

SuperComputação

Aula 13 – Paralelismo de dados

2021 – Engenharia

Igor Montagner <igorsm1@insper.edu.br>
Antônio Selvatici <antoniohps1@insper.edu.br>

Solução de alto desempenho

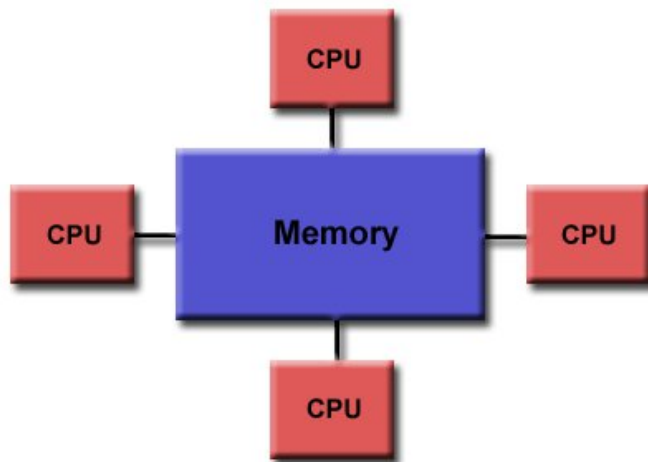
1. Algoritmos eficientes

2. Implementação eficiente

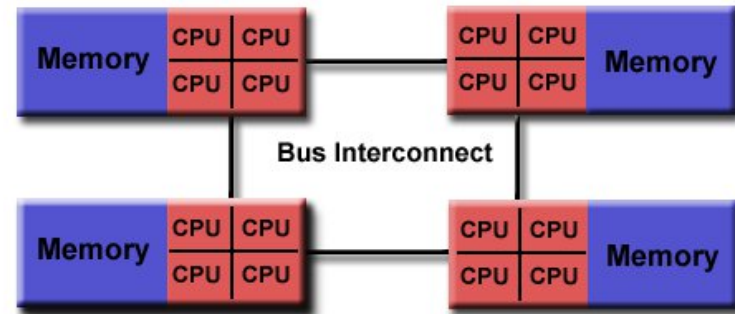
- Cache, paralelismo de instrução
- Linguagem de programação adequada

3. Paralelismo

Sistemas Multi-core



Uniform Memory Access



Non-Uniform Memory Access

Conceito 1: Dependência

Um loop tem uma **dependência** de dados sua execução correta depende da ordem de sua execução.

Isto ocorre quando **uma iteração depende de resultados calculados em iterações** anteriores.

