

OpenMP

Prof. Dr. Márcio Castro
marcio.castro@ufsc.br



1

Introdução

O modelo de programação OpenMP

■ O que é OpenMP?

- Interface de programação (**API**)
- Baseada no modelo de programação paralela de **memória compartilhada**
- Disponível em diferentes linguagens: C, C++ e Fortran



<https://www.openmp.org/spec-html/5.0/openmp.html>

O modelo de programação OpenMP

■ Componentes

- Diretivas de compilação
- Biblioteca de execução
- Variáveis de ambiente

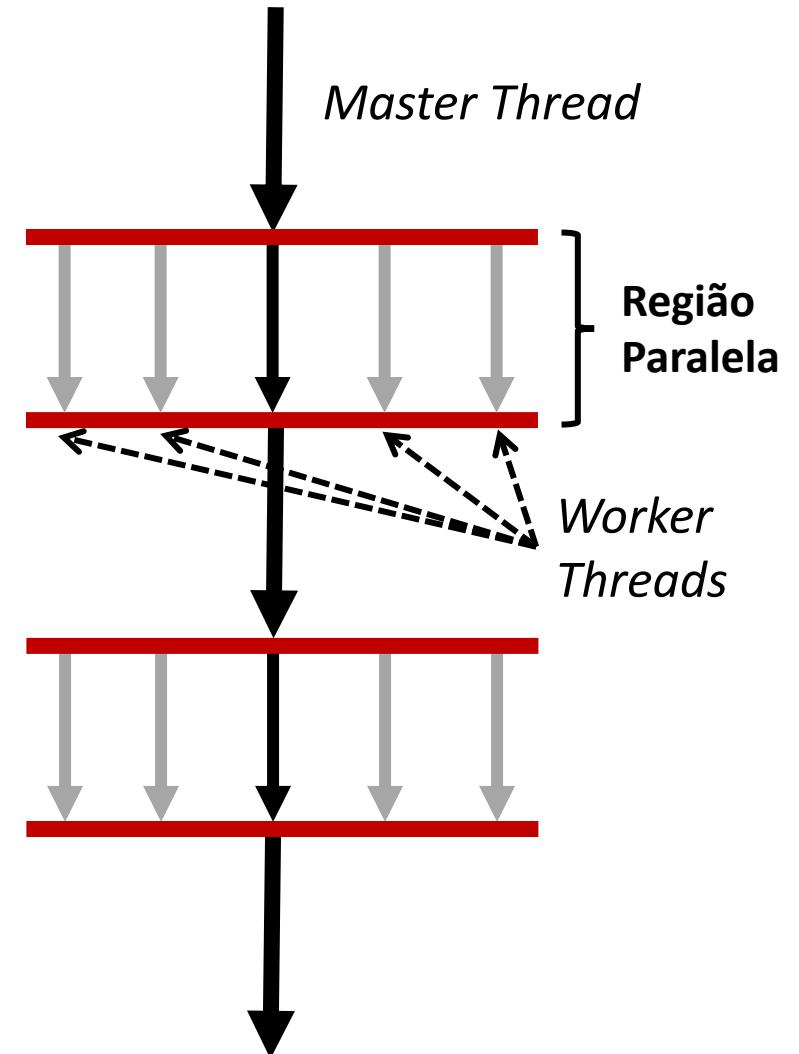
■ Vantagens

- **Bom desempenho e escalabilidade**
- **Portabilidade**
- **Pouco esforço de programação**
- Permite paralelização de aplicações de maneira **incremental**

O modelo de programação OpenMP

Fork and Join

- A execução inicia com uma única *thread* (*master thread*)
- **Fork:**
 - *Worker threads* são criadas em uma **região paralela**
- **Join:**
 - Barreira **implícita** ao final de uma região paralela
 - Execução continua **somente com a master thread**



Biblioteca e variáveis de ambiente

- OpenMP está presente em diferentes compiladores
 - **Fortran:** ifort
 - **C/C++:** gcc/g++ (gnu), icc (Intel)
- **Biblioteca (C):**
 - `#include <omp.h>`
- **Compilação (C):**
 - `gcc -o prog prog.c -fopenmp`

