

The background of the slide is a light gray with an abstract geometric pattern. On the left side, there is a dense network of dark gray lines connecting various points, forming a complex web. Scattered across the rest of the slide are several thin, light gray outlines of triangles of different sizes and orientations. Some of these triangles are solid, while others are just outlines. The overall aesthetic is modern and technical.

OpenMP

Ariane Cristina Fonseca Silva - nº USP 10752119

João Victor Corrêa Rodrigues - nº USP 10697935

Matheus J. Castro - nº USP 10314555

Rafael O. Scatena - nº USP 3484681

Contexto Histórico 01

Como funciona? 02

Conceitos Básicos 03

CONTEÚDO

04 Prós e Contras

05 Hands on

06 Bibliografia



01

CONTEXTO HISTÓRICO

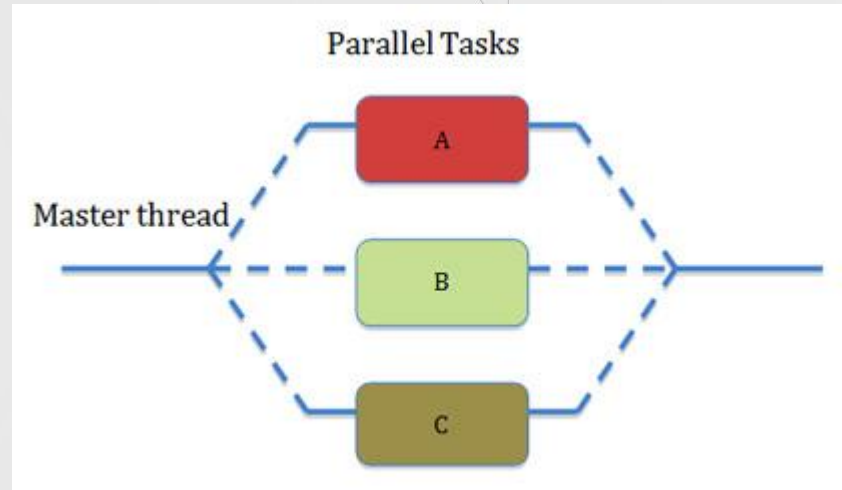
O que é OpenMP?

Cronologia

Objetivos do OpenMP

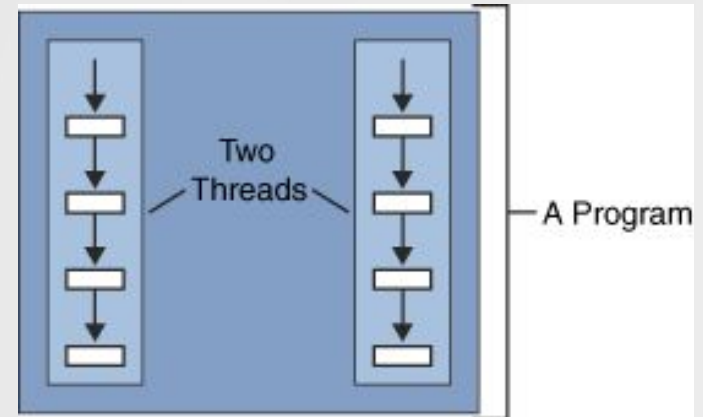
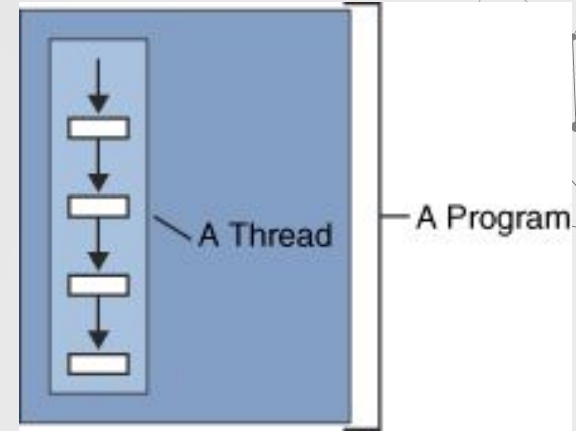
O QUE É?

- **OPEN** specifications for **MULTI PROCESSING** via collaborative work from software/hardware industry, academia and government.
- Modelo de programação em memória compartilhada que se originou da cooperação das grandes fabricantes de hardware e software (Intel, IBM, HP, KAI, etc)
- Constituída por diretivas de compilação, biblioteca de funções e variáveis de ambiente.
- Fortran ou C/C++



O QUE É?

- Desenvolvido e mantido pelo grupo OpenMP ARB (Architecture Review Board).
 - <http://www.openmp.org>
- Possui implementações para UNIX/Linux e Windows.
- Permite a criação de programas paralelos com compartilhamento de memória, através da criação automática e otimizada de um conjunto de threads.



CRONOLOGIA

1997

Fornecedores se juntam e formam o OpenMP ARB.
Lançamento da versão 1.0

1998

Primeiras aplicações híbridas com OpenMP e MPI surgem.

2001

Grupo de usuários (cOMPunity) é formado para divulgação do OpenMP no mundo.
Lançamento da versão 2.0

2005

C/C++ e Fortran são unificados.
Lançamento da versão 2.5

2008

Lançamento da versão 3.0 incorporando o paralelismo de tarefas

2011

Lançamento da versão 3.1 com suporte para reduções máx/mín em C/C++

2013


Lançamento da versão 4.0 com suporte a aceleradores e dispositivos coprocessadores

2018

Lançamento da versão 5.0 com suporte total para aceleradores e compatibilidade com versões recentes do C/C++ e Fortran.

