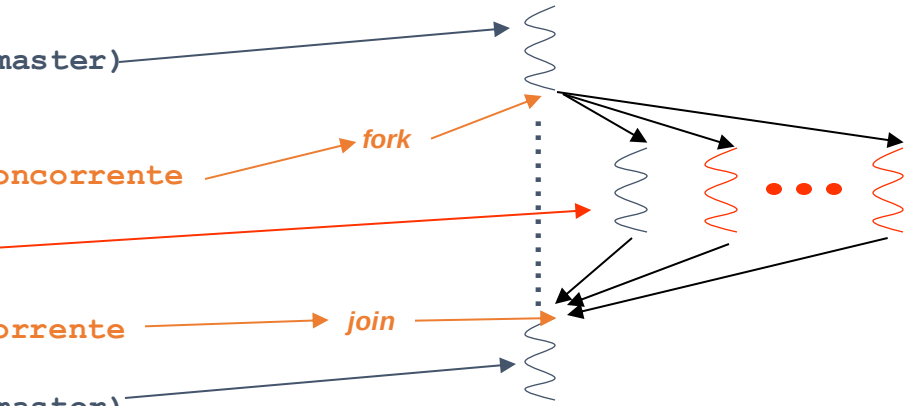
- Modelo básico de execução
  - Fork / Join
  - O programador descreve a concorrência de sua aplicação
  - A execução se da por um *time* de threads



OpenMP

- Primeiro programa

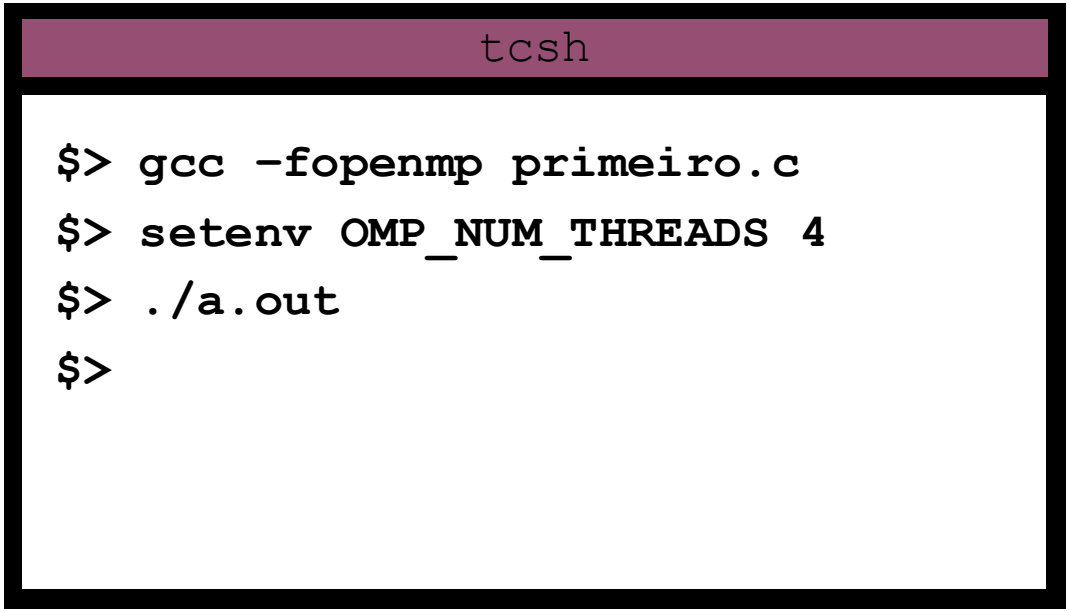
```
#include <omp.h>
main ( ) {

 // Código seqüencial (master)

#pragma omp parallel
{ // Início do código concorrente

  // Código paralelo

} // Fim do código concorrente

 // Código seqüencial (master)
}
```



São executadas tantas instâncias do código paralelo quanto forem o número de threads no time.

