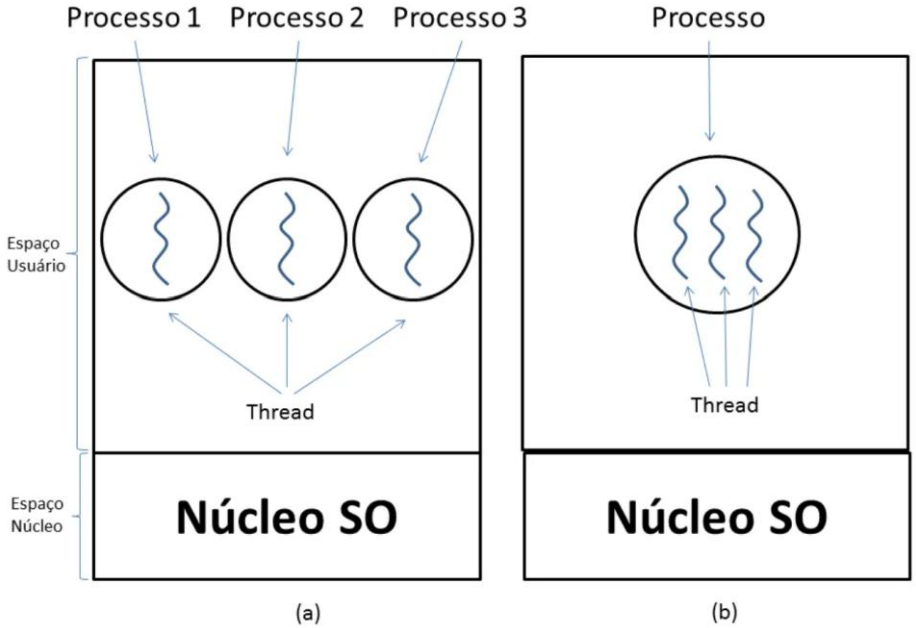
Insper

Supercomputação

Lícia Sales Costa Lima

Threads

- Uma thread é um processo “peso leve”.
- As threads podem compartilhar dados com outras threads, mas também têm dados privados.
- As threads se comunicam através de uma área de dados compartilhada.
- threads podem cooperam em uma tarefa.
- A “thread master” é responsável pela coordenação de outras threads.



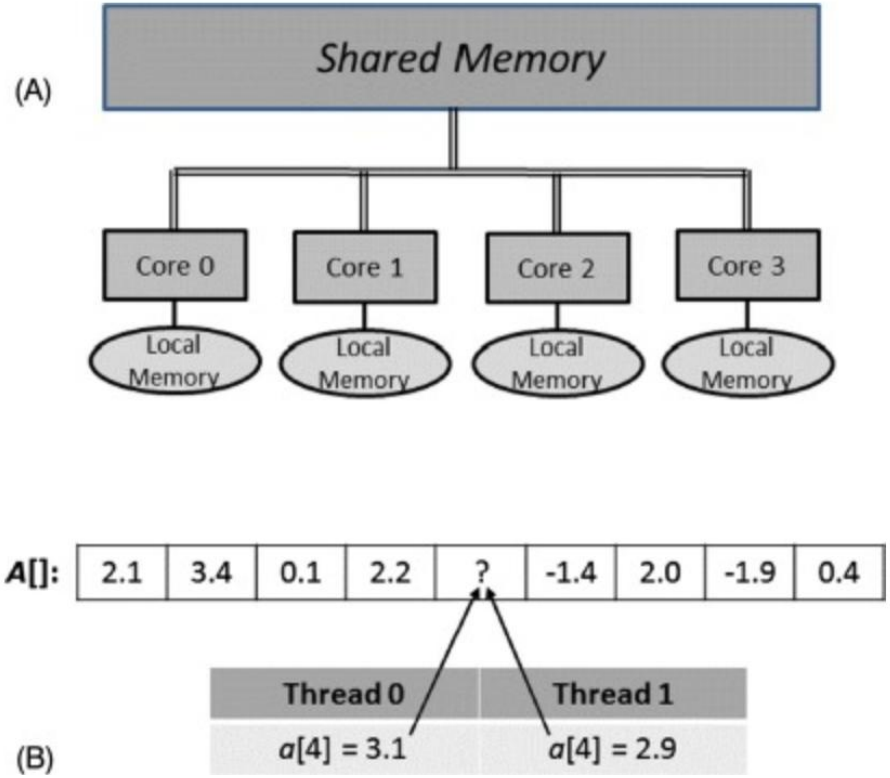
Fonte: https://thiagofelippi.medium.com/concorr%C3%AAncia-paralelismo-threads-e-processos-s-o-parte-4-299a574bc663

