① 6ª Aula

Constrotoles de divisão de trabalho

São responsáveis pela distribuição de trabalho entre os threads e determinam como o trabalho será dividido.

#pragma omp for

#pragma omp sections

#pragma omp single

② Constrotor for

- É responsável pela divisão das iterações do laço executadas em paralelo
- Número de iterações deve ser conhecida (diferente do "while")
- Implementa SIMD

③

```
#include <omp.h>
int main(){
    #pragma omp parallel private(i)
    {
      #pragma omp for
          for (i=0; i<n; i++)
              a[i] = b[i] + c[1];
    }
    return 0;
}
```

④ Cláusula schedule

→ Utilizada apenas no construtor for e controla a forma como as iterações são distribuídas entre as threads.

→ #pragma omp for schedule (~~name_schedule~~ Kind, chunK)

→ Tipos de schedule (~~name_schedule~~ Kind):

- → static: iterações divididas em conjuntos de tamanho "chunK".
- → dynamic: iterações divididas em [redacted] conjuntos de "chunK" iterações contínuas.
- → guided: similar ao dynamic, mas o tamanho dos pedaços decresce de forma exponencial.
- → runtime: esquema de divisão das iterações do laço escolhido em tempo de execução.

⑤ Exemplos

```
#pragma omp parallel num_threads(3)
{
  #pragma omp for schedule (static,2)
  for (i=0; i<6; i++){
      id = omp_get_thread_num();
      printf("Thread %d executa a iteração %d \n", id, i);
  }
}
```

⑥ Exemplos (continuação)

```
#pragma omp parallel num_threads(3)
{
  #pragma omp for schedule (dynamic,1)
  for (i=0; i<6; i++){
      id = omp_get_thread_num();
      printf("Thread %d executa a iteração %d\n", id, i);
  }
}
```

⑦ Exemplos (continuação)

```
#pragma omp parallel num_threads(3)
{
  #pragma omp for schedule(guided)
  for (i = 0; i < 7; i++){
      id = omp_get_thread_num();
      printf("Thread %d executa a iteração %d \n", id, i);
  }
}
```

⑧ Constutor sections

- Faz a divisão das tarefas entre os threads quando não existem iterações (laços)
- Implementa paralelismo de controle (MIMD)
- Cada seção é executada por apenas uma thread

(9) Exemplo:

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    for (i=0; i<n, i++)
      c[i] = a[i] + b[i];
    #pragma omp section
    for (i=0; i<n; i++)
      d[i] = a[i] - b[i];
  }
}
```

(10) Cláusula reduction

- Uma cópia de cada variável é criada para cada thread
- reduction (operator : variables)
- Ao final da região paralela definida pelo constutor, a lista de variáveis original é atualizada com os valores da cópia privada de cada thread usando o operador especificado

(11) Exemplo:

* Sequencial:

```
soma = 0;
for(i=0; i<N; i++)
    soma += v[i]
```

* Paralelo:

```
soma = 0
#pragma omp parallel for
for(i=0; i<N; i++)
    soma += v[i];
```

Funciona?

```
#pragma omp parallel for
reduction (+: soma)
for (i=0; i<N; i++)
    soma += [i];
```

E agora?

(12) Construtor single

→ Indica que o bloco sintático localizado logo abaixo do mesmo deve ser executado por apenas uma thread

→ Não necessariamente a thread mestre

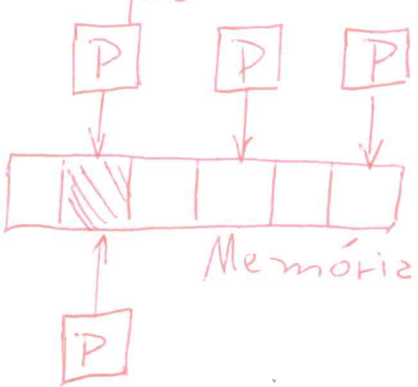
→ #pragma omp single

(13) Constructos de sincronização

- Diretivas de sincronização garantem que o acesso ou atualização de uma variável compartilhada ocorra no momento certo
- Permitem:
  - Estabelecer pontos de sincronização em regiões paralelas
  - Definir regiões críticas
  - Evitar condições de corrida em variáveis compartilhadas

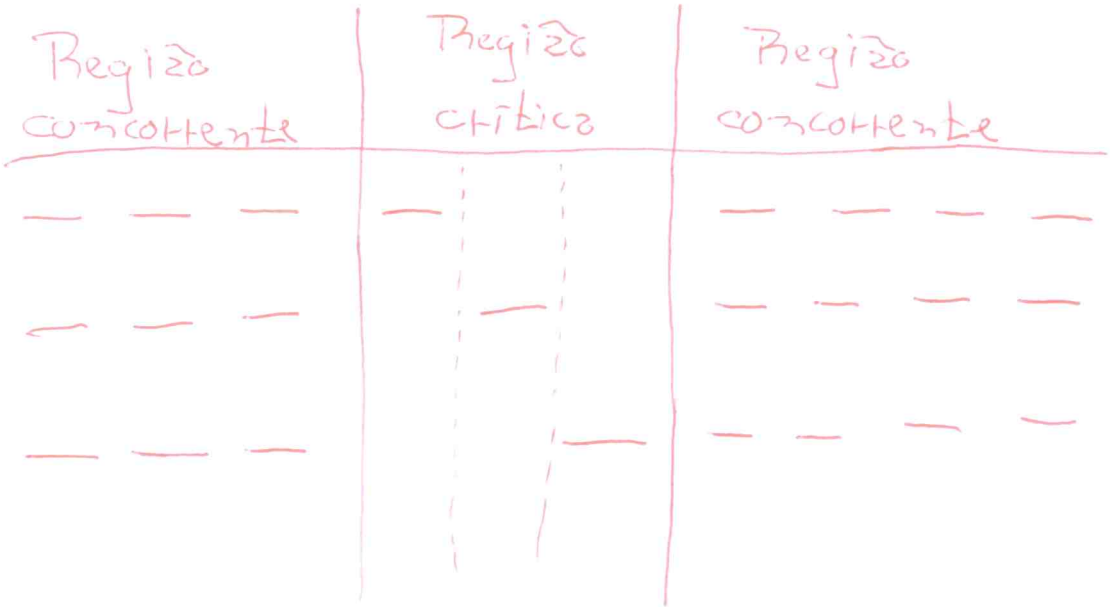
(14) Condições de corrida (race conditions)

- Acontece quando duas ou mais threads tentam atualizar, ao mesmo tempo, uma mesma variável ou quando uma thread atualiza uma variável e outra thread acessa o valor dessa variável ao mesmo tempo



(15) Diretiva #pragma omp critical

→ Garante que o acesso será feito por uma thread por vez

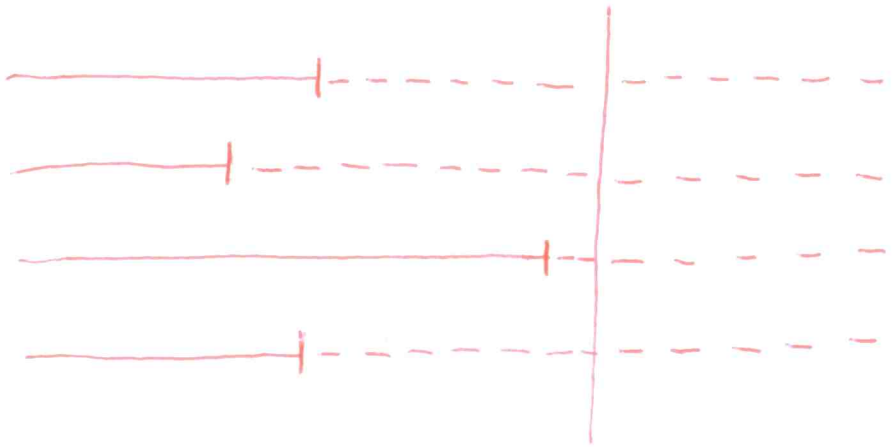


(16) Exemplo

