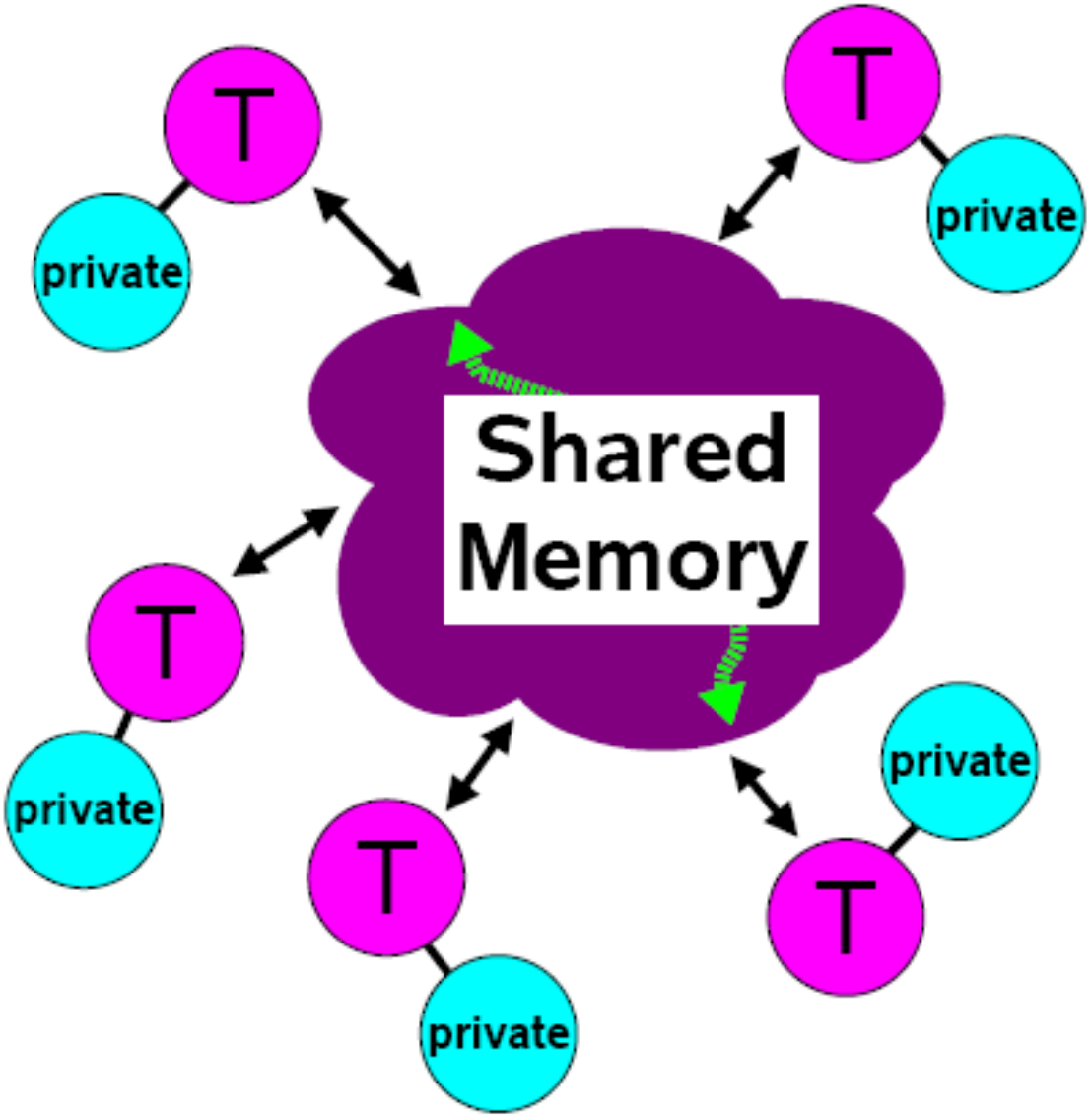
OpenMP

Computação Paralela e Distribuída

www.acso.uneb.br/leandro
leandrocoelho@uneb.br

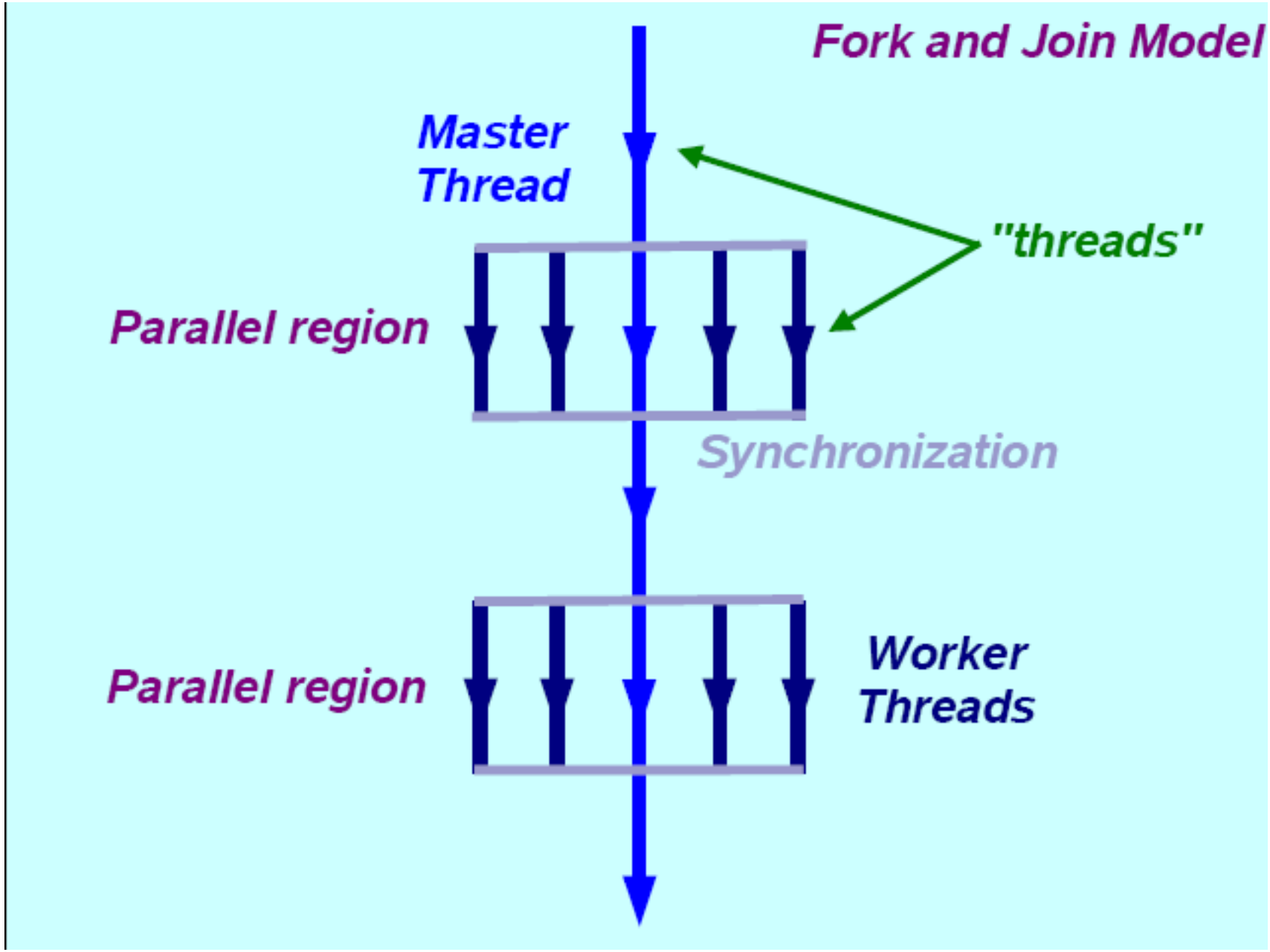
www.muriloboratto.com.br
muriloboratto@uneb.br

Shared Memory Model



Noções básicas OpenMP

- Ferramenta de programação para **memória compartilhada**.
- Modelo de programação ***fork-join***, com geração de múltiplas threads.
- Inicialmente executa-se um thread até que aparece o primeiro ***construtor paralelo***, e criam-se as outras threads.
- Ao final do ***construtor*** se sincronizam as threads e continuam a execução.


Fork and Join Model
Master Thread
"threads"
Parallel region
Synchronization
Parallel region
Worker Threads

Noções básicas OpenMP

- OpenMP é formado por:
  - **Construtores:** indicam como distribuir o trabalho, gerenciar as threads, sincronizar...
  - **Funções:** para estabelecer, obter e comprovar valores
  - **Variáveis de ambiente:** indicam a forma da execução