Shared

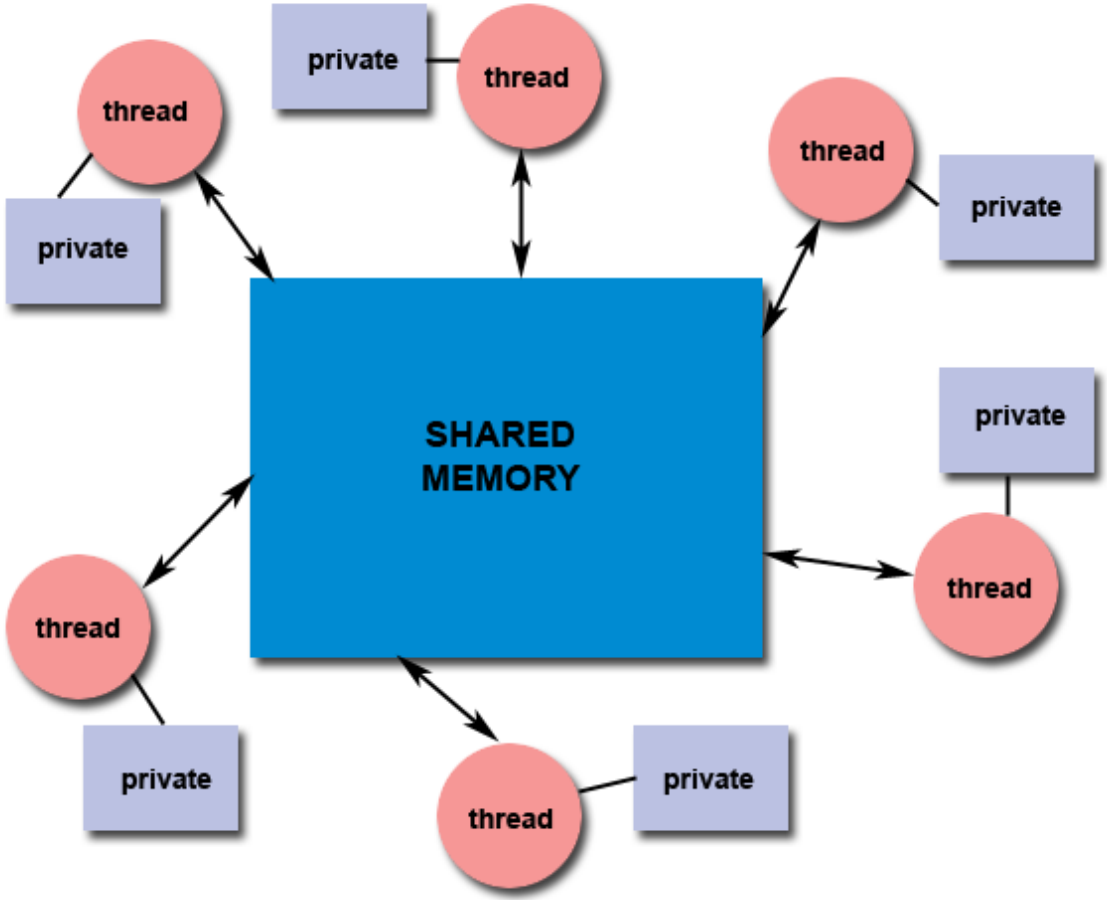
Variável Global, todas as threads acessam o endereço