OpenMP

- Primeiro programa
- Primeira sessão (Linux)



OpenMP

- Programa mais elaborado

```
#include <omp.h>

main ()  {
int nthreads, tid;

#pragma omp parallel private(tid)
  {
   tid = omp_get_thread_num();
   printf("Oi mundo, sou o thread %d\n", tid);

   if( tid == 0 ) {
     nthreads = omp_get_num_threads();
     printf("Total de threads: %d\n", nthreads);
   }
  }
}
```

Lendo o programa:

- Dados globais
- Fork
- Join
- Cópia de dados privado ao thread
- Seção paralela
- Identificação do thread *master*

OpenMP

- Programa mais elaborado

```
#include <omp.h>

main ()  {
int nthreads, tid;

#pragma omp parallel private(tid)
  {
   tid = omp_get_thread_num();
   printf("Oi mundo, sou o thread %d\n", tid);

   if( tid == 0 ) {
     nthreads = omp_get_num_threads();
     printf("Total de threads: %d\n", nthreads);
   }
  }
}
```

tcsh

```
$> setenv OMP_NUM_THREADS 4
$> ./a.out
Oi mundo, sou o thread 3
Oi mundo, sou o thread 0
Total de Threads: 4
Oi mundo, sou o thread 1
Oi mundo, sou o thread 2
$>
$> setenv OMP_NUM_THREADS 5
$> ./a.out
Oi mundo, sou o thread 0
Oi mundo, sou o thread 1
Oi mundo, sou o thread 4
Total de Threads: 5
Oi mundo, sou o thread 3
Oi mundo, sou o thread 2
```

OpenMP

- Programação
  - Diretiva: **sections**

```
main ()  {
int x;

#pragma omp sections
  {
  #pragma omp section
  {
   foo(x);
  }
  #pragma omp section
  {
   bar(x);
  }
  }
}
```