OpenMP – Exemplo Básico

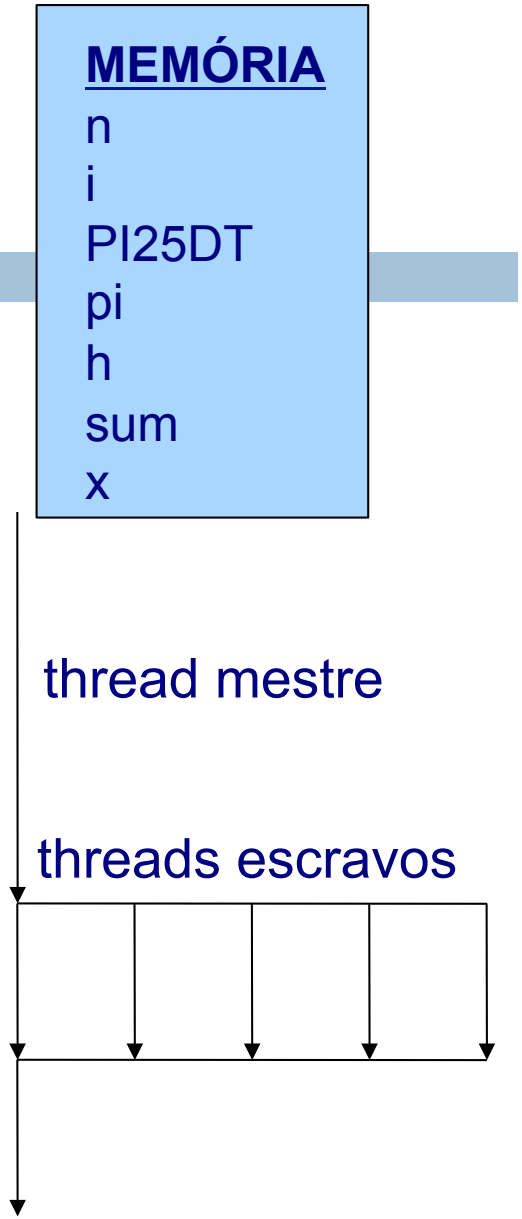
- Funcionamento
  - Exemplo:

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv)
   {
     double vet[1000];
     for (int i = 0; i <1000; i++)
     {
         Calcular(vet[1000]);
     }
}
```

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char **argv)
   {
     double vet[1000];
     #pragma omp parallel for
     for (int i = 0; i <1000; i++)
     {
         Calcular(vet[1000]);
     }
}
```

Exemplo 3-16


```
#include <stdio.h>
#include <math.h>
#include <omp.h>
double f(double x) {   return sqrt(1 - pow(x, 2.)); }
int main(int argc, char *argv[]) {
    int  n, i;
    double PI25DT = 3.14159265358979323846264643, pi, h, sum, x;
    printf("Numero de intervalos a usar: ");
    scanf("%d",&n);
    h   = 1.0 / (double) n;
    sum = 0.0;
    #pragma omp parallel for reduction(+:pi) private(x, i)
    for (i = 1; i <= n; i ++) {
        x = h * ((double)i - 0.5);
        pi += f(x); }
    pi = 4. * h * pi;
    printf("\npi es aproximadamente %.16f, el error es %.16f\n", pi, fabs(pi - PI25DT));
}
```



Exemplo

Compilação:

icc codigo.c -o objeto -openmp

gcc codigo.c -o objeto -fopenmp

Execução:

```
$> export OMP_NUM_THREADS=1
$>time ./codigo3-16 < in10000000
Numero de intervalos a usar:
pi é ...
real    0m0.240s
user    0m0.240s
```

```
$> export OMP_NUM_THREADS=4
$> time ./codigo3-16 < in10000000
Numero de intervalos a usar:
pi é ...
real    0m0.094s
user    0m0.240s
```

Construtores (pragma)

Sintaxe:

#pragma omp nome [cláusulas]

Construtor parallel

#pragma omp parallel [cláusulas]

bloco

- ✓ **Cria-se um grupo de threads.**
- ✓ **O número de threads se obtém por variables de ambiente ou funções de bibliotecas.**
- ✓ **Há barreira implícitas ao final dessa região**

Construtor parallel

- **Quando dentro de uma região há outro construtor paralelo, cada escravo criaria outro grupo de threads escravos dos que seria o mestre.**

- **Há uma série de cláusulas (private, firstprivate, default, shared, copyin e reduction) para indicar a forma em que se acessa as variáveis.**

OpenMP

- Clausulas OpemMP para Parallel
  - private(list)
  - firstprivate(list)
  - shared(list)
  - default(shared|none)
  - num_threads(int_exp)
  - if(exp)
  - copyin(list)
  - reduction(operator: list)

OpenMP

- Clausulas OpemMP
  - private(list)
  - firstprivate(list)
  - shared(list)
  - default(shared|none)
  - num_threads(int_exp)
  - if(exp)
  - copyin(list)
  - reduction(operator: list)

Possibilitam ao usuário o controle dos âmbitos das variáveis na região paralela!