Private

Variável Local, apenas a thread alocada acessa o endereço

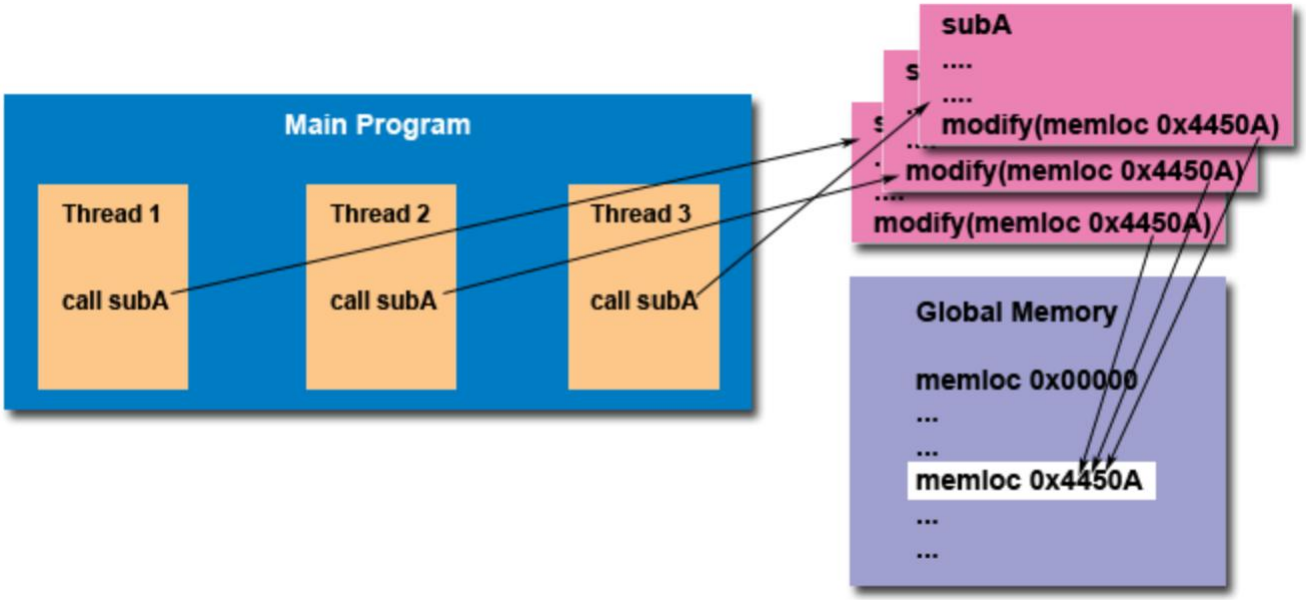


Fonte: https://www.sciencedirect.com/topics/computer-science/shared-memory-system