```
float soma = 0;
#pragma omp parallel
}
  float soma_par = 0;
  #pragma omp for
  for (i = 0; i < N; i++)
      soma_par += v[i];
  #pragma omp critical
  soma += soma_par;
}
```

(17) Diretiva #pragma omp barrier

→ Define um ponto de sincronização explícito entre todas as threads de uma região paralela.



Barreira

(18) Diretiva #pragma omp atomic

→ Define que a próxima instrução de código deve ser executada de forma atômica (uma thread de cada vez)

→ A instrução deve ser uma atribuição

→ Impede que várias threads acessem uma variável ao mesmo tempo

(19) Exemplo

```
float soma = 0;
#pragma omp parallel
{
  float soma_par = 0;
  #pragma omp for
  for (i = 0; i < N; i++)
      soma_par += v[i];
  #pragma omp atomic
  soma += soma_par;
}
```

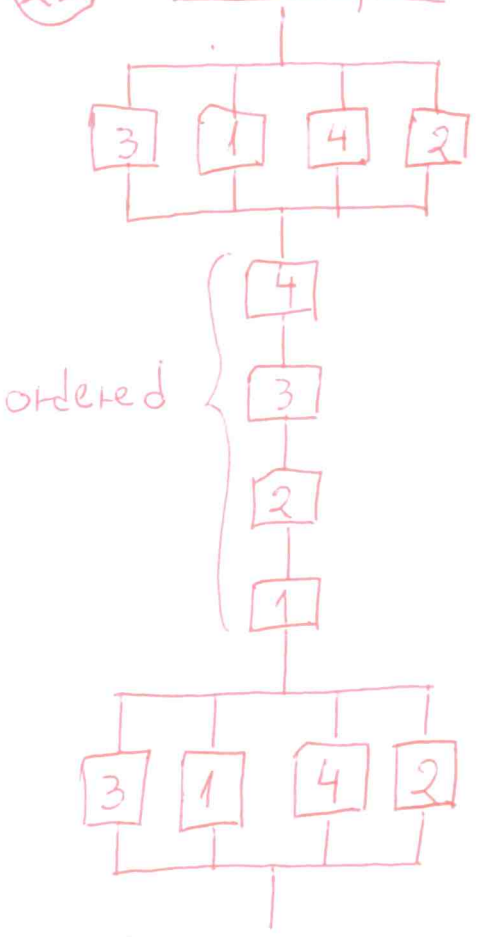
(20) Diretiva #pragma omp flush

- Sincroniza a memória global com a memória de uma thread específica
- #pragma omp flush(list)
- Define um ponto no qual os valores das variáveis em list são atualizadas na memória, produzindo uma visão consistente da memória
- Se o parâmetro list é omitido são atualizadas todas as variáveis compartilhadas.

(21) Diretiva #pragma omp ordered

- Define um bloco de código em que as iterações de um laço "for" devem ser executadas na mesma ordem do programa sequencial
- Deve ser utilizado quando houver dependência entre as iterações de um laço.

(22) Exemplo




```
omp_set_num_threads(4);
#pragma omp parallell for ordered
  for(i=0; i<N; i++){
    a[i] = complex_function_1(i);

    #pragma omp ordered
    printf("a[%d] = %d \n", i, a[i]);

    b[i] = complex_function_2(a[i]);
  }
```