OpenMP

- Clausulas OpemMP
  - private(list)
    - As variáveis da lista ficam privadas a cada thread criado
    - Não são incializadas
  - firstprivate(list)
    - Permite que as variáveis privadas sejam inicializadas
  - shared(list)
    - As variáveis da lista são compartilhadas entre os threads
    - O padrão define as variáveis como “shared”

OpenMP

- Clausulas OpenMP
  - num_threads(int_exp)
    - Define o número de threads a serem criadas
  - if(exp)
    - Avalia se condição é verdadeira para criar os threads.
  - copyin(list)
  - reduction(operator: list)
    - Permite operar sobre as variáveis da lista

OpenMP

- Código Exemplo:

```
#include <omp.h>
int main(int argc, char **argv)
{
    int nthreads, tid;
    #pragma omp parallel private(nthreads, tid)
    { // Descobre e imprime o id do Thread
       tid = omp_get_thread_num();
       printf("Alo da thread = %d\n", tid);
       if (tid == 0) // apenas o master thread faz isto
       {
          nthreads = omp_get_num_threads();
          printf("Msg do Master:Existem + %d\n", nthreads);
       }
    } //Sincronismo de todos os threads
}
```

OpenMP

- Construtores de Work-Sharing
  - Especificam regras de divisão de trabalho entre os threads, ***não cria os threads***!
    - Tipos de compartilhamento
      - **for** – Compartilha as iterações de um ciclo pelos threads
        - Paralelismo de dados
      - **sections** – Divide o trabalho em seções discretas, distintas, que são executadas pelos threads.
        - Pode ser usado para paralelismo funcional.
      - **single** – serializa o código

OpenMP

- Construtores de Work-Sharing
  - **for**
    - **#pragma omp for 'clause'**

```
{
  Loop for();
}.
```

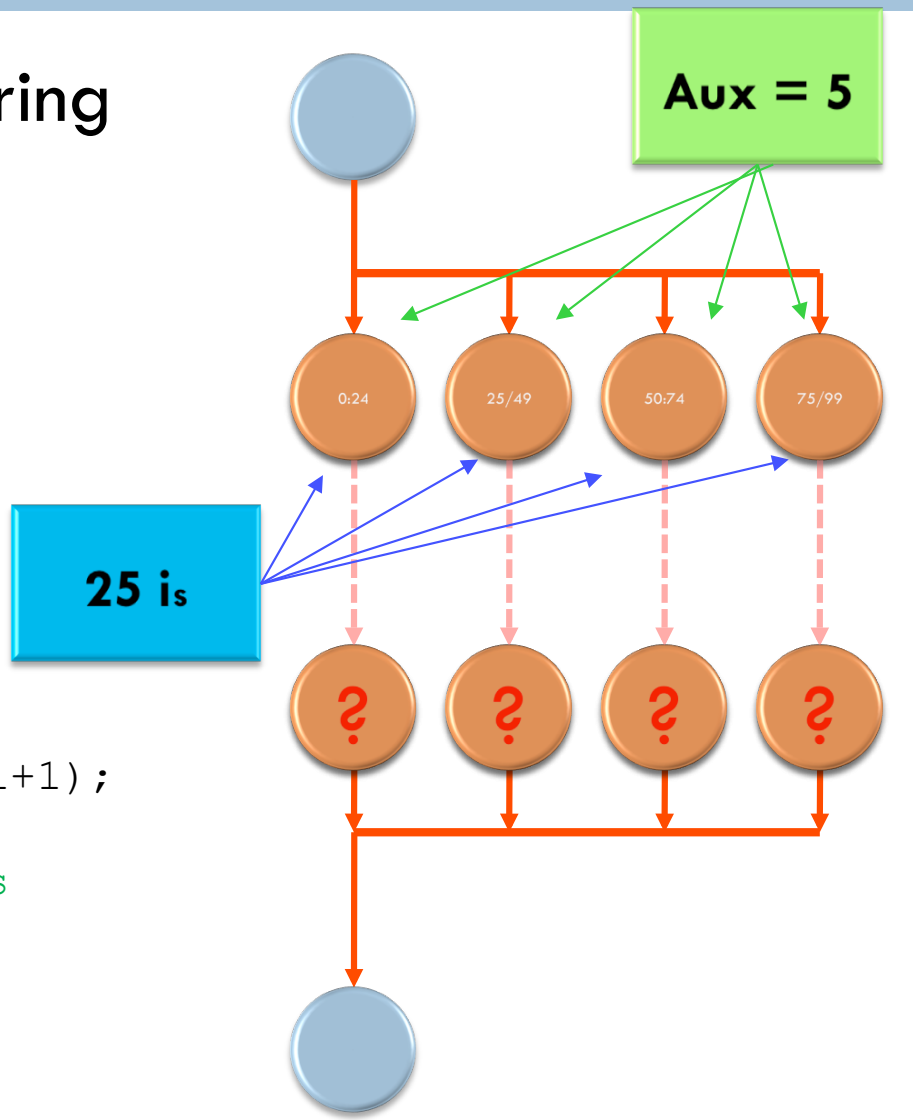
OpenMP

- Construtores de Work-Sharing
  - Clausulas para o **"for"**
    - private(*list*)
    - firstprivate(*list*)
    - lastprivate(*list*)
    - reduction(*operator: list*)
    - ordered
    - schedule(*type*)
    - nowait