Objetivos do OpenMP

- Prover um padrão para uma variedade de plataformas e arquiteturas baseadas em memória compartilhada.
- Estabelecer um conjunto simples e limitado de diretivas de programação.
- Permitir a paralelização incremental de programas sequenciais.
- Implementar paralelismo com granularidade:
 - **Fina:** paralelismo entre as instruções
 - **Média:** paralelismo entre as tarefas
 - **Grossa:** paralelismo entre os processos



02

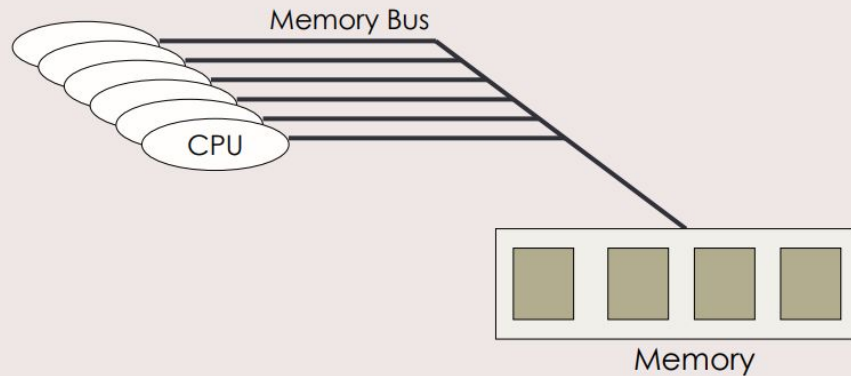
COMO FUNCIONA?

Memória
Modelo Fork-Join
"Nested Fork-Join"
Três tipos de Construção
Diretiva compartilhada Do/for
"Schedule" (Agendamento)
Diretiva Sections
Diretiva Single

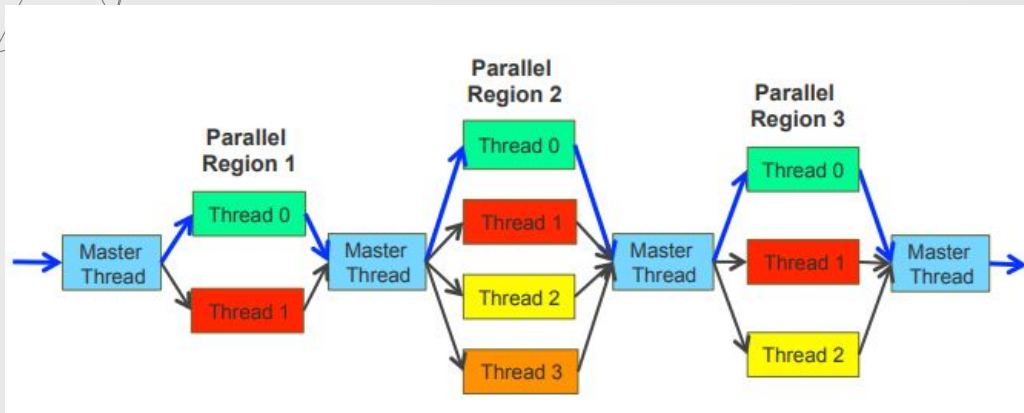
Memória

Modelo de memória compartilhada

- Private
- First Private
- Shared



MODELO FORK-JOIN



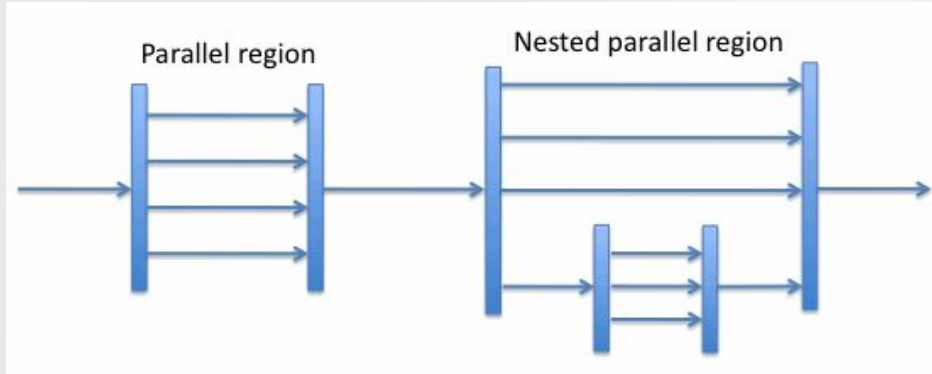
- **Thread mestre:** executa o código até a primeira região paralela ser encontrada
- **Fork (bifurcação):** O thread mestre cria um conjunto de threads paralelos que executarão os comandos da região paralela (o thread mestre faz parte do conjunto e tem nº de identificação 0)
- **Join (união):** Fim da região paralela. Os threads são sincronizados e resta apenas o thread mestre.

“NESTED FORK-JOIN”

```
#pragma omp parallel for num_threads(2)
for (j=0; j<m; j++) {
  #pragma omp parallel for num_threads(3)
  for (i=0; i<n; i++) {
    c[j][i] = a[j][i] + b[j][i];
  }
}
```

First level parallel
region with 2 threads

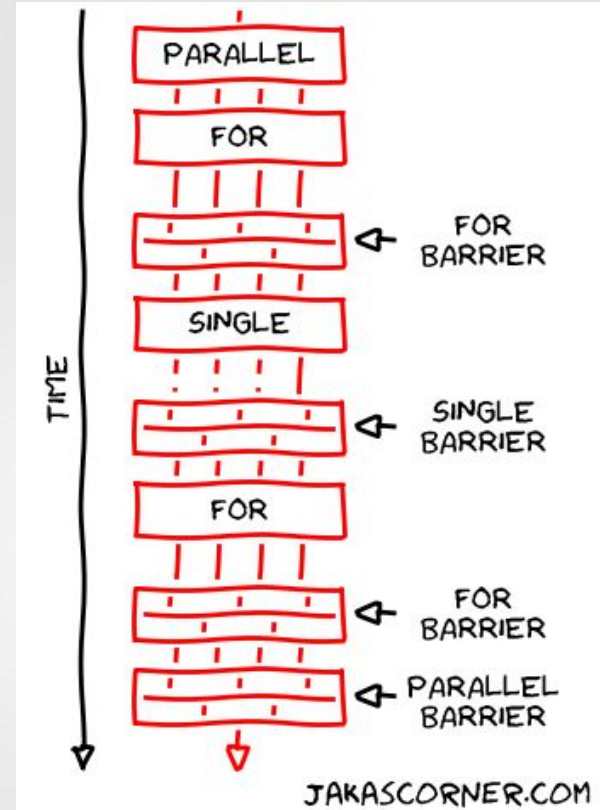
Second level parallel
region with 3 threads
for each outer thread
with a total of 6 threads



Sincronização

- Sincronização das ações em variáveis compartilhadas
- Assegurar ordenação correta de leituras e escritas
- Proteger atualização de variáveis compartilhadas

Por padrão existe uma diretiva “Barrier” ao final de toda região paralela requerida para forçar a sincronização, porém elas podem ser “caras” computacionalmente. Isso pode ser alterado dependendo da construção do código (com cuidado).