Biblioteca e variáveis de ambiente

■ Rotinas úteis

- `omp_set_num_threads(int t)`: define o número total de *threads* a serem utilizadas nas regiões paralelas
 - Também possível via variável de ambiente `OMP_NUM_THREADS`
- `omp_get_num_threads(void)`: retorna o número de *threads* dentro de uma região paralela
- `omp_get_thread_num(void)`: retorna o identificador único da *thread* dentro de uma região paralela
- `omp_get_num_procs(void)`: retorna o número de núcleos de processamento

Diretivas OpenMP

- Todo o paralelismo do OpenMP é especificado através de **diretivas de compilação**
- As diretivas OpenMP seguem o seguinte padrão:
 - **C/C++:** `#pragma omp diretiva [cláusulas]`
 - **Fortran:** `!$OMP diretiva [cláusulas]`
- **Exemplo de uma diretiva OpenMP em C:**
 - `#pragma omp parallel`

2

Diretiva parallel

Criação de regiões paralelas

Diretiva parallel

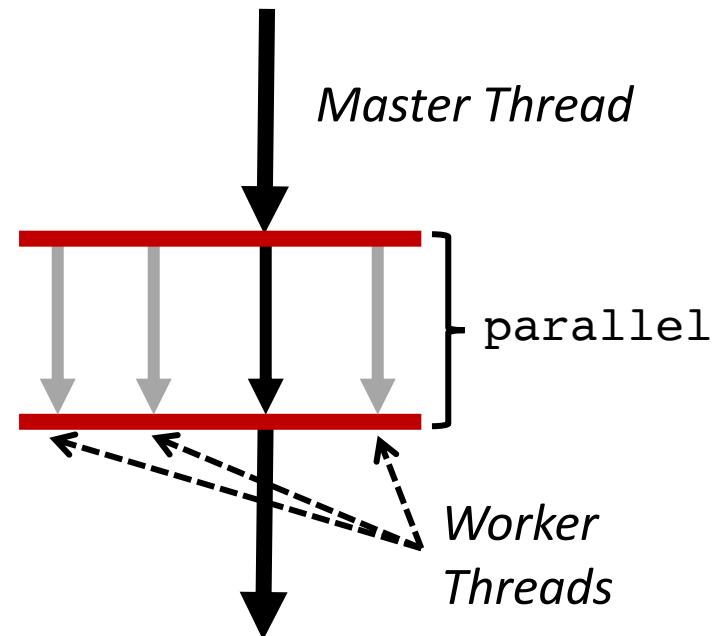
- **#pragma omp parallel**
 - É a construção fundamental do OpenMP
 - **Região paralela:** identifica um bloco de código que será executado por múltiplas threads

Exemplo:

```
#pragma omp parallel  
printf("Hello world!\n");
```

Saída (com 4 threads):

```
Hello world!  
Hello world!  
Hello world!  
Hello world!
```



Diretiva parallel

Escopo da diretiva parallel

Exemplo 1:

```
#pragma omp parallel
printf("Primeiro!\n");
printf("Segundo!\n");
```

Saída (com 2 threads):

```
Primeiro!
Primeiro!
Segundo!
```

Exemplo 2:

```
#pragma omp parallel
{
    printf("Primeiro!\n");
    printf("Segundo!\n");
}
```

Saída (com 2 threads):

Primeiro!	Primeiro!
Primeiro!	Segundo!
Segundo!	Primeiro!
Segundo!	Segundo!

Diretiva parallel

```
#pragma omp parallel [clauses]
```

Clauses:

```
if([parallel :] scalar-expression)
num_threads(integer-expression)
default(shared | none)
private(list)
firstprivate(list)
shared(list)
copyin(list)
reduction(reduction-identifier : list)
proc_bind(master | close | spread)
allocate([allocator :] list)
```

Diretiva parallel

- Cláusula: **private(list)**

- Cria uma **cópia local** das variáveis em cada *thread*
- Cópias locais **não são inicializadas**
- **Argumentos:** uma lista de variáveis (list)

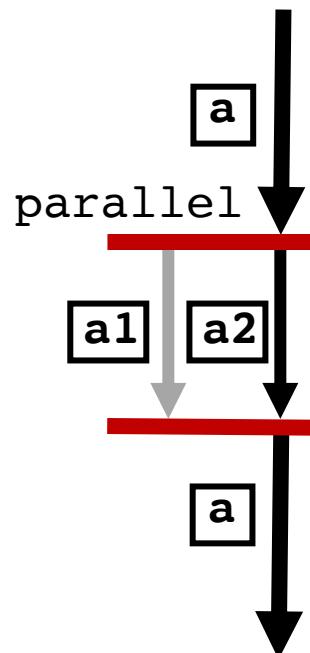
Exemplo:

```
int a = -1;  
#pragma omp parallel private(a)  
printf("dentro = %d\n", a);  
printf("fora = %d\n", a);
```

Saída (com 2 threads):

```
dentro = 37613  
dentro = 0  
fora = -1
```

Valores indefinidos!