OpenMP

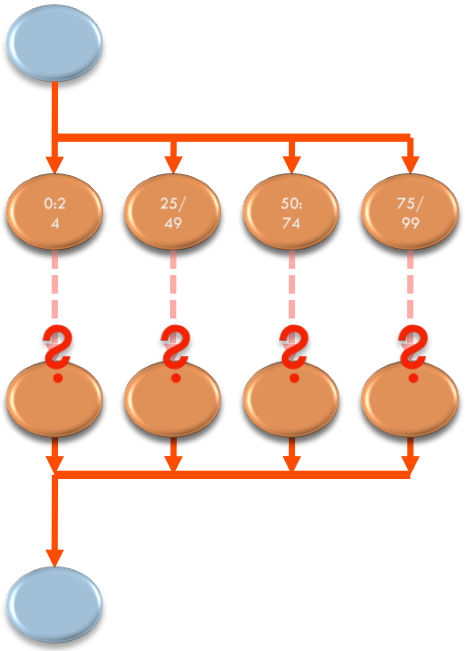
- Construtores de Work-Sharing
  - Exemplo **for**:

```
#include <omp.h>
int main(int argc, char **argv){
     int aux;
     #pragma omp parallel private(aux)
     {
        aux = 5;
        #pragma omp for
        {
          for (int i=0; i<100; i++)
            vet_a[i]= vet_b[i] + aux *(i+1);
        }
     } //Sincronismo de todos os threads
}
```



OpenMP

- Construtores de Work-Sharing
  - Clausulas para o **"for"**
  - A questão agora é:
    - É possível o balanceamento de carga de uma clausula for?



OpenMP

- Construtores de Work-Sharing
  - Clausulas para o **"for"**
    - Schedule :
      - Indica a forma como se dividem os repetições do for entre os threads
        - **static** – as iterações são agrupadas em conjuntos (*chunks*), estaticamente atribuídos aos threads.
          - `schedule(static,chunk)`
        - **dynamic** – as iterações são agrupadas em *chunks* e são dinamicamente distribuídos pelos threads; quando um Thread termina, recebe dinamicamente outro *chunk*.
          - `schedule(dynamic,chunk)`