Dado compartilhado

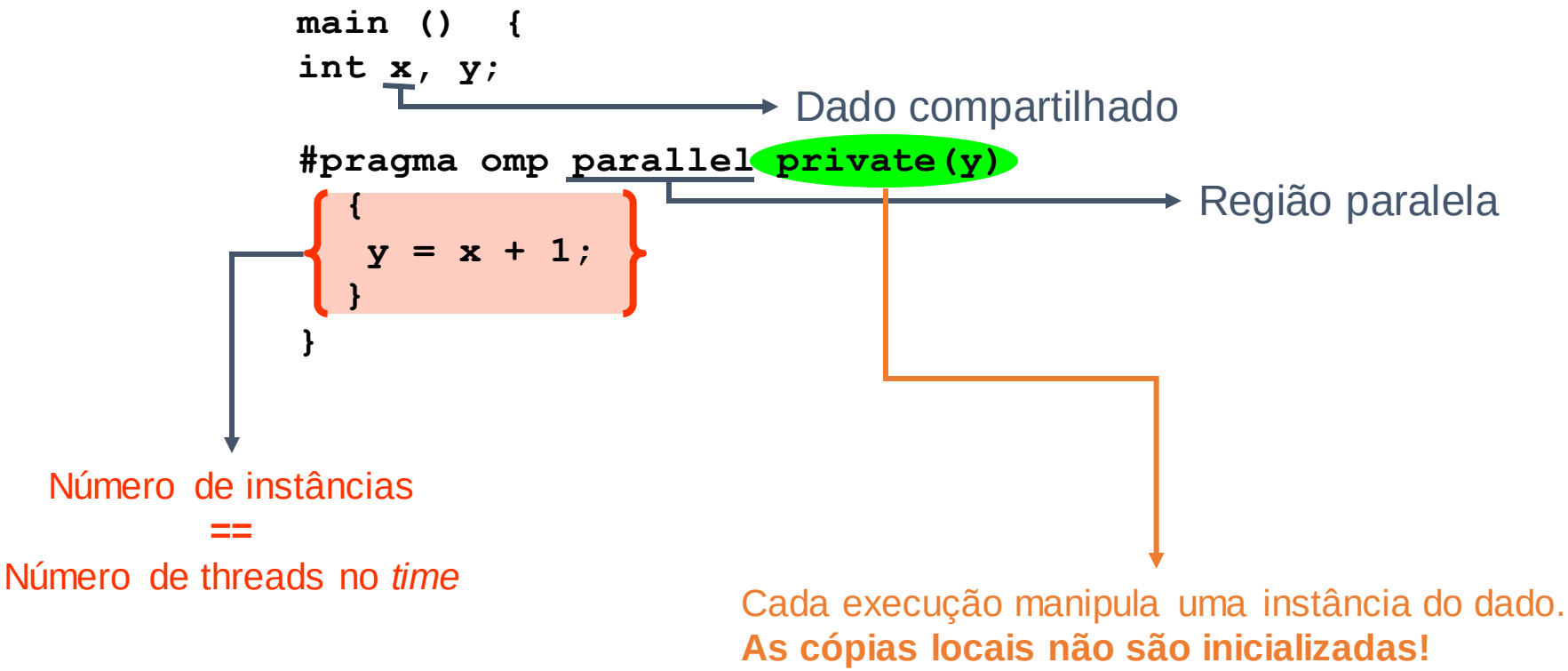
Região paralela

OpenMP

- Programação
  - Diretiva: `sections`
  - **Cláusulas aceitas**
    - `private`
    - `firstprivate`
    - `lastprivate`
    - `reduction`
    - `nowait`

OpenMP

- Programação
  - Diretiva: **parallel** Cláusula **private**



OpenMP

- Programação
  - Diretiva: **`parallel`** Cláusula **`private`**
  - **Cláusulas alternativas:**
    - **`firstprivate`**
      - Cada cópia local é inicializada com o valor que o dado possui no escopo do thread master
    - **`lastprivate`**
      - O dado no escopo do thread master é atualizado com o valor da última avaliação da região paralela
    - Exemplo

```
#pragma omp parallel lastprivate(y) firstprivate(y)
```

Pode ser *last* e *first* ao mesmo tempo!

OpenMP

- Programação
  - Diretiva: **parallel** Serviço **omp_set_num_threads**

```
main ()  {
int x, y;

omp_set_num_threads( 5 );

#pragma omp parallel private(y)
   {
    y = x + 1;
   }
}
```

Novo número de threads no *time*

OpenMP

- Programação
  - Diretiva: **`parallel`** Cláusula **`reduction`**
    - Cada execução pode manipular sua cópia, como em `private`
    - O valor local inicial é definido pela operação de *redução*
    - No final uma operação de *redução* atualiza o dado no thread master

Exemplo:

```
int soma = 100;

omp_set_num_threads( 4 );
#pragma omp parallel reduction( + : soma )
  {
   soma += 1;
  }
// No retorno ao master: soma = ?
```

OpenMP

- Programação
  - Diretiva: **`parallel`** Cláusula **`reduction`**
    - Cada execução pode manipular sua cópia, como em `private`
    - O valor local inicial é definido pela operação de *redução*
    - No final uma operação de *redução* atualiza o dado no thread master

Exemplo:

```
int soma = 100;

omp_set_num_threads( 4 );
#pragma omp parallel reduction( + : soma )
  {
   soma += 1;
  }
// No retorno ao master: soma = 104
```

OpenMP

- Programação
  - Diretiva: **`parallel`** Cláusula **`reduction`**
    - Cada execução pode manipular sua cópia, como em `private`
    - O valor local inicial é definido pela operação de *redução*
    - No final uma operação de *redução* atualiza o dado no thread master

Operações de redução:

Operador	Operação	Valor inicial
+	Soma	0
-	Subtração	0
*	Multiplicação	1
&	E aritmético	-1
&&	E lógico	1
|	OU artitmético	0
||	OU lógico	0
^	XOR aritmético	0

OpenMP

- Programação
  - Diretiva: **`parallel`**
    - Cláusulas aceitas
      - **`reduction`**
      - **`private`**
      - **`firstprivate`**
      - **`shared`**
      - **`default`**
      - **`copyin`**
      - **`if`**
      - **`num_threads`**

OpenMP

- Programação
  - Diretiva: **`for / parallel for`**
    - Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
      - Note: **`for`** e **`paralell for`** não é exatamente igual
        - **`for`**: restringe o número de cláusulas aceitas
        - **`parallel for`**: aceita todas as cláusulas do **`for`** e do **`parallel`**