Diretiva parallel

- Cláusula: **firstprivate(list)**

- Cria uma **cópia local** das variáveis em cada *thread*
- Cópias locais são inicializadas com valor que as variáveis possuíam antes da **região paralela**
- **Argumentos:** uma lista de variáveis (list)

Exemplo:

```
int a = -1;
#pragma omp parallel firstprivate(a)
{
    printf("dentro = %d\n", a);
    a = 123;
}
printf("fora = %d\n", a);
```

Saída (com 2 threads):

```
dentro = -1
dentro = -1
fora = -1
```

Diretiva parallel

■ Cláusula: **shared**(list)

- Indica que as variáveis serão **compartilhadas entre threads**
- Todas as **threads** usam a mesma instância das variáveis
- **Argumentos:** uma lista de variáveis (list)

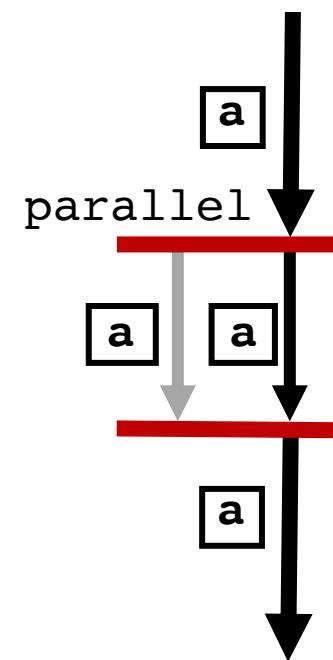
Exemplo:

```
int a = -1;
#pragma omp parallel shared(a)
{
    printf("dentro antes = %d\n", a);
    a = omp_get_thread_num();
    printf("dentro depois = %d\n", a);
}
printf("fora = %d\n", a);
```

Saída (com 2 threads):

```
dentro antes = -1
dentro antes = -1
dentro depois = 0
dentro depois = 1
fora = 1
```

```
dentro antes = -1
dentro depois = 1
dentro antes = 1
dentro depois = 0
fora = 0
```



Diretiva parallel

- **Sobre o compartilhamento de variáveis**

- Salvo algumas exceções, variáveis declaradas **fora de uma região paralela** serão *shared*
- Variáveis declaradas dentro de uma região paralela serão *private*

```
float var1;

void f() {
    int var2;
    #pragma omp parallel
    {
        double var3;
        // var1 e var2 serão shared!
        // var3 será private!
    }
}
```

Diretiva parallel

- **Cláusula:** `reduction(reduction-id : list)`
 - Cria uma **cópia local** das variáveis em cada *thread*
 - Cópias locais **são inicializadas com valor 0 ou 1**, em função do **operador de redução** usado
 - Ao final da região paralela ocorre uma **operação de redução** em todas as variáveis locais
 - Os valores armazenados nas variáveis locais **são copiados para as variáveis externas** à região paralela
 - **Argumentos:** um operador de redução (`reduction-id`) e uma lista de variáveis (`list`)

Diretiva parallel

Cláusula: reduction(reduction-id : list)