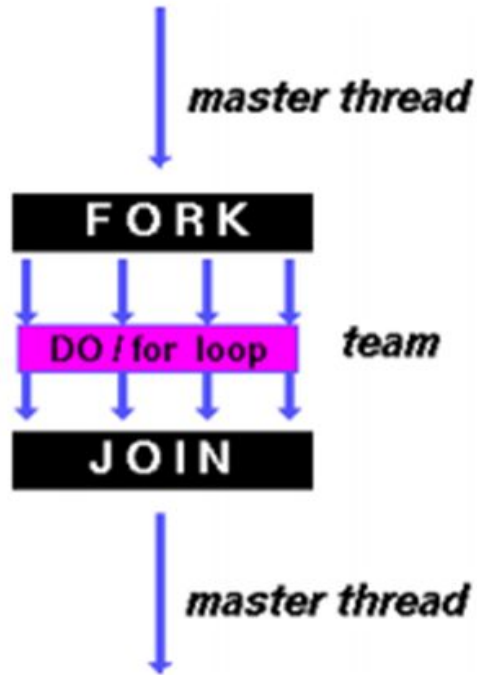




TRÊS TIPOS DE CONSTRUÇÃO

- **DO/FOR:** Compartilhamento das iterações de loops entre as threads.
- **Sections:** Quebra do código em seções. Cada uma delas será executada por uma thread.
- **Single:** Determina que uma seção da região paralela seja executada por uma única thread.

DIRETIVA COMPARTILHADA Do/for

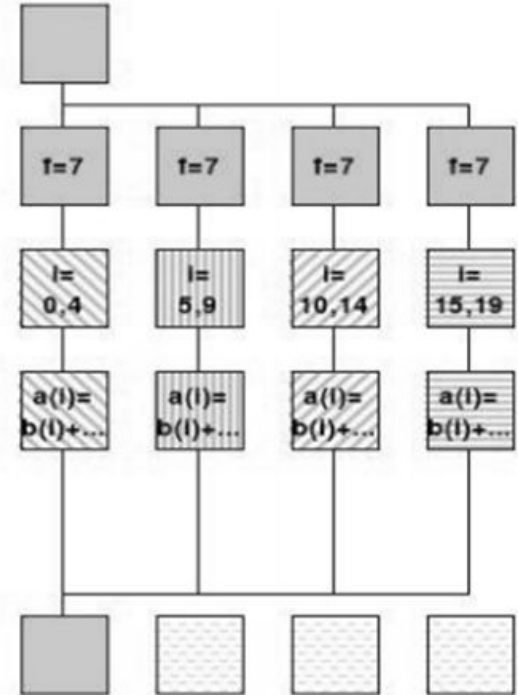


C / C++:

```
#pragma omp parallel private(f)
{
    f=7;
```

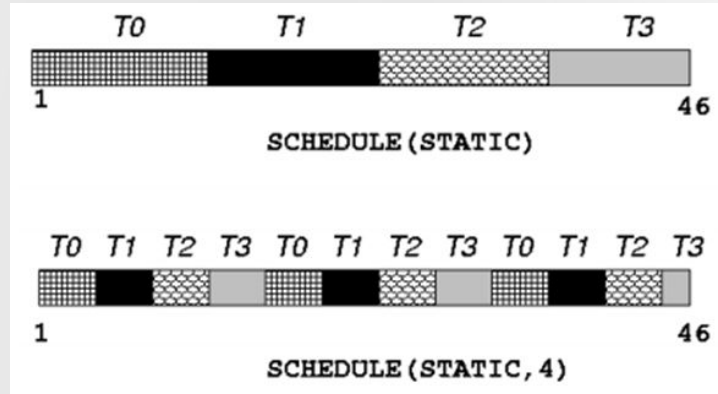
```
#pragma omp for
for (i=0; i<20; i++)
    a[i] = b[i] + f * (i+1);
```

```
 } /* omp end parallel */
```



“SCHEDULE” (Agendamento)

- **STATIC:** O loop é dividido em pedaços e distribuído estaticamente para cada thread. Se o tamanho do pedaço não for definido, o compilador divide em partes iguais (quando possível).





```
schedule(static):
*****
                *****
                        *****
                                *****
```

```
schedule(static, 4):
****          ****          ****          ****
  ****      ****      ****      ****
    ****    ****    ****    ****
      ****  ****  ****  ****
        ****      ****      ****      ****
```

```
schedule(static, 8):
*****          *****
  *****      *****
    *****    *****
      *****  *****
        *****      *****
```


“SCHEDULE” (Agendamento)

- **DYNAMIC:** O loop é dividido em pedaços com certo tamanho e distribuídos dinamicamente para cada thread conforme disponibilidade. O tamanho padrão do pedaço é 1.





```
schedule(dynamic):
```

```
*  * * * *  * * * *  * * * *  * * * *  * * * *
*      *      *      *      *      *      *      *
*      *      *      *      *      *      *      *
* *      *      *      *      *      *      *      *
```

```
schedule(dynamic, 1):
```

```
      *      *      *      *      *      *      *      *
* * * *  * *      * * * *  * *      *      *      *
* * * *  * * * *  * *      *      *      *      *
*      *      *      *      *      *      *      *
```

```
schedule(dynamic, 4):
```

```
      * * * *      * * * *      * * * *      * * * *
* * * *      * * * *      * * * *      * * * *      * * * *
      * * * *      * * * *      * * * *      * * * *
      * * * *      * * * *      * * * *      * * * *
```

```
schedule(dynamic, 8):
```

```
      * * * * * * * *      * * * * * * * *
      * * * * * * * *      * * * * * * * *
* * * * * * * *      * * * * * * * *      * * * * * * * *
      * * * * * * * *
```

“SCHEDULE” (Agendamento)

- **GUIDED:** O número de iterações para cada thread irá variar, começando com um valor grande e sendo reduzido exponencialmente à medida que é enviado o tamanho do “loop” para cada thread. O tamanho do pedaço será o mínimo permitido para o número de iterações (por padrão é 1).

$$1^{\text{a}}\text{Th}=46/4\approx 11 \quad 2^{\text{a}}\text{Th}=(46-11)/4\approx 8 \quad 3^{\text{a}}\text{Th}=(46-11-8)/4\approx 6 \quad 4^{\text{a}}\text{Th}=(46-11-8-6)\approx 5 \dots$$



