

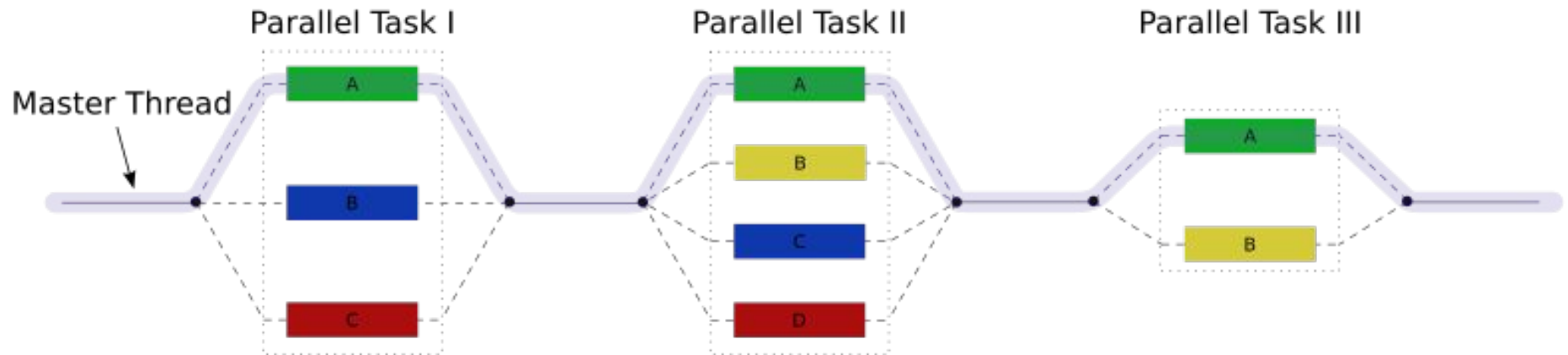
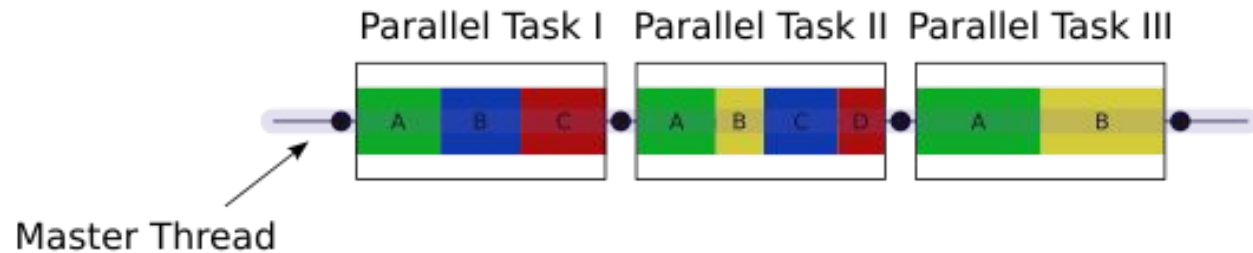
SuperComputação

Aula 10 – For paralelo, reduções e escopo

2019 – Engenharia

Igor Montagner, Luciano Soares [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

Aulas passadas



Aulas passadas

Modelo fork-join:

- Usando `std::thread`
- Usando OpenMP `#pragma parallel`

Aulas passadas

Modelo fork-join:

```
#pragma omp parallel
{
    int id = omp_get_thread_num();
    long nt = omp_get_num_threads();
    double d = 0;
    for (long i = id; i < num_steps; i+=nt) {
        double x = (i-0.5)*step;
        d += 4.0/(1.0+x*x);
    }
    #pragma omp atomic
    sum += d;
}
```

Hoje

- Construção do for paralelo
- Compartilhamento de dados
- Dúvidas da aula passada

Single Program Multiple Data

OpenMP foi inicialmente criado para minimizar as modificações a um programa sequencial.

Construções de divisão de trabalho

- For paralelo
- Seções
- `single/master`

Construções de tarefas

For paralelo

Cria threads e distribui entre elas as iterações de um loop.

```
#pragma omp  
parallel  
{  
    #pragma omp for  
    for (int i=0;i<n;i++) {  
        trabalhe(i);  
    }  
}
```

A variável "i" é feita privada para cada thread por padrão.