Fonte: https://hpc-tutorials.llnl.gov/posix/designing_threaded_programs/

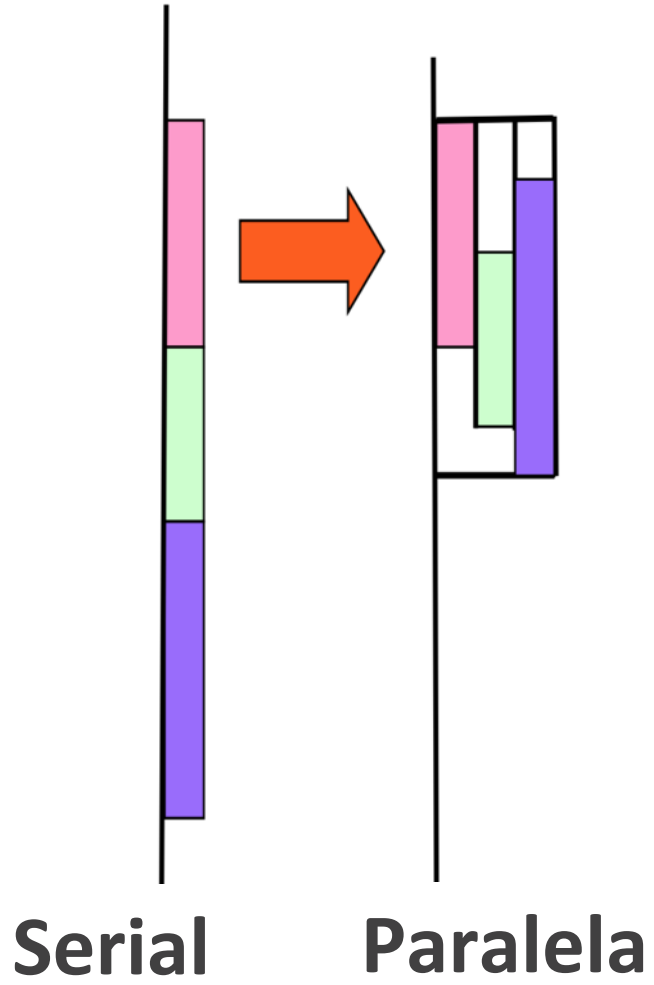
Efeitos Colaterais do paralelismo



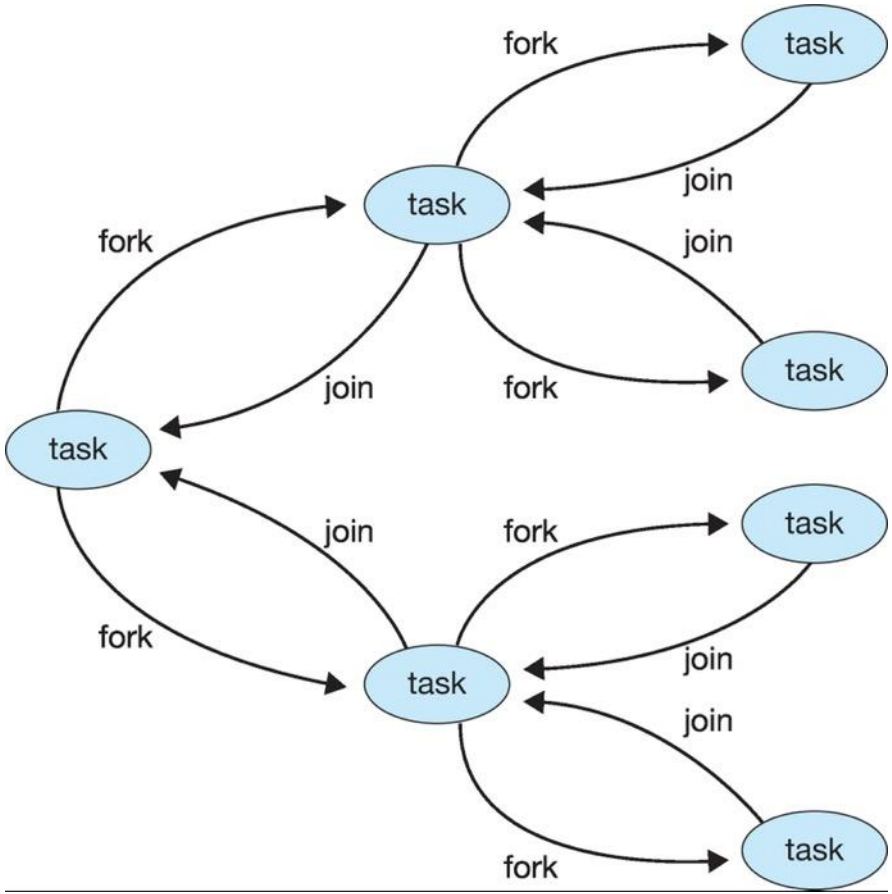
Fonte: https://hpc-tutorials.llnl.gov/posix/designing_threaded_programs/

O que são tarefas (tasks)?

- A tarefa é definida em um bloco estruturado de código
- Tarefas podem ser aninhadas: isto é, uma tarefa pode gerar outras novas tarefas
- Cada thread pode ser alocada para rodar uma tarefa
- Não existe ordenação no início das tarefas
- Tarefas são unidades de trabalho independentes