1

SCHEDULE (GUIDED, 3)

46



```
schedule(guided):
```

```
*****  
*****  
*****  
*****
```



```
schedule(guided, 2):
```

```
*****  
*****  
*****  
*****
```



```
schedule(guided, 4):
```

```
*****  
*****  
*****  
*****
```



```
schedule(guided, 8):
```

```
*****  
*****  
*****  
*****
```

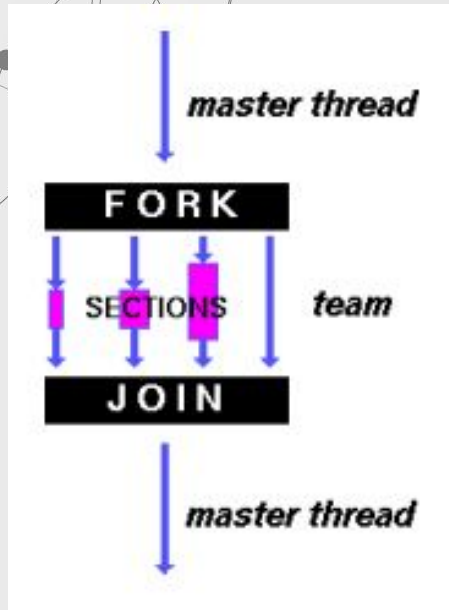


“SCHEDULE” (Agendamento)

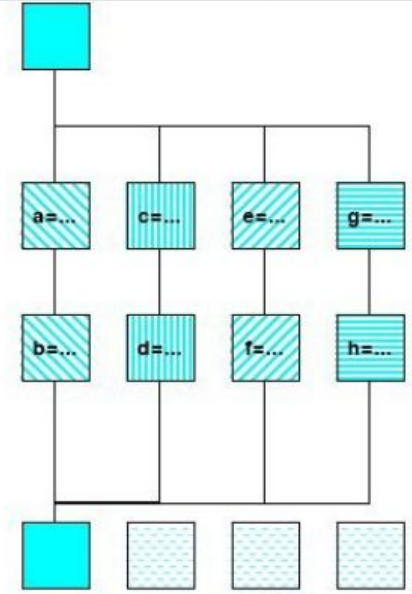
- **RUNTIME:** A decisão do agendamento é feita durante a execução do programa, com a definição da variável de ambiente `OMP_SCHEDULE`

OBS: As threads não serão sincronizadas no final do loop, continuando para os próximos comandos, caso o atributo `nowait` (C/C++) ou `NO WAIT` (Fortran) seja especificado.

Diretiva Sections



```
C / C++: #pragma omp parallel
{
  #pragma omp sections
  { { a=...;
      b=...; }
    #pragma omp section
    { c=...;
      d=...; }
    #pragma omp section
    { e=...;
      f=...; }
    #pragma omp section
    { g=...;
      h=...; }
  } /*omp end sections*/
} /*omp end parallel*/
```

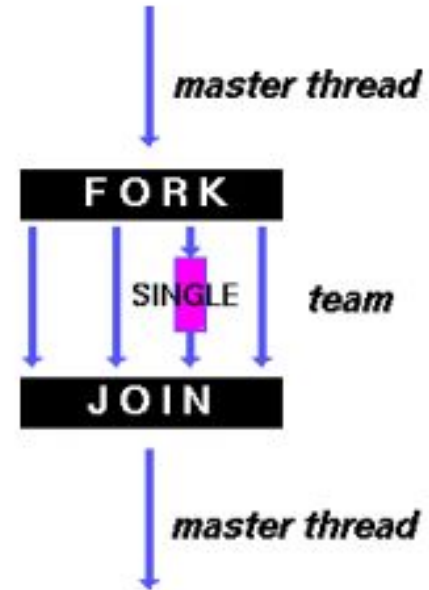


Existe um ponto de sincronização implícita no final da diretiva SECTIONS, a menos que, se especifique o atributo `nowait` (C/C++) ou `NOWAIT` (Fortran)

Se existirem mais Threads do que sections, o próprio OpenMP decide quais os threads irão realizar as tarefas e quais não

Diretiva Single

As threads que não executam a tarefa esperam pelo fim do processamento, a menos que o atributo `nowait` (C/C++) ou `NOWAIT` (Fortran) seja especificado





03

CONCEITOS BÁSICOS

Variáveis de Ambiente
Funções em Tempo de Execução
Construções Trabalho Compartilhado
Construções Combinadas
Exemplo Sections
Paralelismo Aninhado
Combinação Nested, Sections
Construções de Sincronização
Tarefa

C/C++

```
1 #ifdef _OPENMP
2   #include <omp.h>
3 #else
4   #define omp_get_num_threads() 1
5 #endif
```

Fortran

```
1 !$      use omp_lib
2          ....
3          nthreads = 1
4 !$      nthreads = omp_get_num_threads()
```

INICIA REGIÃO PARALELA

C/C++

Fortran

```
#pragma omp parallel [clause[[, clause]....] new-line  
structured block
```

```
!$omp parallel [clause[[, clause]....]  
structured block
```

```
!$omp end parallel
```

DEFINIR NÚMERO DE Threads

Função	<code>omp_set_num_threads()</code>
Variável Ambiente	<code>OMP_NUM_THREADS</code>

AJUSTAR NÚMERO DE Threads

Função	<code>omp_set_num_dynamic()</code>
Variável Ambiente	<code>OMP_DYNAMIC</code>

Threads ANINHADOS

Função	<code>omp_set_num_nested()</code>
Variável Ambiente	<code>OMP_NESTED</code>



VARIÁVEIS DE AMBIENTE

Definidas antes do programa começar. Podem ser alteradas por funções no programa.

Function name	Description
OMP_NUM_THREADS	Set the number of threads used.
OMP_SCHEDULE	Set the runtime scheduling type and chunk size.
OMP_DYNAMIC	Enable/disable dynamic thread adjustment.
OMP_NESTED	Enable/disable support for nested parallelism.



