Insper

SuperComputação

Aula 11 – Granularidade e Produtor/Consumidor

2018 – Engenharia

Igor Montagner, Luciano Soares <igorsm1@insper.edu.br>

Aulas passadas

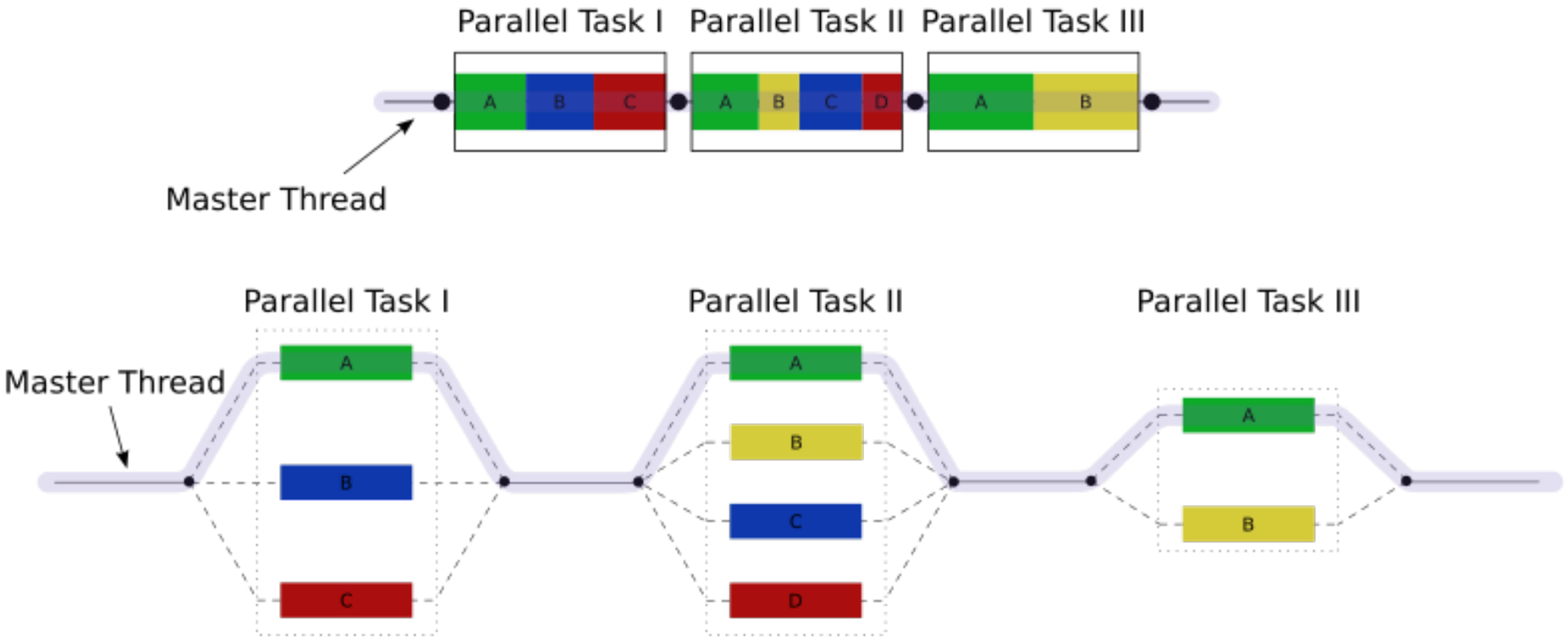


Figura: https://en.wikipedia.org/wiki/File:Fork_join.svg

Aulas passadas

1) Modelo fork-join

2) Tarefas facilmente paralelizáveis

3) Tarefas inerentemente sequenciais

    - exemplo com números aleatórios

Hoje

1) Granularidade

2) Modelo produtor consumidor

Granularidade

- Relação entre o tamanho da tarefa e o custo de lançar/trocar de tarefas
- Alta granularidade:
  - Tarefas pouco complexas
  - Facilmente paralelizáveis
  - Fáceis de escalonar

Granularidade

- Relação entre o tamanho da tarefa e o custo de lançar/trocar de tarefas
- Baixa granularidade:
  - Tarefas complexas
  - Mais difíceis de escalonar
  - Úteis se o custo de lançar/trocar tarefas for alto