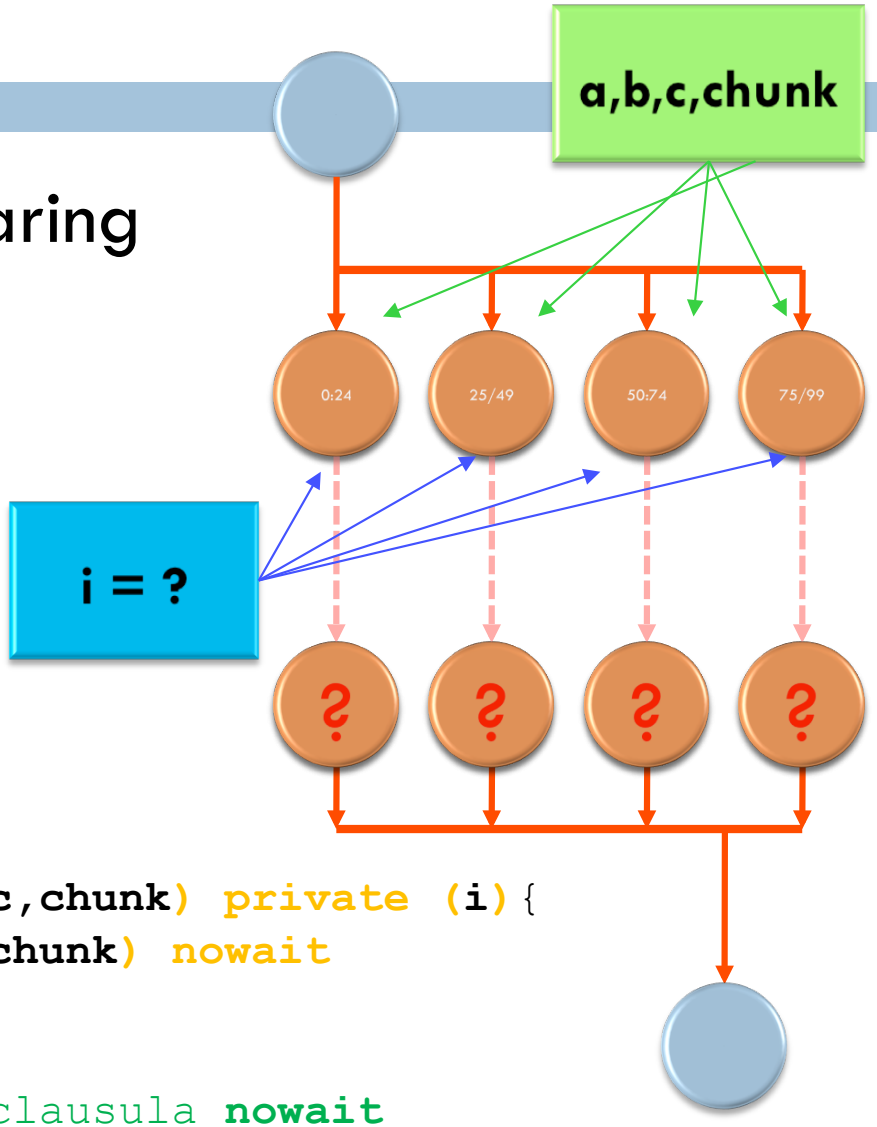
OpenMP

- Construtores de Work-Sharing
  - Clausulas para o **"for"**
    - Schedule :
      - Indica a forma como se dividem os repetições do for entre os threads
        - **guided** – indica o número mínimo de iterações a agrupar numa tarefa;
          - `schedule(guided,chunk)`
        - **runtime** – a decisão é tomada em tempo de execução a partir da variavel de ambiente: OMP_SCHEDULE
          - `schedule(runtime)`

OpenMP

- Construtores de Work-Sharing
  - Exemplo **for**:

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000
int main(int argc, char **argv){
     int i, chunk;
     float a[N], b[N], c[N];
     for (i=0; i < N; i++)
          a[i] = b[i] = i * 1.0;
     chunk = CHUNKSIZE;
     #pragma omp parallel shared (a,b,c,chunk) private (i){
     #pragma omp for schedule(dynamic,chunk) nowait
          for ((i=0; i < N; i++)
               c[i] = a[i] + b[i];
     } //Sinc. dos threads ? Observar clausula nowait
}
```



OpenMP

- Construtores de Work-Sharing
  - Especificam regras de divisão de trabalho entre os threads, ***não cria os threads***!
    - Tipos de compartilhamento
      - **for** – Compartilha as iterações de um ciclo pelos threads
        - Paralelismo de dados
      - **sections** – Divide o trabalho em seções discretas, distintas, que são executadas pelos threads.
        - Pode ser usado para paralelismo funcional.
      - **single** – serializa o código

OpenMP

- Construtores de Work-Sharing
  - **sections**
    - **#pragma omp sections 'clause'**

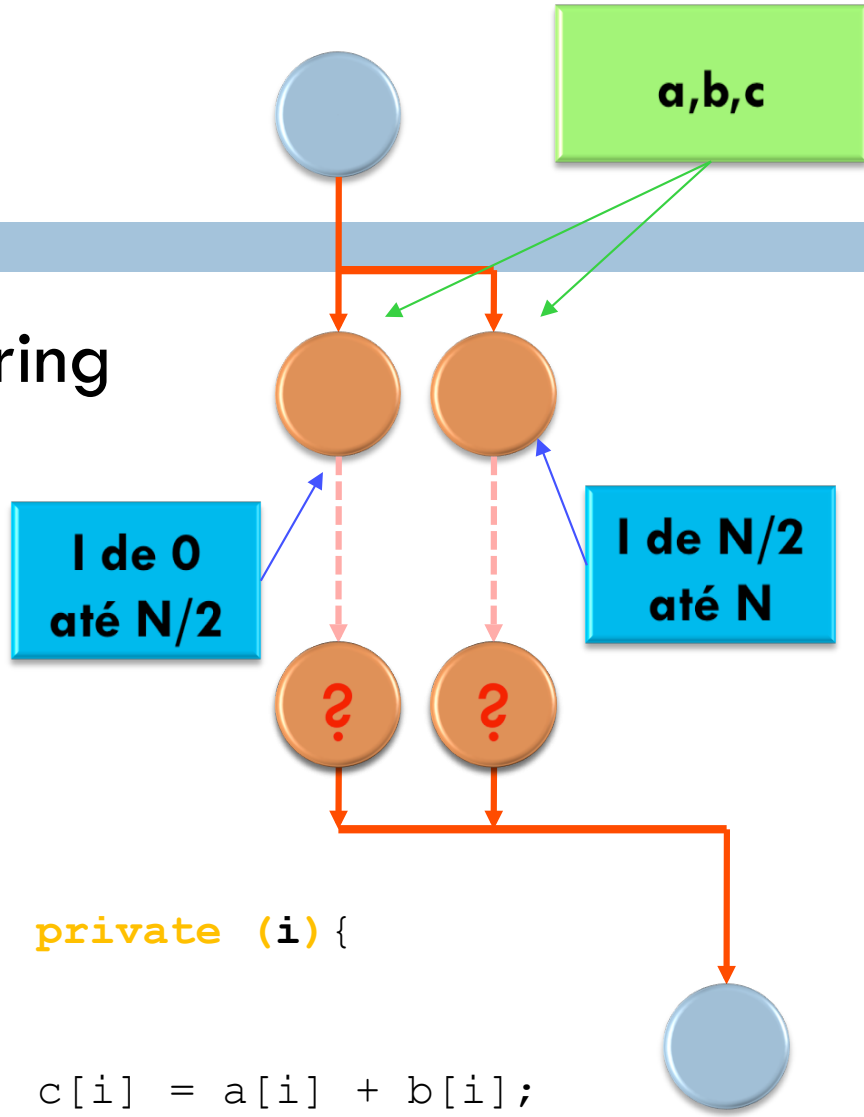
```
{
   #pragma omp section newline
        código_blc_01;
  #pragma omp section newline
        código_blc_02;
};
```

OpenMP

- Construtores de Work-Sharing
  - Clausulas para o "**sections**"
    - private(*list*)
    - firstprivate(*list*)
    - lastprivate(*list*)
    - reduction(*operator: list*)
    - nowait

OpenMP

- Construtores de Work-Sharing
  - Exemplo **sections**:




```
#include <omp.h>
#define N 1000
int main(int argc, char **argv){
     int i, chunk;
     float a[N], b[N], c[N];
     for (i=0; i < N; i++)
          a[i] = b[i] = i * 1.0;
     #pragma omp parallel shared (a,b,c) private (i){
          #pragma omp sections nowait{
               #pragma omp section
                  for ((i=0; i < N/2; i++) c[i] = a[i] + b[i];
               #pragma omp section
                  for ((i=N/2; i < N; i++) c[i] = a[i] + b[i];
          } //Sinc. dos threads ? Observar clausula nowait
     }
}
```

OpenMP

- Construtores de Sincronização
  - Funcionam como sinalizadores de tempo
    - São fundamentais para a sincronização dos threads
      - **master**
        - Especifica uma região que será executada apenas pelo master
      - **critical**
        - Especifica uma região crítica de código que deve ser executada apenas por um thread de cada vez.
      - **barrier**
        - Quando esta diretiva é alcançada por um thread, este espera até que os demais alcancem o mesmo ponto.

OpenMP

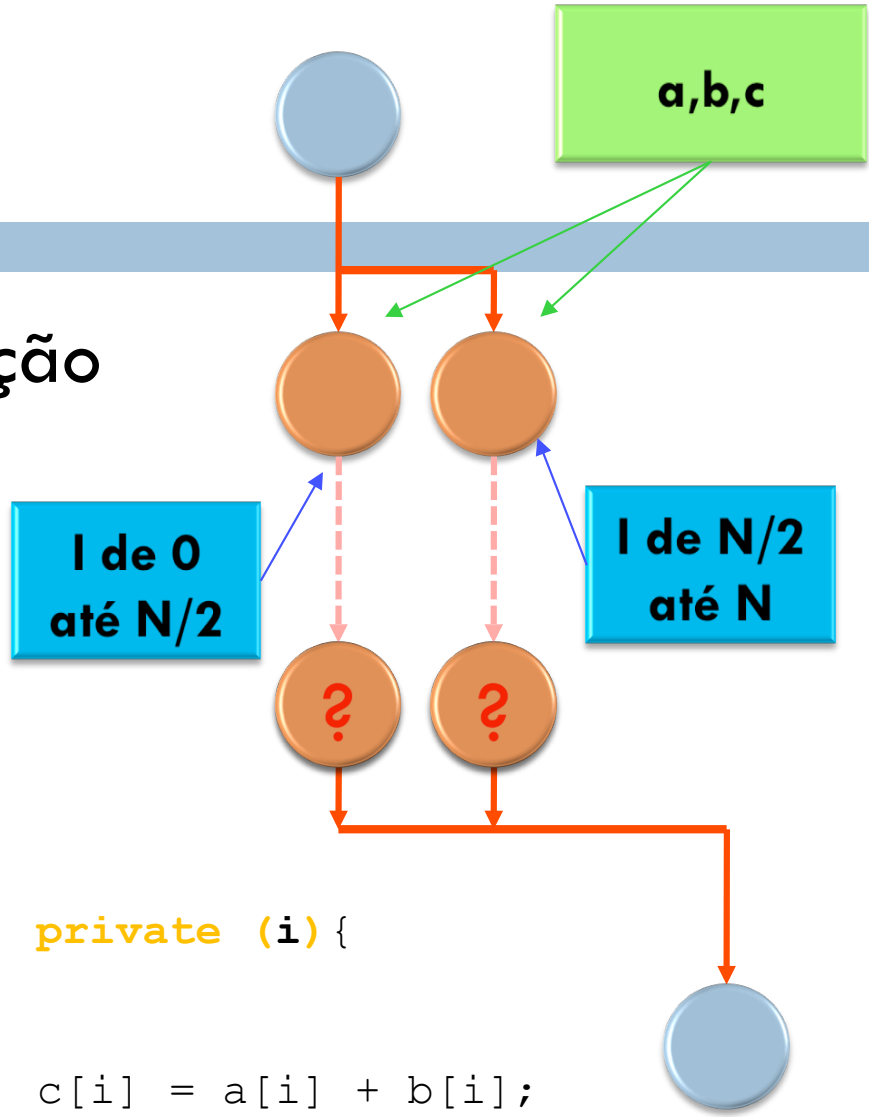
- Construtores de Sincronização
  - Funcionam como sinalizadores de tempo
    - São fundamentais para a sincronização dos threads
      - **atomic**
        - Define um endereço de memória para atualização atômica.
        - Código é executado sem separação! Ex....
      - **flush**
        - Identifica um ponto de sincronização. Necessário para garantir visão consistente da memória.
        - Assegura que as variáveis ficam atualizadas para todos os threads. Se não há lista, são atualidas todas as variáveis compartilhadas.

OpenMP

- Construtores de Sincronização
  - Funcionam como sinalizadores de tempo
    - São fundamentais para a sincronização dos threads
      - **ordered**
        - Obriga as iterações serem executadas na mesma ordem como se a execução fosse sequencial

OpenMP

- Construtores de Sincronização
  - Exemplo **sections**:




```
#include <omp.h>
#define N 1000
int main(int argc, char **argv){
     int i, chunk;
     float a[N], b[N], c[N];
     for (i=0; i < N; i++)
          a[i] = b[i] = i * 1.0;
     #pragma omp parallel shared (a,b,c) private (i){
          #pragma omp sections nowait{
               #pragma omp section
                  for ((i=0; i < N/2; i++) c[i] = a[i] + b[i];
               #pragma omp section
                  for ((i=N/2; i < N; i++) c[i] = a[i] + b[i];
          } //Sinc. dos threads ? Observar clausula nowait
     }
}
```

OpenMP

- Funções OpemMP
  - void omp_set_num_threads (int)
    - Chamada na parte sequencial
    - Define o número de Threads que vão ser utilizados
  - int omp_get_num_threads (void)
    - Retorna o número de threads ativos no sistema
  - int omp_get_max_threads (void)
    - Retorna o número máximo de threads “disponíveis”

OpenMP

- Funções OpemMP
  - int omp_get_thread_num (void)
    - Retorna o ID do thread (entre 0 e N-1)
  - int omp_get_num_procs (void)
    - Retorna o número de processadores existentes
  - void omp_init_lock (omp_lock_t*)
    - Inicializa um bloqueio à variável parâmetro
  - void omp_destroy_lock (omp_lock_t*)
    - Limpa a o bloqueio à variável parâmetro

OpenMP

- Funções OpemMP
  - void omp_set_lock (omp_lock_t*)
    - Aguarda condição favorável para conseguir o lock
  - void omp_unset_lock (omp_lock_t*)
    - Libera o bloqueio à variável parâmetro
  - double omp_get_wtime (void)
    - Retorna número de segundos "*elapsed time*"
  - double omp_get_wtick (void)
    - Retorna os segundos decorridos entre chamadas sucessivas

Construtor sections

- A cada secção executa-se por um thread.
- Há barreira ao final se não se utiliza a cláusula **nowait.**
- Cláusulas de partição de variables: **private, firstprivate, lastprivate** e **reduction**

```
#pragma omp sections [cláusulas]
{
   [#pragma omp section]
      bloque
   [#pragma omp section
      bloque
      ...
   ]
}
```

Variáveis de Ambiente

- **OMP_SCHEDULE:** indica o tipo de scheduling para ***for*** e ***parallel for.***
- **OMP_NUM_THREADS:** põe o número de threads a ser utilizada no escopo do código.
- **OMP_DYNAMIC:** habilita ou desabilita o ajuste dinâmico do número de threads.