FUNÇÕES EM TEMPO DE EXECUÇÃO

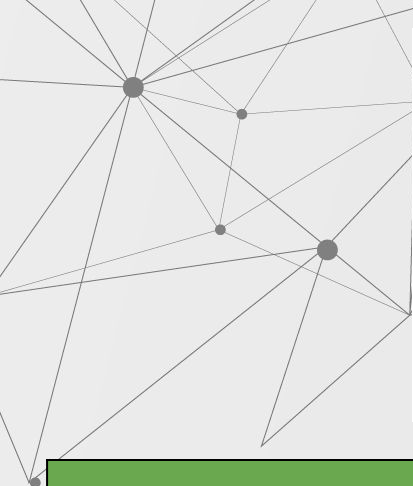
Podem alterar variáveis ambiente

Retornam o número de threads

Function name	Description
<code>omp_set_num_threads</code>	Set the number of threads.
<code>omp_get_num_threads</code>	Number of threads in the current team.
<code>omp_get_max_threads</code>	Number of threads in the next parallel region.
<code>omp_get_num_procs</code>	Number of processors available to the program.
<code>omp_get_thread_num</code>	Thread number within the parallel region.
<code>omp_in_parallel</code>	Check if within a parallel region.
<code>omp_get_dynamic</code>	Check if thread adjustment is enabled.
<code>omp_set_dynamic</code>	Enable, or disable, thread adjustment.
<code>omp_get_nested</code>	Check if nested parallelism is enabled.
<code>omp_set_nested</code>	Enable, or disable, nested parallelism.

Function name	Description
<code>omp_get_wtime</code>	Absolute wall-clock time in seconds.
<code>omp_get_wtick</code>	The number of seconds between successive clock ticks.

Marcam o tempo



```
1 x = 5;
2 y = 20;
3 #pragma omp parallel private(x) firstprivate(y)
4 {
5     x = 10;          // x is undefined on entry, but now set to 10
6     int z = x + y;    // y was pre-initialized to a value of 20
7     ....
8     y = 30;          // (first)private variables may be modified
9     ....
10 } // End of parallel region
```

Variáveis Privadas

Acessadas apenas pela memória do thread específico. Outros threads não interferem

Last Private

Copia Valor da última iteração do “loop” do thread que finalizou

First Private

Cada thread pré-inicializa o valor de y de antes de entrar na região paralela.

Shared

Conteúdo compartilhado com todos os thread do grupo

CONSTRUÇÕES TRABALHO COMPARTILHADO

Do/for

Designa o trabalho dos “loops” ao thread

Sections

Cada seção executada por um thread

Single

Uma secção da região paralela executada por um único thread

```
#pragma omp for [clause[,] clause]... new-line  
for-loop(s)
```

```
!$omp do [clause[,] clause]...  
do-loop(s)  
[!$omp end do [nowait/]
```

```
#pragma omp sections [clause[,] clause]... new-line  
{  
    [#pragma omp section ]  
        structured block  
    [#pragma omp section  
        structured block ]  
    ...  
}
```

```
!$omp sections [clause[,] clause]...  
[!$omp section ]  
    structured block  
[!$omp section  
    structured block ]  
    ...  
!$omp end sections [nowait/]
```

```
#pragma omp single [clause[,] clause]... new-line  
structured block  
!$omp single [clause[,] clause]...  
structured block  
!$omp end single [nowait,[copyprivate]]
```

CONSTRUÇÕES TRABALHO COMPARTILHADO

Workshare (apenas Fortran)

Paralelizar sintaxe de vetores

```
!$omp workshare  
    structured block  
!$omp end workshare [nowait]
```

Construção mestre

Relacionado ao Single, processo executa bloco de código mas sem barreira na saída

```
#pragma omp master new-line  
    structured block  
!$omp master  
    structured block  
!$omp end master
```


CONSTRUÇÕES COMBINADAS

Parallel Do/For

Determina região paralela e distribui as iterações do loop da região entre os thread do grupo

Parallel Sections

Determina região paralela e simultaneamente seções que cada thread irá executar

Full version	Combined construct
<pre>#pragma omp parallel { #pragma omp for for-loop }</pre>	<pre>#pragma omp parallel for for-loop</pre>
<pre>#pragma omp parallel { #pragma omp sections { [#pragma omp section] structured block [#pragma omp section structured block] ... } }</pre>	<pre>#pragma omp parallel sections { [#pragma omp section] structured block [#pragma omp section structured block] ... }</pre>
<pre>!\$omp parallel !\$omp do do-loop [\$omp end do] !\$omp end parallel</pre>	<pre>!\$omp parallel do do-loop !\$omp end parallel do</pre>
<pre>!\$omp parallel !\$omp sections [\$omp section] structured block [\$omp section structured block] ... !\$omp end sections !\$omp end parallel</pre>	<pre>!\$omp parallel sections [\$omp section] structured block [\$omp section structured block] ... !\$omp end parallel sections</pre>



EXEMPLO SECTIONS

```
1 #pragma omp parallel sections
2 {
3     #pragma omp section
4     {
5         for (int i=0; i<n; i++) {
6             (void) read_input(i);
7             (void) signal_read(i);
8         }
9     }
10    #pragma omp section
11    {
12        for (int i=0; i<n; i++) {
13            (void) wait_read(i);
14            (void) process_data(i);
15            (void) signal_processed(i);
16        }
17    }
18    #pragma omp section
19    {
20        for (int i=0; i<n; i++) {
21            (void) wait_processed(i);
22            (void) post_process_results(i);
23        }
24    }
25 } // End of parallel sections
```

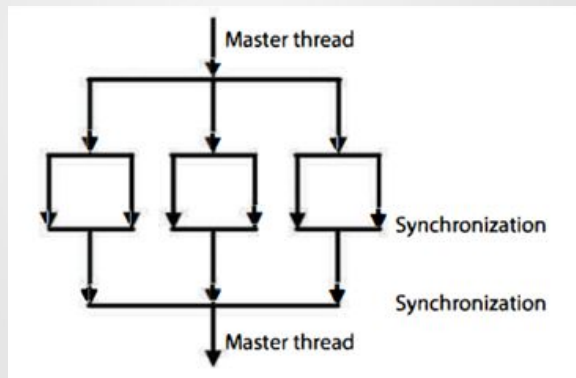
PARALELISMO ANINHADO

Paralelismo Aninhado “Parallel”

Cada thread na região inicia uma segunda região paralela.

```
1 #pragma omp parallel num_threads(3)
2 {
3     <code in here is executed by 3 threads>
4     #pragma omp parallel num_threads(2)
5     {
6         <code executed by 3x2 = 6 threads>
7     } // End of second level parallel region
8 } // End of first level parallel region
```