OpenMP

- Programação
  - Diretiva: **`for / parallel for`**
    - Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
    - Exemplo:

```
int vet[100], i, soma = 0;

#pragma omp parallel for private(i)
 for( i = 0 ; i < 100 ; i++ ) vet[i] = i;

#pragma omp parallel for private(i) reduction(+:soma)
 for( i = 0 ; i < 100 ; i++ ) soma = soma + vet[i];

printf( "Somatorio: %d\n", soma );
```

OpenMP

- Programação
  - Diretiva: **`for / parallel for`**
    - Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
    - Exemplo:

```
int mat[TAM][TAM], aux[TAM][TAM], i, j;
inicializa(mat);
copia(mat, aux);
while( não estabilizou os resultados ) {
  #pragma omp parallel for
    for( i = 0 ; i < TAM ; i++ )
      #pragma omp parallel for
      for( j = 0 ; j < TAM ; j++ )
        mat[i][j] = func(aux, i, j, ...);
  copia(mat,aux);
}
```

OpenMP

- Programação
  - Diretiva: **`for / parallel for`**
    - Permite que os grupos de instruções definidas em um *loop* sejam paralelizadas
    - Regras:

```
#pragma omp parallel for private(i)
 for( i = 0 ; i < 100 ; i++ ) vet[i] = i;
```

- A variável de iteração é tornada local implicitamente no momento em que um loop é paralelizado
- A variável de iteração e o valor de teste necessitam ser inteiros e com sinal
- A variável de iteração deve ser incrementa ou decrementada, não são permitidas outras operações
- Os testes são: <, >, <=, >=
- Não é permitido: `for( i = 0 ; i < 100 ; vet[i++] = 0 );`

OpenMP

- Programação
  - Diretiva: **`critical[(nome)]`**
    - Permite que trechos de código sejam executados em regime de exclusão mútua
    - É possível associar *nomes* aos trechos

```
int x, y;
int prox_x, prox_y;
#pragma omp parallel shared(x, y) private(prox_x, prox_y)
 {
 #pragma omp critical(eixox)
  prox_x = dequeue( x );
 trataFoo( prox_x, x );

 #pragma omp critical(eixoy)
  prox_y = dequeue( y );
 trataFoo( prox_y, y );
}
```

OpenMP

- Programação
  - Diretiva: **`single [nowait]`**
    - Indica, em uma região paralela, um trecho de código que deve ser executado apenas por uma única thread
    - Representa uma barreira
    - Com **`nowait`** a barreira pode ser relaxada

```
#pragma omp parallel
 {
 #pragma omp single
  printf("Serao %d threads\n", omp_get_num_threads() );

  foo();

 #pragma omp single nowait
   printf("O threads %d terminou\n", omp_get_thread_num() );
}
```

OpenMP

- Programação
  - Diretiva: **`master [nowait]`**
    - Indica, em uma região paralela, um trecho de código que deve ser executado apenas thread master
    - Representa uma barreira
    - Com **`nowait`** a barreira pode ser relaxada

```
#pragma omp parallel
 {
 #pragma omp master
  printf("Serao %d threads\n", omp_get_num_threads() );

  foo();

 #pragma omp master nowait
   printf("O thread master terminou\n" );
}
```

OpenMP

- Programação
  - Cláusula: **`schedule(static|dynamic|guided)`**
    - Indica qual estratégia deve ser utilizada para distribuir trabalho entre os threads do time
      - **`static`**
        - Implica na divisão do trabalho por igual entre cada thread
        - Indicado quando a quantidade de trabalho em cada grupo de instruções (como em cada iteração em um `for`) é a mesma
      - **`dynamic`**
        - Cada thread busca uma nova quantidade de trabalho quando terminar uma etapa de processamento
        - Indicado quando a quantidade de trabalho em cada grupo de instruções (como em cada iteração em um `for`) não for a mesma
      - **`guided`**
        - Semelhante ao `dynamic`, mas com divisão não uniforme na divisão das tarefas
        - Normalmente o tamanho da porção de trabalho (chunk) começa grande e vai diminuindo conforme necessário para tratar possíveis desbalanceamentos de carga

Referência

- Material baseado nos slides do Prof Gerson Geraldo H. Cavalheiro da UFPel.