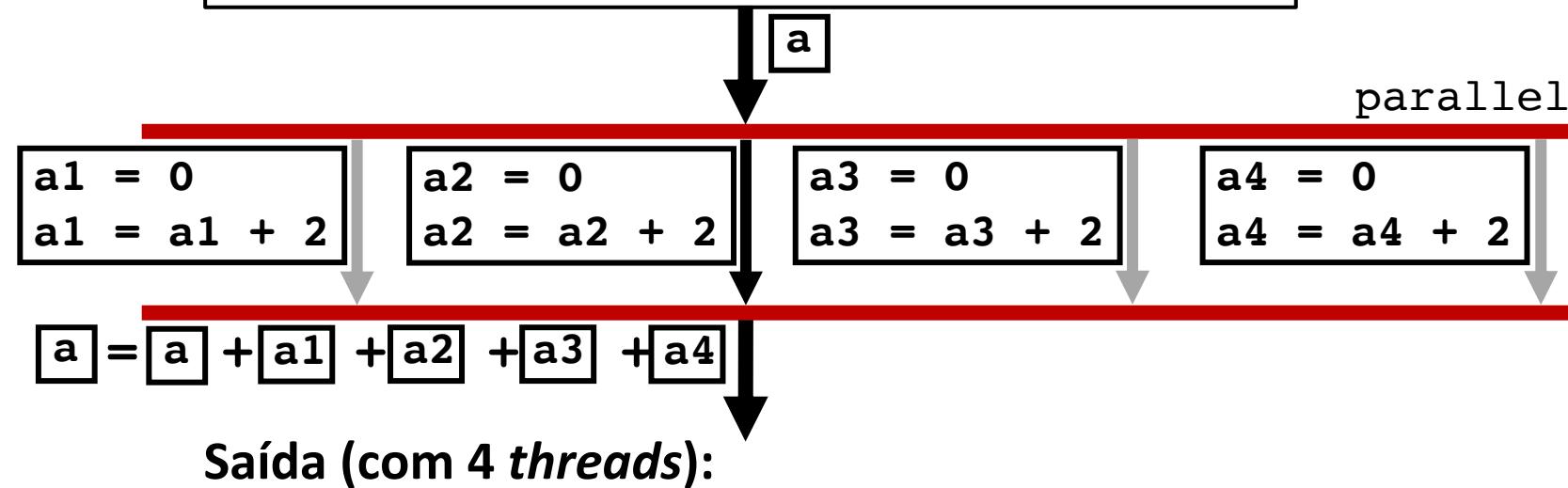
reduction-id	Inicialização	Operação
+	0	out += in
-	0	out -= in
*	1	out *= in
&	~0	out &= in
	0	out = in
^	0	out ^= in
&&	1	out = in && out
	0	out = in out
max	Menor número representável	out = in > out ? in : out
min	Maior número representável	out = in < out ? in : out

Diretiva parallel

- Cláusula: **reduction**(reduction-id : list)

Exemplo:

```
int a = 1;  
#pragma omp parallel reduction(+:a)  
a = a + 2;  
printf("resultado = %d\n", a);
```



```
resultado = 9
```

2

Diretiva `for`

Paralelização de laços

Diretiva for

- Utilizada dentro de uma **região paralela** ou através de uma **construção combinada parallel for**
- Permite **paralelizar laços** de maneira automática
- Iterações do laço são **distribuídas e executadas em paralelo** pelas *threads* da região paralela

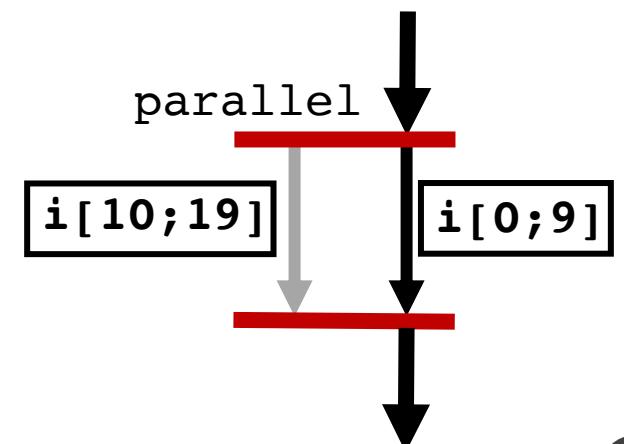
Exemplo (sequencial):

```
for (i = 0; i < 20; i++)  
    c[i] = a[i] + b[i];
```

Exemplo (paralelo):

```
#pragma omp parallel  
#pragma omp for  
for (i = 0; i < 20; i++)  
    c[i] = a[i] + b[i];
```

Com 2 threads:



#pragma omp for [clauses]