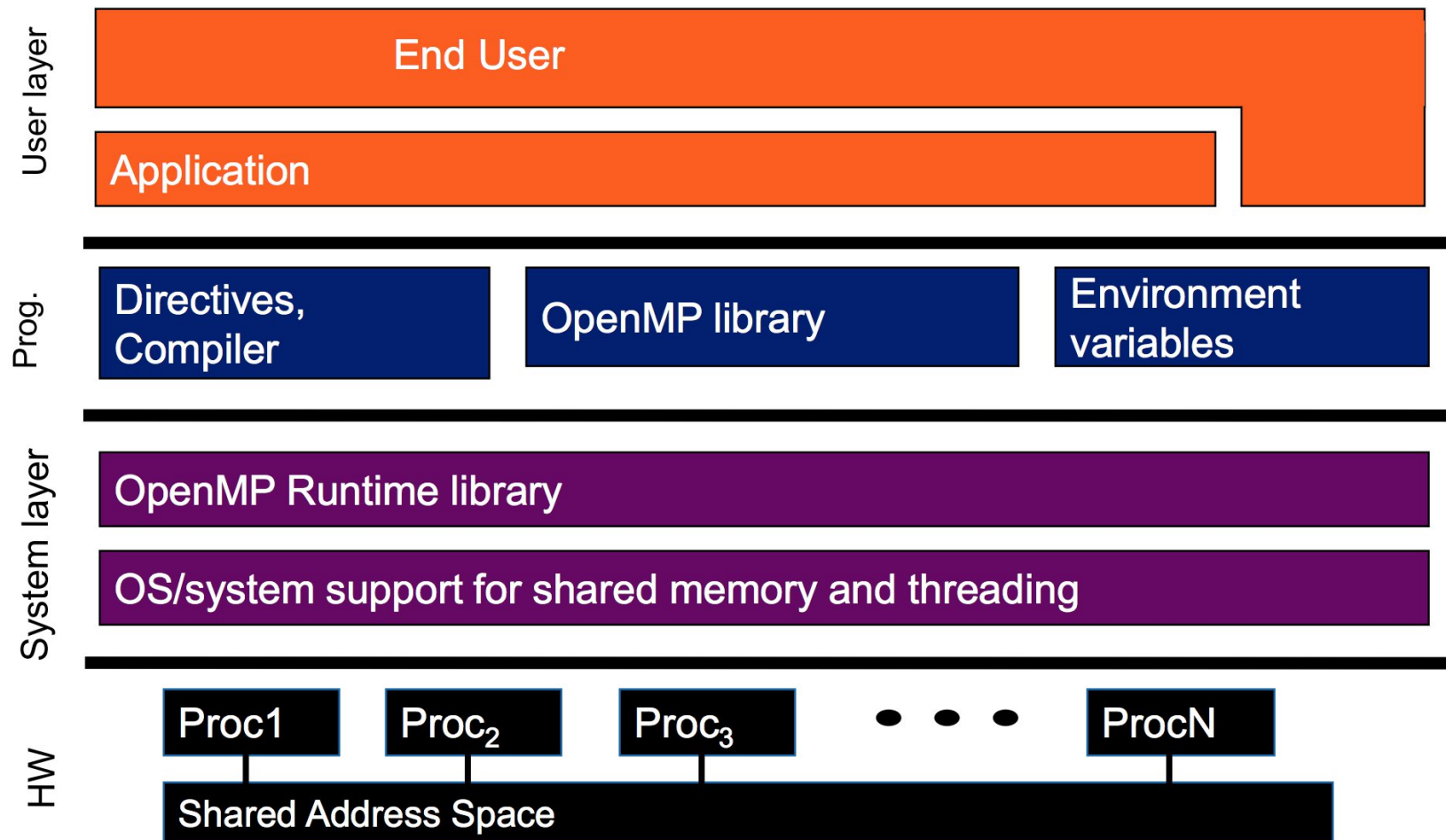
Quando não existe nenhuma dependência em um loop ele é dito **ingenuamente paralelizável**.

Conceito 2: Paralelismo

Paralelismo de dados: faço em paralelo a mesma operação (lenta) para todos os elementos em um conjunto de dados (grande).

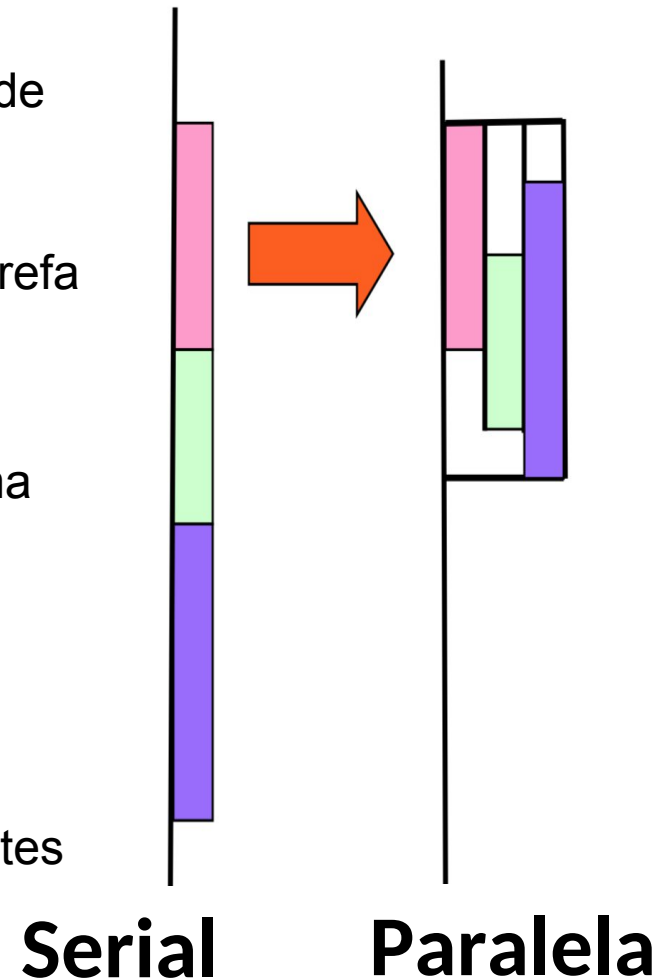
Paralelismo de tarefas: faço em paralelo duas (ou mais) tarefas independentes. Se houver dependências quebro em partes independentes e rodo em ordem.