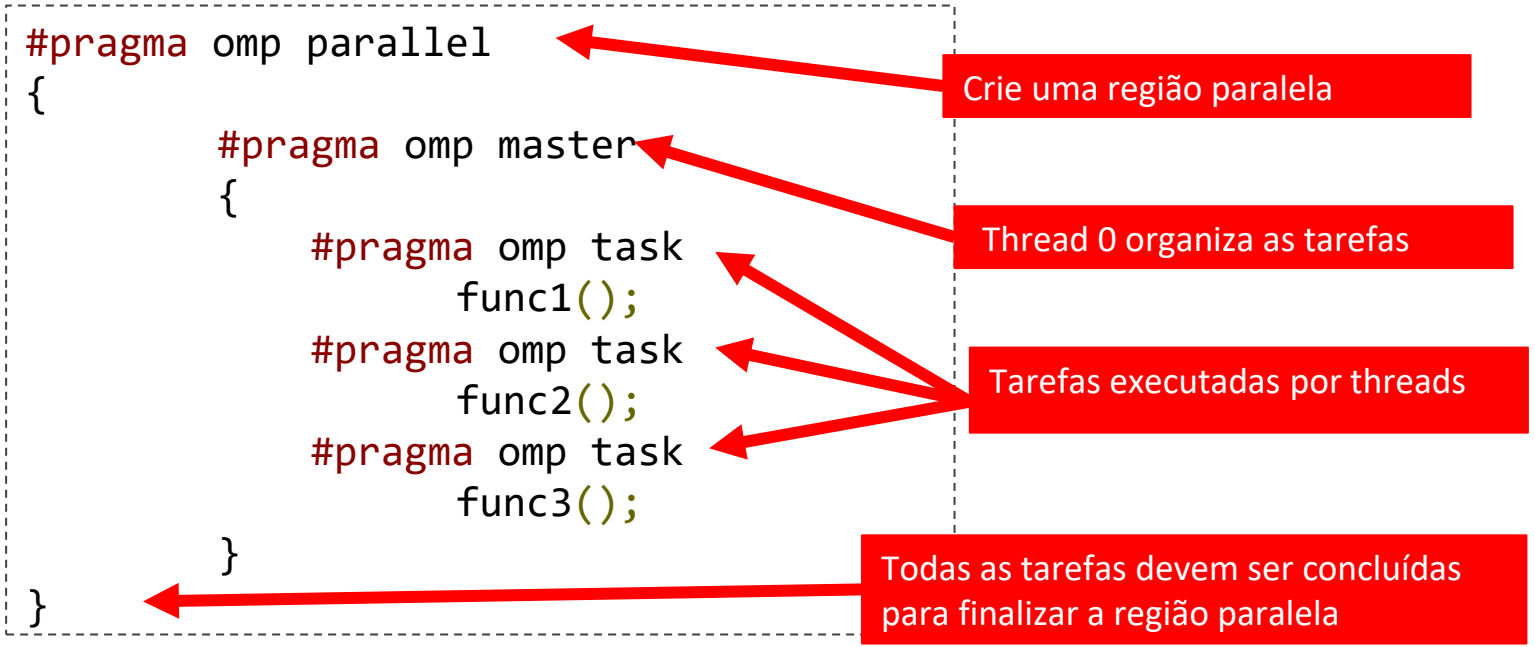


Tasks / Fork-Join



Tarefas em OpenMP

#pragma omp task[clauses]



Esperando (taskwait)

```
#pragma omp parallel
{
  #pragma omp single
   {
     #pragma omp task
        fred();
     #pragma omp task
        daisy();
     #pragma taskwait
     #pragma omp task
        billy();
   }
}
```

billy() só será executado quando **fred()** e **daisy()** finalizarem suas tarefas

Sections

- A diretiva sections divide o trabalho de forma não iterativa em seções separadas, onde cada seção será executada por uma “thread” do grupo.

- Algumas observações:

  - A diretiva sections define a seção do código sequencial onde será definida as seções independentes, através da diretiva section;

  - Cada section é executada por uma *thread* do grupo;

  - Existe um ponto de sincronização implícita no final da diretiva section, a menos que se especifique o atributo **nowait**;

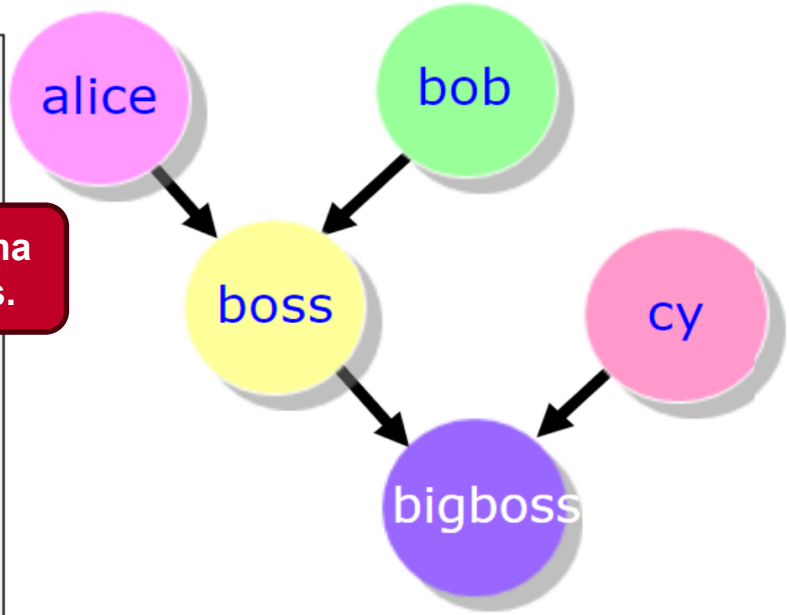
  - Se existirem mais *threads* do que seções, o OpenMP decidirá, quais *threads* executarão os blocos de section, e quais, não executarão.

```
#pragma omp parallel sections
{
#pragma omp section
    double a = alice();
#pragma omp section
    double b = bob();
#pragma omp section
    double c = cy();
}

 double s = boss(a, b);
 printf ("%6.2f\n", bigboss(s,c));
```