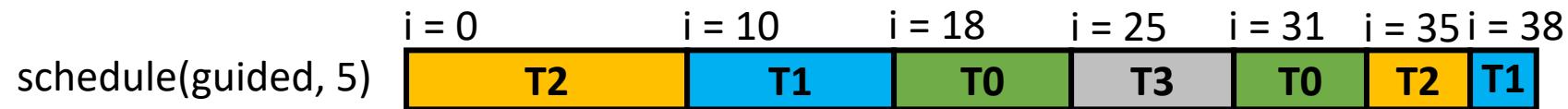
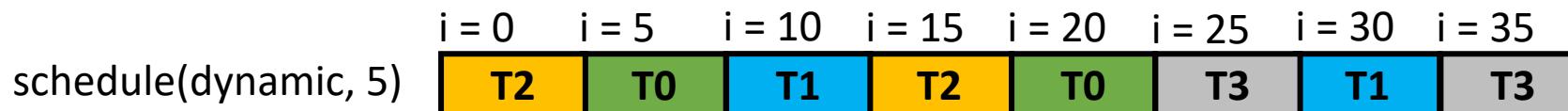
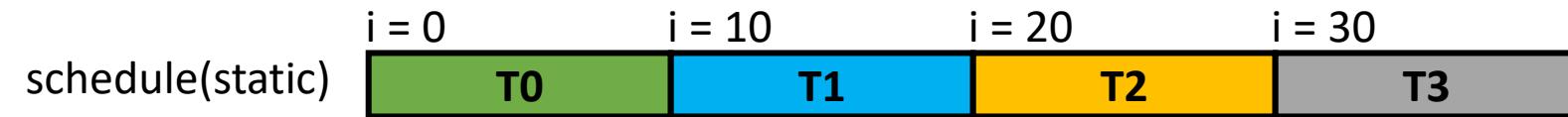
Clauses:

```
private(list)
firstprivate(list)
lastprivate([ lastprivate-modifier:] list)
linear(list[ : linear-step])
reduction(reduction-identifier : list)
schedule(kind[, chunk_size])
collapse(n)
ordered( n )
nowait
allocate([ allocator :]list)
order(concurrent)
```

- **Cláusula:** `schedule(kind[, chunk_size])`
 - Define como as iterações de um laço serão **divididas entre as threads (estratégia de escalonamento)**
 - Iterações individuais podem ser agrupadas em **blocos**, denominados **chunks**
- **Argumentos:** estratégia de escalonamento (`kind`) e tamanho de blocos de iterações (`chunk_size`)
 - `kind`: static, dynamic, guided ou auto
 - `chunk_size`: quantidade de iterações em cada bloco
 - static e dynamic possuem **chunk de tamanho fixo**; guided possui **chunk de tamanho variável**

Diretiva `for`

Exemplo: laço de 40 iterações e 4 threads



Exemplo: produto escalar

Código sequencial:

```
soma = 0;  
for (i = 0; i < N; i++)  
    soma += a[i] * b[i];
```

arrays *a* e *b* são
compartilhados

Código paralelo:

```
soma = 0;  
#pragma omp parallel shared(a, b)  
#pragma omp for schedule(static) reduction(+:soma)  
for (i = 0; i < N; i++)  
    soma += a[i] * b[i];
```

Escalonamento

Redução ocorrerá sobre **soma**
(considerada como **private** em cada *thread*)

3

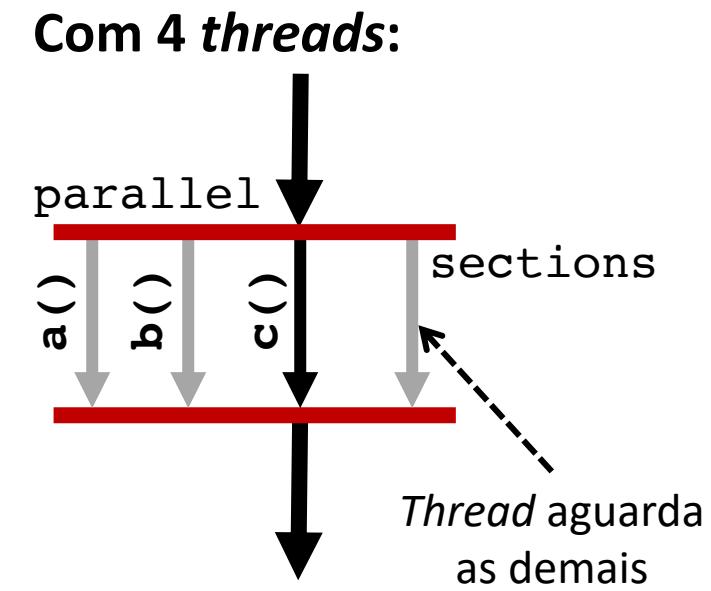
Diretiva sections

Execução paralela de trechos de código

Diretiva sections

- Utilizada dentro de uma **região paralela** ou através de uma **construção combinada parallel sections**
- Permite dividir **trechos do código (sections)** entre **threads**
 - Cada **section** é executada por uma **thread**
- Sincronização implícita no final (exceto com nowait)

```
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        a();
        #pragma omp section
        b();
        #pragma omp section
        c();
    }
}
```