OpenMP (host / NUMA)



O quê são tarefas?

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



Tarefas em OpenMP

#pragma omp task[clauses]

```
#pragma omp parallel  
{
```

```
  #pragma omp master  
  {
```

```
    #pragma omp task  
    func1();
```

```
    #pragma omp task  
    func2();
```

```
    #pragma omp task  
    func3();
```

```
  }
```

```
}
```

Crie um conjunto de threads

Thread 0 organiza as tarefas

Tarefas executadas por alguma thread em alguma ordem

Todas as tarefas devem ser concluídas antes que esta barreira seja liberada



Atividade prática

Paralelismo de tarefas com OpenMP (20 minutos)

1. Resolver um problema prático usando tarefas.

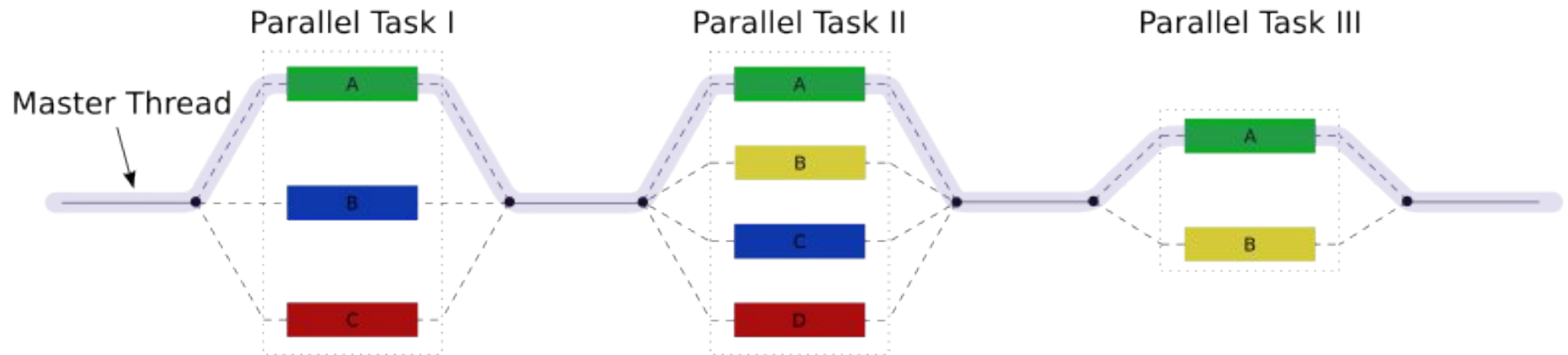
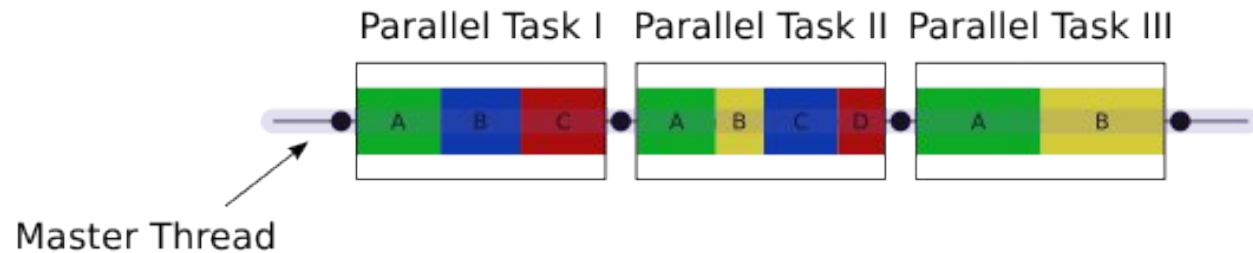
Paralelismo de dados