Definição de uma área de seções.

Seções



Fonte: https://www.serc.iisc.ac.in/serc_web/wp-content/uploads/2019/06/AP-OpenMPDay2.pdf

Fibonacci

Dividir para conquistar!.

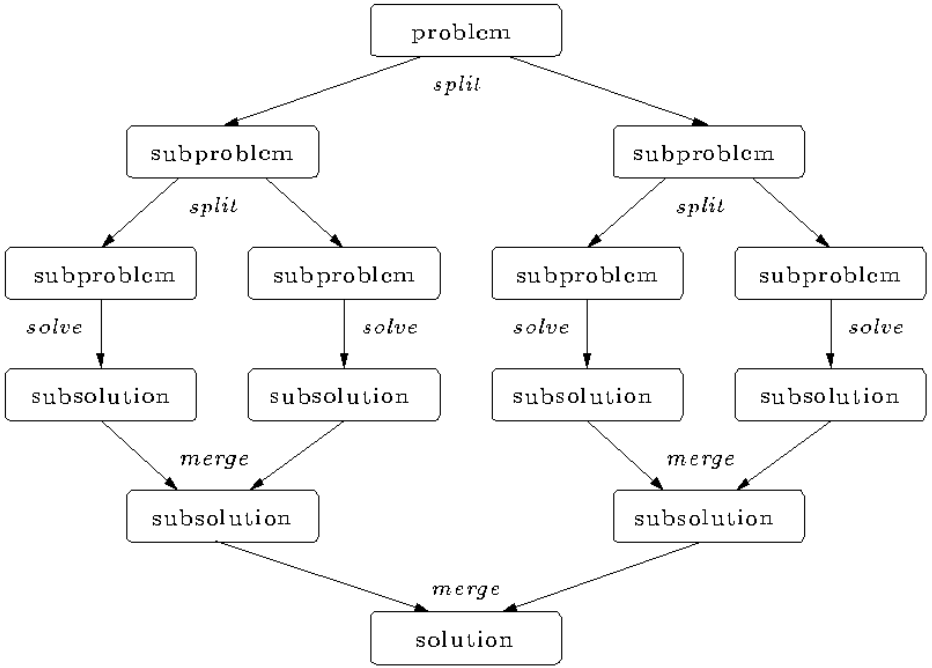
```
#include<iostream>
#include<omp.h>
using namespace std;


int fib (int n) {
    int x, y;
    if(n<2) return n;
    x = fib(n-1);
    y = fib(n-2);
    return (x+y);
}

int main() {
    int NW = 1000;
    float time = omp_get_wtime();
    fib(NW);
    time = omp_get_wtime() - time;
    cout << "Tempo em segundos : " << time << endl;

}
```

Resolução




```
#include<iostream>
#include<omp.h>
#include <math.h>
using namespace std;

int fib (int n) {
    int x, y;
    if(n<2) return n;
    if (n < 20) {
        return fib(n-1) + fib(n-2);
    } else {
    #pragma omp task shared(x)
    x = fib(n-1);
    #pragma omp task shared(y)
    y = fib(n-2);
    #pragma omp taskwait
    return x+y;
    }
}

int main() {
    int NW = 50;
    float time;
    time = omp_get_wtime();
    #pragma omp parallel
    {
        #pragma omp single
        fib(NW);
    }
    time = omp_get_wtime() - time;
    cout << "Tempo em segundos : " << time << endl;
}
```

OpenMP

- Missão de hoje: seguir o roteiro da aula