Cada região aninhada tem seu
processo Master



COMBINAÇÃO NESTED, SECTIONS

```
1 pragma omp parallel sections
2 {
3     #pragma omp section
4     { printf("I am section 1\n"); }
5     #pragma omp section
6     {
7         printf("I am section 2\n");
8         #pragma omp parallel for shared(n) num_threads(4)
9         for (int i=0; i<n; i++)
10        {
11            printf("Section 2:\tIteration = %d Thread ID = %d\n",
12                i, omp_get_thread_num());
13        } // End of parallel for loop
14    }
15    #pragma omp section
16    { printf("I am section 3\n"); }
17
18 } // End of parallel sections
```

CONSTRUÇÕES DE SINCRONIZAÇÃO

Barreira

Força os thread esperarem até todos tiverem atingido a região da barreira. Sincroniza os thread.

```
#pragma omp barrier new-line  
!$omp barrier
```

Construção Crítica

Região executada por todos os thread mas apenas por um processo de cada vez.

```
#pragma omp critical [(name)] new-line  
    structured block  
!$omp critical [(name)]  
    structured block  
!$omp end critical [(name)]
```

Atômico

Enquanto um processo está alterando lugar na memória nenhum outro pode alterar tal lugar.

```
#pragma omp atomic new-line  
    statement  
!$omp atomic  
    statement
```



CONSTRUÇÕES DE SINCRONIZAÇÃO

Ordenamento

Thread são executados em forma serial na ordem do loop.


```
#pragma omp ordered new-line  
structured block
```

```
!$omp ordered  
structured block  
!$omp end ordered
```

Flush

Atualização dos dados compartilhados entre os thread.

```
#pragma omp flush [(flush-set)] new-line  
!$omp flush [(flush-set)]
```



```
1 #pragma omp parallel
2 {
3     *****
4     #pragma omp atomic
5         x += 1;
6     *****
7 } // End of parallel region
```

```
1 #pragma parallel
2 {
3     ...
4     #pragma omp critical (c_region1)
5     {
6         sum1 += ...
7     }
8     ...
9     #pragma omp critical (c_region2)
10    {
11        sum2 += ...
12    }
13    ...
14 } // End of the parallel region
```



TAREFA

Tarefa

Região paralela escolhe uma tarefa da lista e a executa.

```
1 #pragma omp parallel
2 {
3     #pragma omp single
4     {
5         #pragma omp task
6         {printf("race ");} // Task #1
7         #pragma omp task
8         {printf("car ");} // Task #2
9     } // End of single region
10 } // End of parallel region
```

04

PRÓS E CONTRAS

Prós
Contras
Cuidados - False Sharing
Cuidados - Race Condition

PRÓS

- Facilidade de conversão de programas sequenciais em paralelos.
- O OpenMP é uma maneira simples de explorar o paralelismo.
- Minimiza a interferência na estrutura do algoritmo.
- Permite a execução das tarefas em menor tempo, através da execução em paralelo.

PRÓS

- Requer uma menor quantidade de modificações no código em comparação com o MPI.
- As diretivas do OpenMP podem ser tratadas como comentários se o OpenMP não estiver disponível.
- As diretivas podem ser adicionadas de forma incremental.
- Compila e executa em ambientes paralelo e sequencial.

CONTRAS

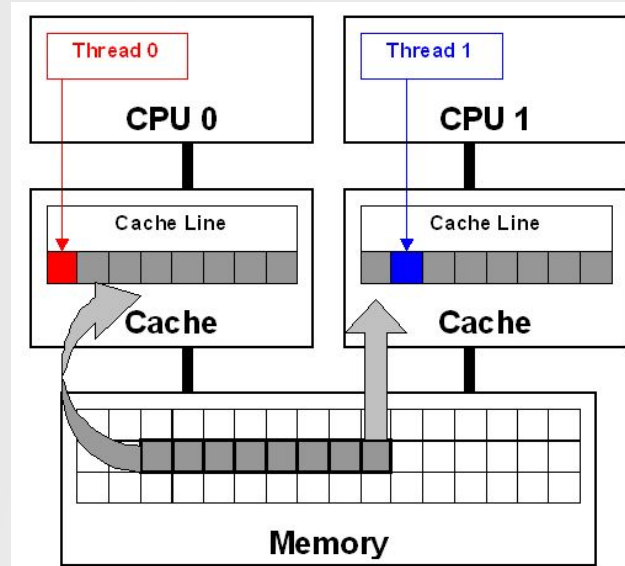
- O OpenMP é limitado pelo número de processadores disponíveis em um único computador.
- Requer um compilador que tenha suporte ao OpenMP.
- Os códigos OpenMP não podem ser executados em computadores com memória distribuída (com exceção do OpenMP da Intel).
- Falta um tratamento confiável de erros.
- Carece de mecanismos refinados para controlar o mapeamento do processador de thread.

CUIDADOS

False Sharing

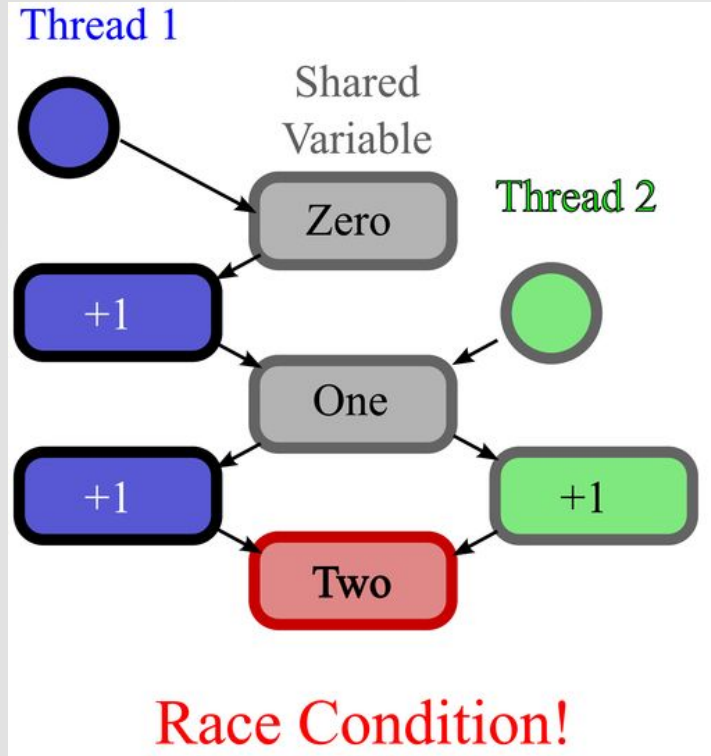
- Causado pelas múltiplas tarefas atualizando dados na mesma linha do cache.
- Ocorre quando:
 - Os dados compartilhados são modificados por vários processadores.
 - Vários processadores atualizam dados na mesma linha de cache.
 - Essa atualização ocorre com muita frequência (por exemplo em um loop fechado).

```
double sum=0.0, sum_local[NUM_THREADS];  
#pragma omp parallel num_threads(NUM_THREADS)  
{me = omp_get_thread_num();  
  sum_local[me] = 0.0;  
  
  #pragma omp for(i = 0; i < N; i++)  
    sum_local[me] += x[i] * y[i];  
  
  #pragma omp atomic+= sum_local[me];  
}
```