Conceito 2: Paralelismo

Paralelismo de dados: faço em paralelo a mesma operação (lenta) para todos os elementos em um conjunto de dados (grande).

Paralelismo de tarefas: faço em paralelo duas (ou mais) tarefas independentes. Se houver dependências quebro em partes independentes e rodo em ordem.

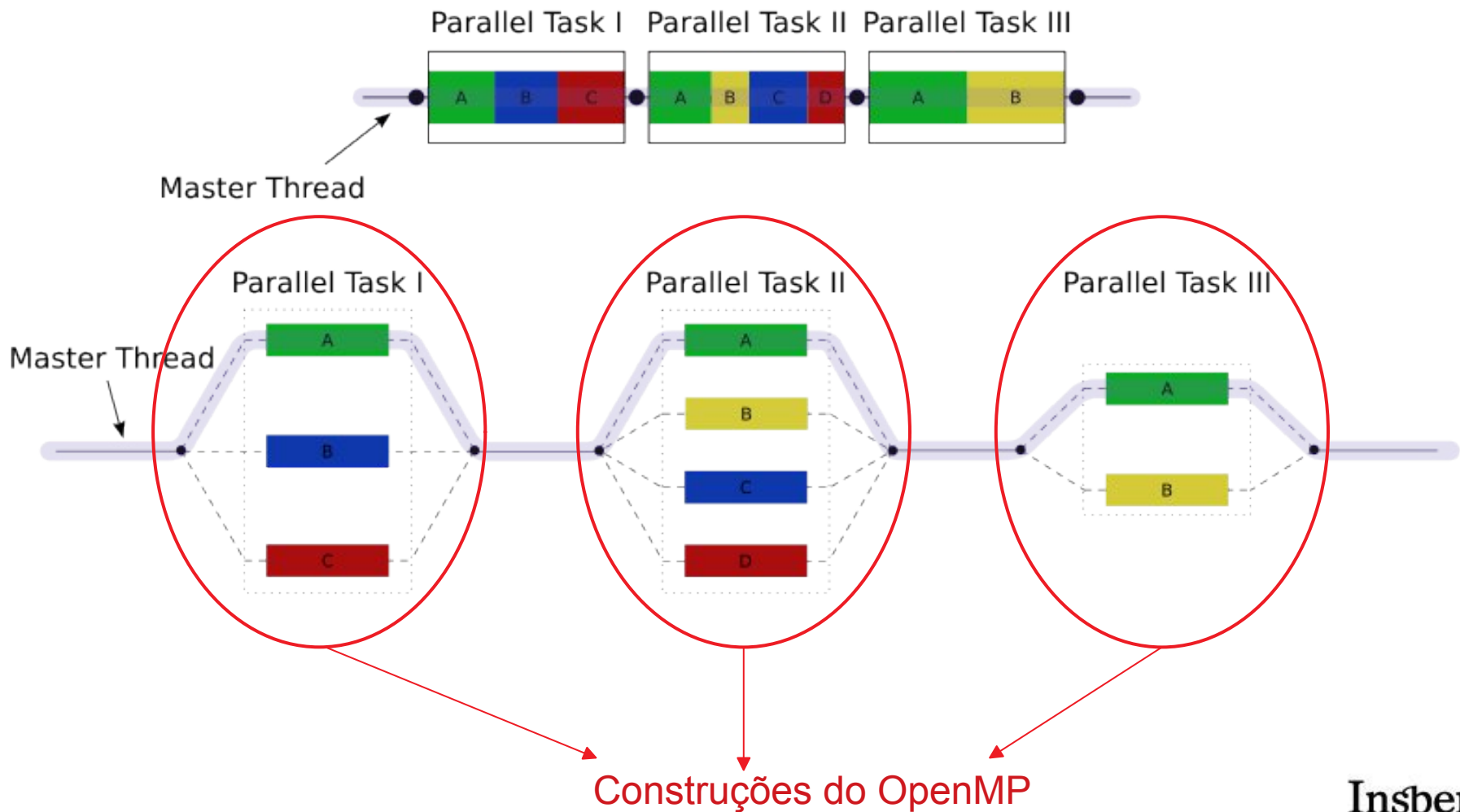
Modelo fork-join



Modelo fork-join e paralelismo de dados

- Todas as threads rodam a mesma função
- Espero todas acabarem para recolher os resultados
- Digo explicitamente quais variáveis são usadas em cada thread e se elas são locais da thread ou se são compartilhadas

OpenMP – aplicação do modelo fork-join



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 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];}
```

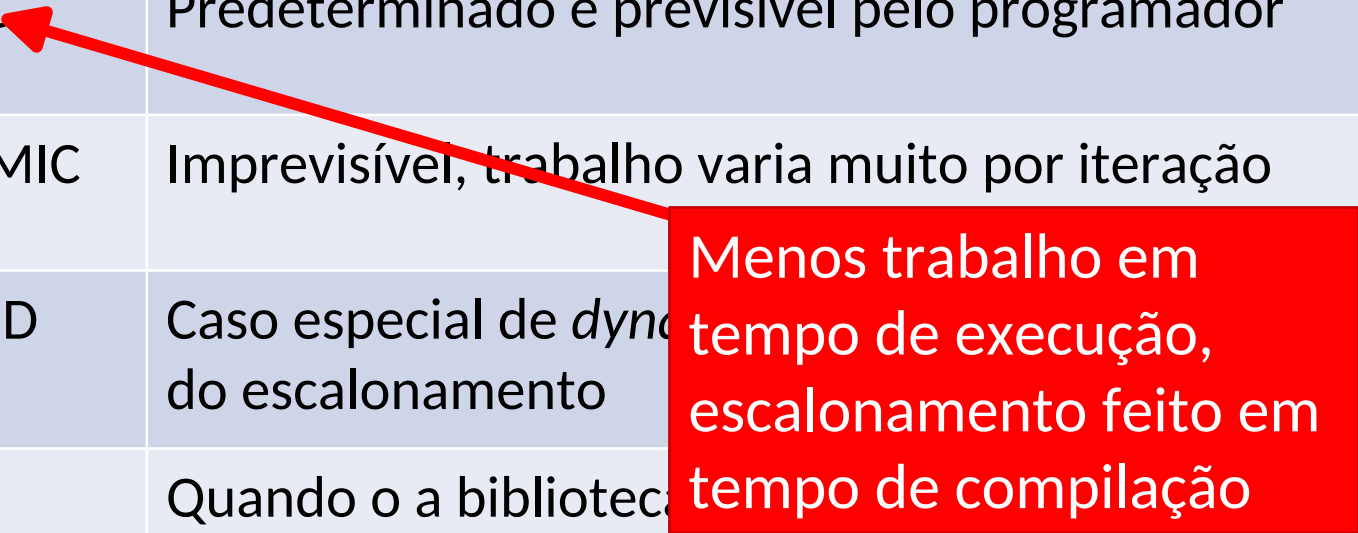
Escalonamento

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	Caso especial de <i>dynamic</i> onde o compilador pode "Aprender" o melhor modo de dividir o trabalho no mesmo loop

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

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



Atividade prática

Paralelismo de dados com OpenMP (30 minutos)

1. Utilização de parallel for para resolver problemas simples

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`

Operações de redução

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

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



Atividade prática

Exercício prático de for paralelo (20 minutos)

1. Identificação de dependências de dados
2. Tentativas para evitar dependências

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

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

main.c

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

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

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: "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**

Insper

www.insper.edu.br