For paralelo

- Código Sequencial
- Loop com omp parallel de forma manual
- Loop com omp parallel for

```
for(i=0;i<N;i++) { a[i] = a[i] + b[i];}
```

```
#pragma omp parallel
```

```
{
```

```
    int id, i, Nthrds, istart, iend;
```

```
    id = omp_get_thread_num();
```

```
    Nthrds = omp_get_num_threads();
```

```
    istart = id * N / Nthrds;
```

```
    iend = (id+1) * N / Nthrds;
```

```
    if (id == Nthrds-1)iend = N;
```

```
    for(i=istart;i<iend;i++) { a[i] = a[i] + b[i];}
```

```
}
```

```
#pragma omp parallel
```

```
#pragma omp for
```

```
    for(i=0;i<N;i++) { a[i] = a[i] + b[i];}
```

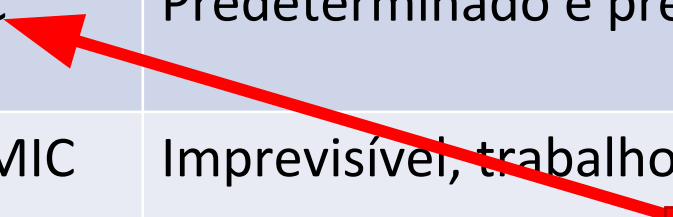

Escalonamento

| Tipo | Quando usar |
|---------|--|
| STATIC | Predeterminado e previsível pelo programador |
| DYNAMIC | Imprevisível, trabalho varia muito por iteração |
| GUIDED | Caso especial de <i>dynamic</i> para reduzir a sobrecarga do escalonamento |
| AUTO | Quando o a biblioteca de runtime pode "Aprender" de execuções anteriores do mesmo loop |

Uso: `#pragma omp parallel for schedule(tipo, chunk)`

Escalonamento

| Tipo | Quando usar |
|---------|--|
| STATIC | Predeterminado e previsível pelo programador |
| DYNAMIC | Imprevisível, trabalho varia muito por iteração |
| GUIDED | Caso especial de <i>dynamic</i> do escalonamento |
| AUTO | Quando o a biblioteca usa resultados de execuções anteriores do mesmo loop |



Menos trabalho em tempo de execução, escalonamento feito em tempo de compilação

Uso: `#pragma omp parallel for schedule(tipo, chunk)`

Escalonamento

| Tipo | Quando usar |
|---------|--|
| STATIC | Predeterminado e previsível pelo programador |
| DYNAMIC | Imprevisível, trabalho varia muito por iteração |
| GUIDED | Caso especial de <i>dynamic</i> para reduzir a sobrecarga do escalonamento |
| AUTO | O compilador decide o melhor time pode "Aprender" com o mesmo loop |

Mais trabalho em tempo de execução, lógica de escalonamento mais complexa, consumindo tempo de execução

Uso: `#pragma omp taskwait schedule(tipo, chunk)`

Operações de redução

- Como lidar com esse caso?

```
double ave, A[MAX];  
for (int i=0;i<MAX; i++) {  
    ave += A[i];  
}  
ave = ave/MAX;
```

- Acumulamos os resultados das iterações em `ave`
- Iterações dependentes = não podemos paralelizar
- Esta operação é chamada "redução".

Operações de redução

- Construção `reduction (op:var)`

```
double ave, A[MAX];  
#pragma omp parallel for reduction (+:ave)  
for (int i=0; i<MAX; i++) {  
    ave += A[i];  
}  
ave = ave/MAX;
```

- Cada thread utiliza uma cópia local
- No fim as cópias são acumuladas em `var`

Aulas passadas

Modelo fork-join:

```
#pragma omp parallel
{
    int id = omp_get_thread_num();
    long nt = omp_get_num_threads();
    double d = 0;
    for (long i = id; i < num_steps; i+=nt) {
        double x = (i-0.5)*step;
        d += 4.0/(1.0+x*x);
    }
    #pragma omp atomic
    sum += d;
}
```

Operações de redução

| Operador | Valor Inicial |
|----------|---------------|
| + | 0 |
| - | 0 |
| * | 1 |
| MIN | $+\infty$ |
| MAX | $-\infty$ |

| Operador | Valor Inicial |
|----------|---------------|
| & | ~ 0 |
| | 0 |
| ^ | 0 |
| && | 1 |
| | 0 |

Compartilhamento de dados

Tudo o que já existe é compartilhado:

- Variáveis globais e alocadas dinamicamente (new, malloc)
- Variáveis apontadas por ponteiros
- Variáveis locais criadas fora das regiões paralelas

Declarações de variáveis locais dentro das threads são privadas.