OpenMP - Escalonamento

Tipo	Quando usar
STATIC	Predeterminado e previsível pelo programador
DYNAMIC	Imprevisível, trabalho varia muito por iteração
GUIDED	Caso especial de *dynamic* 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)`

OpenMP - Escalonamento

Tipo	Quando usar
STATIC	Predeterminado e previsível pelo programador
DYNAMIC	Imprevisível, trabalho varia muito por iteração
GUIDED	Caso especial de sobrecarga do es
AUTO	Quando o a biblio "Aprender" 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)`

OpenMP - Escalonamento

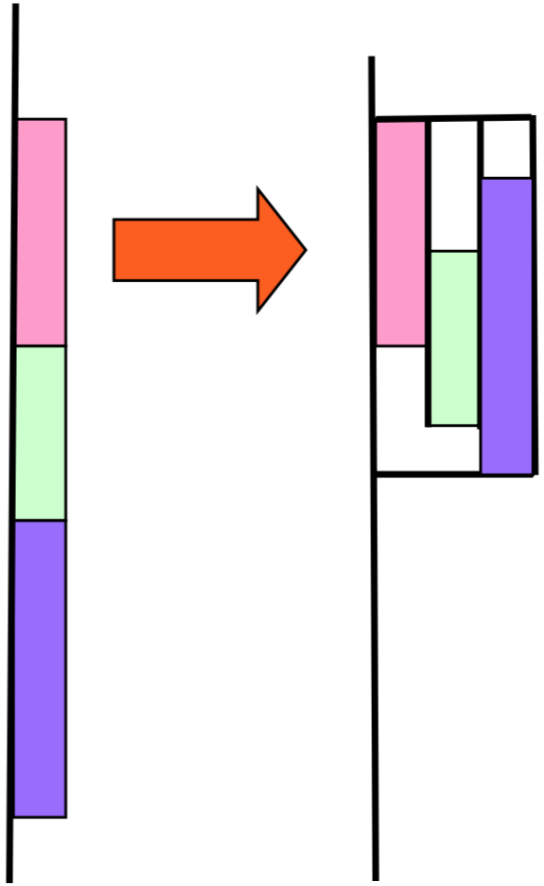
Tipo	Quando usar
STATIC	Predeterminado e previsível pelo programador
DYNAMIC	Imprevisível, trabalho varia muito por iteração
GUIDED	Caso especial de *dynamic* para reduzir a sobrecarga do escalonamento
AUTO	Quando a biblioteca de runtime pode "[illegible]es anteriores do r[illegible]

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

Uso: `#pragma [illegible]edule(tipo, chunk)`

Escalonamento

- Atribuir tarefa a processador
- Várias estratégias possíveis
- Troca entre tarefas tem custo
- Influencia no tempo final



Serial **Paralela**

Tarefas heterogêneas

- Algumas dependem de recursos compartilhados
- Algumas são ingenuamente paralelizáveis
- Algumas são inerentemente sequenciais, mas são independentes entre si.

Exclusão mútua

- Acessamos um recurso compartilhado
- Região crítica: porção do código que manipula o recurso
- Propriedade: somente um thread por vez na região crítica

Exclusão mútua - OpenMP

```
#pragma omp critical [name]
```

- Cria uma região crítica
- Se *name* for passado, somente uma thread pode estar ativa nas regiões com o mesmo nome.

Modelo produtor-consumidor

Dois conjuntos de threads

- Produzem tarefas a serem executadas
    - pode depender de um recurso compartilhado
    - controlar tamanho das tarefas
- Consomem as tarefas e as executam
    - tarefas independentes entre si
    - tarefas independentes da produção

Modelo produtor-consumidor

- Depende de uma fila compartilhada
- Consumidor retira tarefas da fila
- Produtor adiciona tarefas à fila

Próximas aulas

- Situações em que este modelo é vantajoso
- Implementação do modelo produtor-consumidor
- Primitivas de sincronização (em C++)

Referências

- Livros:
  - Hager, G. ; Wellein, G. **Introduction to High Performance Computing for Scientists and Engineers**. 1ª Ed. CRC Press, 2010.
- Artigos:
  - Duran, Alejandro, Julita Corbalán, and Eduard Ayguadé. "Evaluation of OpenMP task scheduling strategies." In *International Workshop on OpenMP*, pp. 100-110. Springer, Berlin, Heidelberg, 2008
- 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_830aug3_HandsOnIntro.pdf

Insper

www.insper.edu.br