CUIDADOS

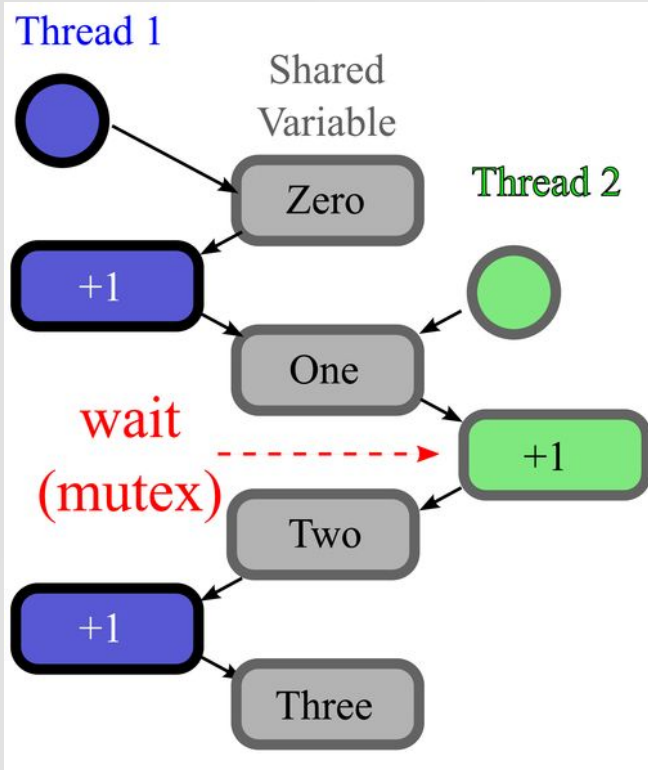
Race Condition



- Problema muito comum de ser encontrado na programação de memória compartilhada.
- Os resultados não são determinísticos no modo paralelo.
- Ocorre quando:
 - Múltiplos threads acessam o mesmo local de memória simultaneamente.
 - Dependência de dados transportados por loop.

CUIDADOS

Race Condition



```
1 int sharedVariable = 0;
2 #pragma omp parallel for
3 for (int i = 0; i < n; i++){
4     #pragma omp critical
5     { // Heavy-weight critical section mutex:
6         sharedVariable++; // Only one thread at a time will enter this scope
7     }
8     // Alternatively, a light-weight atomic mutex:
9     #pragma omp atomic
10    shared_variable++; // Only one thread at a time increments shared_variable
11 }
```

05

HANDS ON

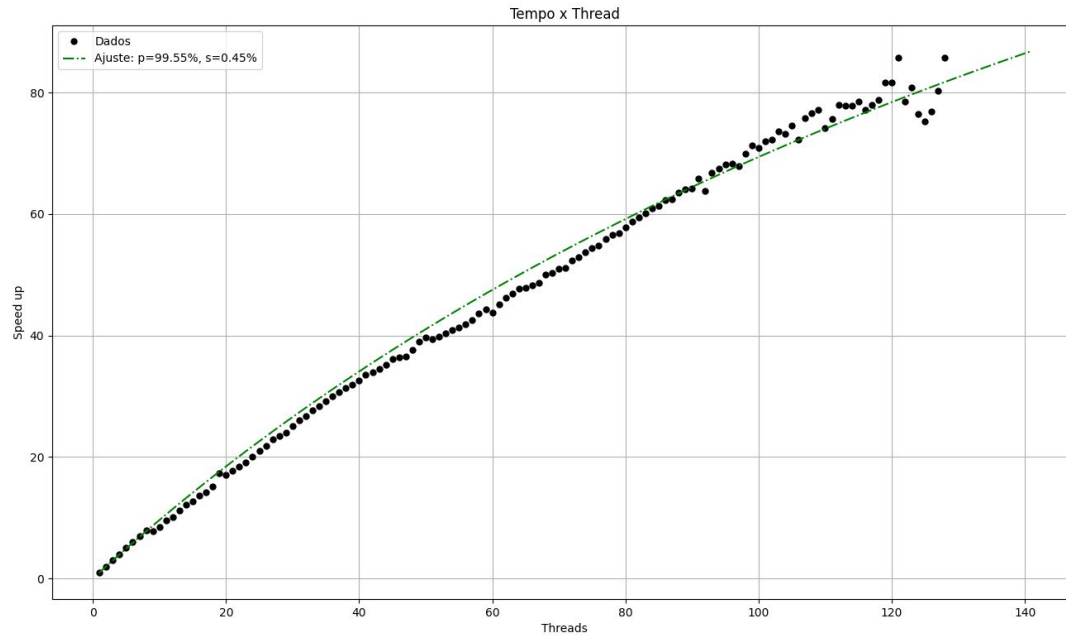
Google Colab

Link:

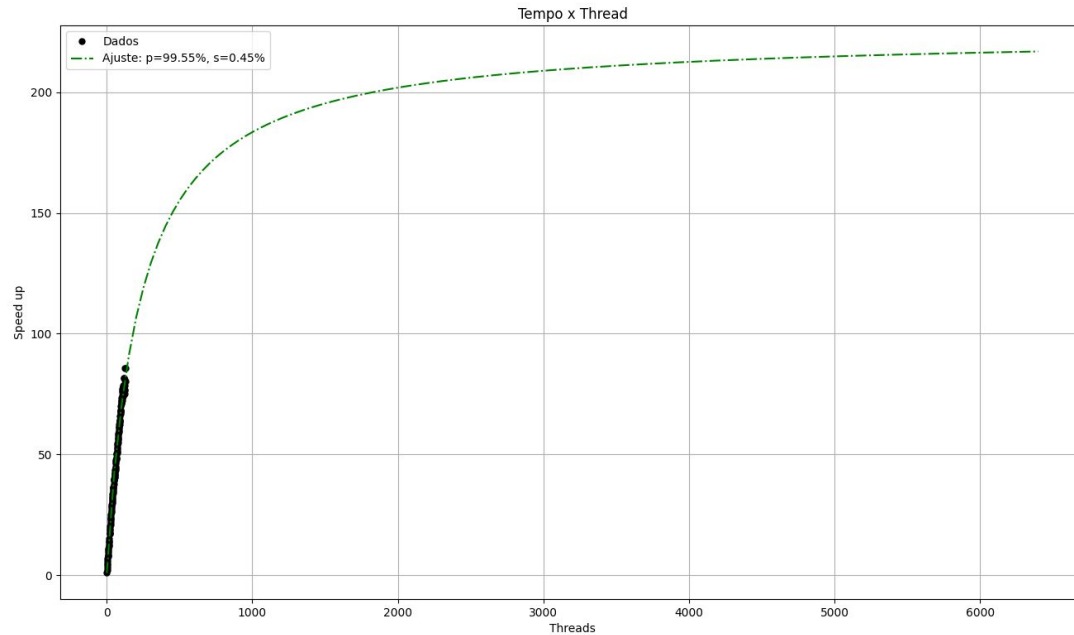
- <https://colab.research.google.com/drive/1zRvwaDACp0-nG4zqxx7KGebF-VE5MwyV?usp=sharing>
- shorturl.at/quzNW

Clicar em **Google Colaboratory** => **Arquivo** => **Salvar uma cópia no drive**

Lei de Amdahl para a Agulha de Buffon



Lei de Amdahl para a Agulha de Buffon





06

BIBLIOGRAFIA

Referências bibliográficas

- **Advanced OpenMP Topics** - NAS Webinar 2012. Disponível em:
<https://cac-staff.github.io/summer-school-2018/files/OpenMP.pdf>
- **Introdução ao OpenMp** - Universidade Estadual de Campinas Centro Nacional de Processamento de Alto Desempenho São Paulo - Disponível em:
https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_openmp.pdf
- **Introdução ao OpenMP- SILVA F.** - DCC FCUP - Disponível em:
http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/intro_openmp-Fernando-Silva.pdf
- **Introduction to OpenMP. - MATTSON, T.** Disponível em:
<https://www.youtube.com/watch?v=x0HkbluJILk&list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG&index=5>
- **Programação em Memória Partilhada com o OpenMP - ROCHA, R.** - Universidade do Porto - 2015/2016 - Disponível em:
https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides_openmp.pdf
- **Shared-Memory Programming With OpenMP**-2018 Ontario HPC Summer School Hartmut Schmider Centre for Advance Computing, Queen's University July/August 2017. Disponível em:
<https://cac-staff.github.io/summer-school-2018/files/OpenMP.pdf>

Referências bibliográficas

- **Simple Tutorial with OpenMP** - . Algorithms, Blockchain and Cloud. Disponível em:
<https://helloacm.com/simple-tutorial-with-openmp-how-to-use-parallel-block-in-cc-using-openmp/>
- **OpenMp** - The OpenMP API specification for parallel programming - Disponível em:
<https://www.openmp.org/>
- **Programação em Memória Compartilhada com OpenMP** - Orellana, E. T. V. - Universidade Estadual de Santa Cruz - Disponível em:
http://nbcgib.uesc.br/nbcgib/files/PP/Aula05_OpenMP.pdf
- **What is OpenMP?** - Dartmouth College - Disponível em:
https://www.dartmouth.edu/~rc/classes/intro_openmp/print_pages.shtml
- **An Introduction to OpenMP** - Chandler, C.; Chhetri, H.. - Louisiana Tech University - Los Alamos National Laboratory - Disponível em:
<http://www2.latech.edu/~box/hapc/openmp.pdf>
- **OpenMP: Uma Introdução** - Geyer, C. - Universidade Federal do Rio Grande do Sul - Informática UFRGS - Disponível em:
<ftp://ftp.inf.ufrgs.br/pub/geyer/PDP-CIC-ECP/slidesAlunos/SemestresAnteriores/ProvaP2-2012/openmp-intro-v5-1-mai2012-mac.pdf>

Referências bibliográficas

- **Disciplina de Arquitetura de Computadores** - Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação - Disponível em:
<http://wiki.icmc.usp.br/images/0/0c/SSC0510-Aula12.pdf>
- **Arquiteturas Paralelas - Amory, A.; Moreno, E.** - Disponível em:
https://www.inf.pucrs.br/~emoreno/undergraduate/SI/orgarg/class_files/Aula17.pdf
- ***What Is False Sharing?*** - Oracle Corporation and/or its affiliates - Disponível em:
<https://docs.oracle.com/cd/E19205-01/819-5270/aewcy/index.html>
- ***Avoiding and Identifying False Sharing Among Threads*** - Intel Corporation - Development Topics and Technologies - Disponível em:
<https://software.intel.com/content/www/us/en/develop/articles/avoiding-and-identifying-false-sharing-among-threads.html>
- ***Data Race*** - Supercomputing Center of USTC - Disponível em:
https://scc.ustc.edu.cn/zlsc/sugon/intel/ssadiag_docs/pt_reference/references/sc_omp_anti_dependence.htm
- ***Multi-Threading and Parallel Reduction*** - TechEnablement - Disponível em:
<https://www.techenablement.com/intel-xeon-phi-optimization-part-1-of-3-multi-threading-and-parallel-reduction/>

Referências bibliográficas

- ***Using Open MP - The Next Step*** Ruud van der Pas, Eric Stotzer, and Christian Terboven - The MIT Press Cambridge, Massachusetts 2017
- ***OpenMP Barrier - Jaka Speh***, Disponível em:
<http://jakascorner.com/blog/2016/07/omp-barrier.html>



CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.

Please keep this slide for attribution.