Compartilhamento de dados

main.c

```
double A[10];
int main(){
    int index[10];
    #pragma omp parallel
        work(index);
    printf("%d\n",index[0]);
}
```

```
extern double A[10];
void work(int *index) {
    double temp[10];
    static int count;
    ...
}
```

- **A**, **index** e **count** são compartilhados por todas as threads. **Temp** é local para cada thread

Compartilhamento de dados

Podemos especificar a forma de compartilhamento:

- `shared(lista de variáveis)`
- `private(lista de variáveis)`
- `firstprivate(lista de variáveis)`
- `lastprivate (lista de variáveis)`
- `default (none)`

Compartilhamento de dados

Podemos especificar a forma de compartilhamento:

- `private(lista de variáveis)`
 - Não inicializadas
- `firstprivate(lista de variáveis)`
 - Inicializadas com o valor existente
- `lastprivate (lista de variáveis)`
 - Assumem valor da última iteração ao terminar

Exemplo: private

Nomenclatura: a versão do tmp anterior à construção é chamada de variável "original"

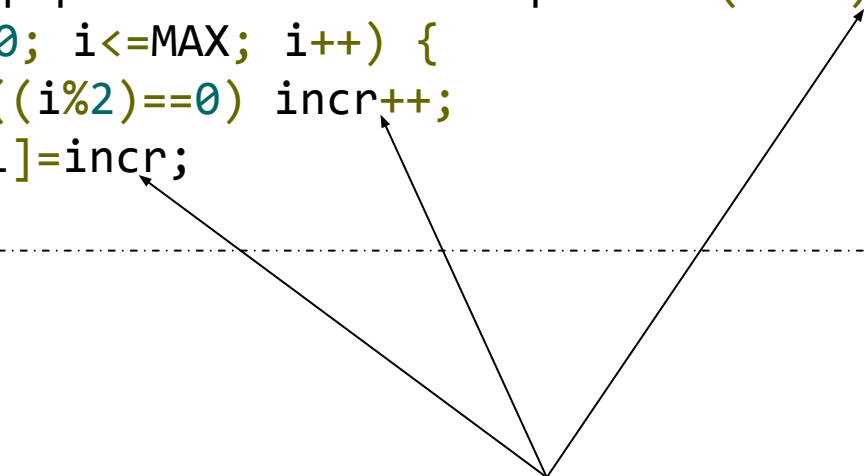
```
void wrong() {  
    int tmp=0;  
    #pragma omp parallel for private(tmp)  
        for(int j=0;j<1000;++j)  
            tmp+=j;  
    printf("%d\n",tmp);  
}
```

tmp não foi inicializado

tmp fica com o valor da variável original após a região construída (0 neste caso)

Exemplo: firstprivate

```
incr=0;
#pragma omp parallel for firstprivate(incr)
    for(i=0; i<=MAX; i++) {
        if((i%2)==0) incr++;
        A[i]=incr;
    }
```



Cada thread obtém sua própria cópia de incr, com um valor inicial de 0

Exemplo: lastprivate

```
void sq2(int n, double *lastterm) {  
    double x; int i;  
    #pragma omp parallel for lastprivate(x)  
    for(i=0;i<n;i++) {  
        x=a[i]*a[i]+b[i]*b[i];  
        b[i]=sqrt(x);  
    }  
    *lastterm=x;  
}
```

"X" tem o valor que ocupou para a iteração
"última sequência" (isto é, para $i = (n-1)$)

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável A é:

- a) Compartilhada entre todas threads e começa com 1
- b) Compartilhada entre todas as threads, mas não inicializada
- c) Privada para cada thread e começa com 1

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável A é:

- a) **Compartilhada entre todas threads e começa com 1**
- b) Compartilhada entre todas as threads, mas não inicializada
- c) Privada para cada thread e começa com 1

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável B é:

- a) Compartilhada entre todas threads e começa com 1
- b) Privada para cada thread, mas não inicializada
- c) Privada para cada thread e começa com 1

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável B é:

- a) Compartilhada entre todas threads e começa com 1
- b) Privada para cada thread, mas não inicializada**
- c) Privada para cada thread e começa com 1

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável C é:

- a) Compartilhada entre todas threads e começa com 1
- b) Privada para cada thread, mas não inicializada
- c) Privada para cada thread e começa com 1

Exemplo: "QUIZ"

No exemplo abaixo:

```
variáveis: A=1,B=1,C=1  
#pragma omp parallel private(B) firstprivate(C)
```

A variável C é:

- a) Compartilhada entre todas threads e começa com 1
- b) Privada para cada thread, mas não inicializada
- c) Privada para cada thread e começa com 1**

Atividade prática

- Final da atividade para entrega
- Exercício sobre compartilhamento de dados

Referências

- Livros:
 - Hager, G. ; Wellein, G. **Introduction to High Performance Computing for Scientists and Engineers**. 1ª Ed. CRC Press, 2010.
- Artigos:
 - Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." *IEEE computational science and engineering* 5, no. 1 (1998): 46-55.
- Internet:
 - <https://www.youtube.com/playlist?list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG>
 - <http://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf>
 - http://extremecomputingtraining.anl.gov/files/2016/08/Mattson_830auq3_HandsOnIntro.pdf

Inspire

www.inspire.edub.org