Diretiva sections

```
#pragma omp sections [clauses]
{
    [#pragma omp section]
        structured-block
    [#pragma omp section
        structured-block]
    ...
}
```

Clauses:

```
private(list)
firstprivate(list)
lastprivate([ lastprivate-modifier:] list)
reduction(reduction-identifier : list)
allocate([allocator :]list)
nowait
```

Diretivas de sincronização

Sincronização entre *threads* em uma região paralela

Diretiva single

- A diretiva `single` determina que o código em uma região paralela seja executado por **somente uma única *thread***
- As demais *threads* aguardam a execução (exceto com `nowait`)

Exemplo

```
#pragma omp parallel
{
    printf("Thread %d: iniciada\n", omp_get_thread_num());
    #pragma omp single
    printf("Total de threads = %d\n", omp_get_num_threads());
}
```

Saída (com 4 *threads*)

```
Thread 2: iniciada
Thread 1: iniciada
Total de threads = 4
Thread 0: iniciada
Thread 3: iniciada
```

Diretiva single

```
#pragma omp single [clauses]
```

Clauses:

```
private(list)
firstprivate(list)
copyprivate(list)
allocate([allocator :]list)
nowait
```

Diretiva critical

```
#pragma omp critical [ (name) ]
```

- Funciona como um **mecanismo de exclusão mútua**
- Todas as *threads* executam, porém, **somente uma por vez**

Exemplo

```
x = 0;  
#pragma omp parallel shared(x)  
{  
    #pragma omp critical  
    x++;  
}  
printf("x = %d\n", x);
```

Saída (com 4 threads)

```
x = 4
```

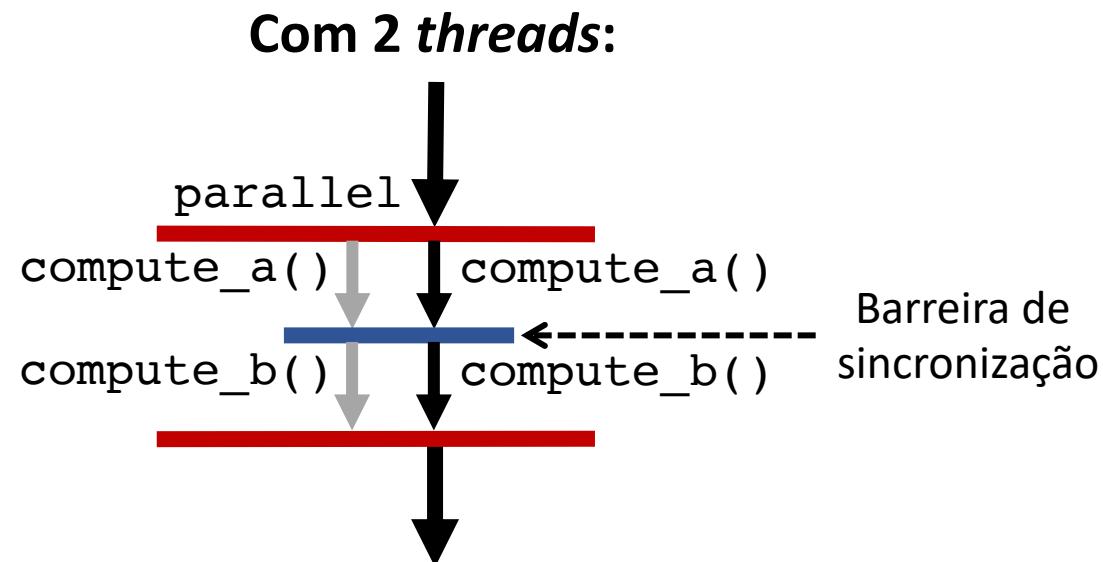
Diretiva barrier

#pragma omp barrier

- Sincroniza todas as *threads* em uma barreira

Exemplo

```
#pragma omp parallel
{
    compute_a();
    #pragma omp barrier
    compute_b();
}
```



!

Obrigado pela atenção!



Dúvidas? Entre em contato:

- marcio.castro@ufsc.br
- www.marciocastro.com



Distributed Systems Research Lab
www.lapesd.inf.ufsc.br