Insper

SuperComputação

Aula 11 – Granularidade e Produtor/Consumidor

2018 – Engenharia

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

Aulas passadas

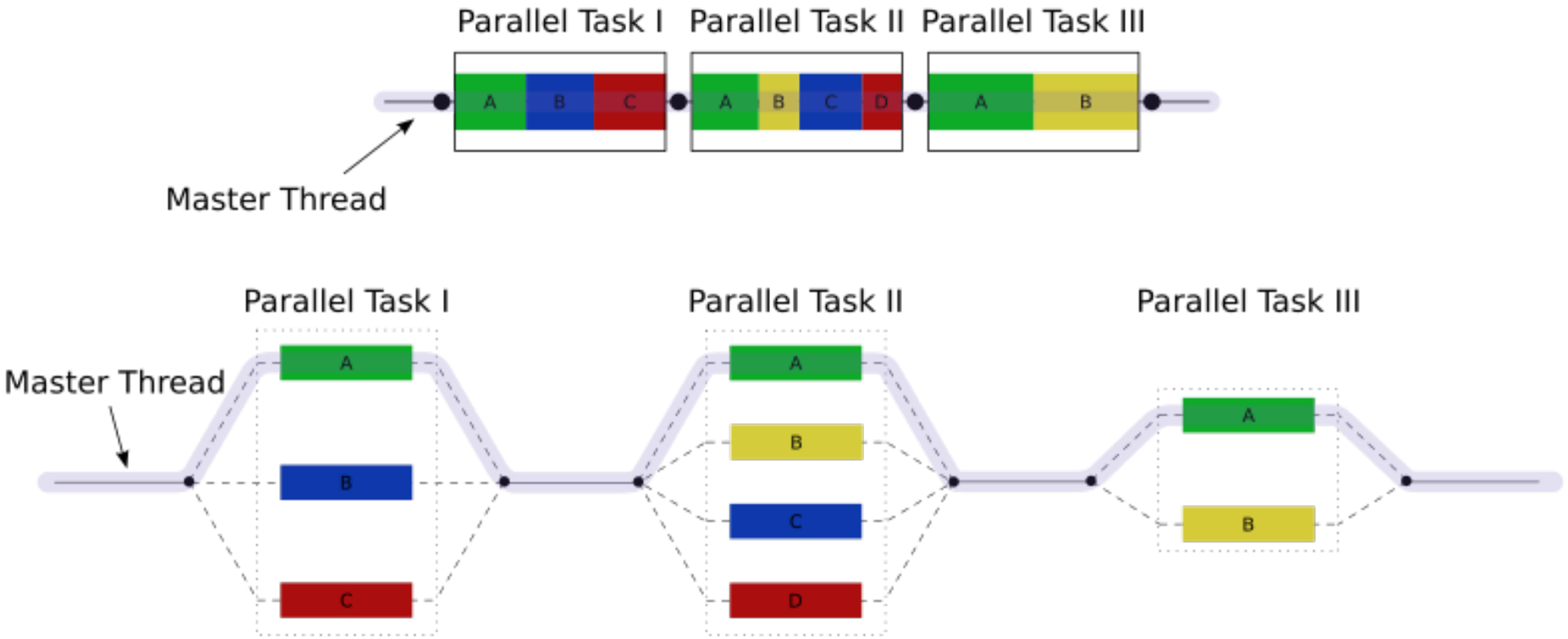


Figura: https://en.wikipedia.org/wiki/File:Fork_join.svg

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

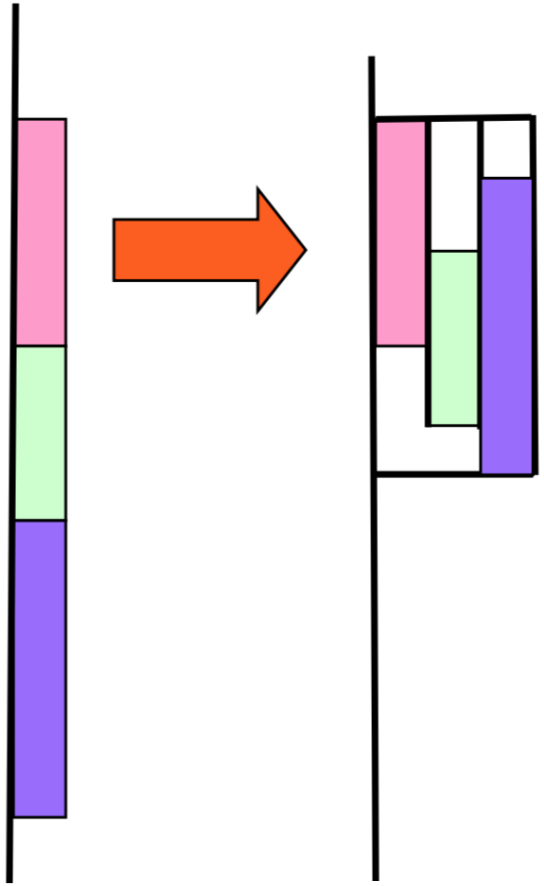
- Relação entre o tamanho da tarefa e o custo de lançar/trocar de tarefas
- Alta granularidade:
  - Tarefas pouco complexas
  - Facilmente paralelizáveis
  - Fáceis de escalonar

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

Escalonamento

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



Serial **Paralela**

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 [illegible]

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

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

Insper

OpenMP - Escalonamento

Tipo	Quando usar
S	
D	
G	
A	mesmo loop

Isto não vale para tasks! Quando crio task defino seu tamanho e, se não houver cuidado, este tamanho pode ser pequeno demais!

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

Tarefas homogêneas

- Tempo de processamento similar
- Execução independente
- Pouco uso de recursos compartilhados
- Fácil de escalonar (agrupadas em `chunks`)

Tarefas heterogêneas

- Algumas dependem de recursos compartilhados
- Algumas são ingenuamente paralelizáveis
- Algumas são inerentemente sequenciais, mas são independentes entre si.
- Progresso de uma tarefa depende de outra

Tarefas heterogêneas

- Algumas dependem de recursos compartilhados
- Alguma veis
- Alguma is, mas são independentes entre si.
- Progresso de uma tarefa depende de outra

Primitivas de sincronização

Primitivas de sincronização

- Mutex
- Barreira
- Variável de condição
- Semáforo

Exclusão mútua - Mutex

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

Variável de Condição

- Preciso de
  - Acesso Exclusivo (Mutex)
  - Condição seja verdadeira
- Útil para sincronizar progresso de threads de maneira condicional

Variável de Condição - Wait

wait (m, P): espera ter o mutex e a condição P ser verdadeira

```
while !P
    release(m)
    wait(m)
```

- Se a condição for verdadeira na primeira checagem prossegue
- Se não espera ser notificado para checar de novo

Variável de Condição - Notify

- notify_one: notifica uma thread que a condição P se tornou verdadeira

- notify_all: notifica todas as threads que a condição P se tornou verdadeira

Duas filas:

1) Threads esperando serem notificadas para checarem a condição

2)Threads esperando o mutex

Semáforo

Inteiro especial com duas funções

- Up() → soma um no valor
- Down() → se valor > 0: valor -= 1
  senão: espera até que valor > 0

Se inicializado com S, permite até S acessos concorrentes

Semáforo

- Generalização do mutex
- Pode ser construído usando Mutex e Variável de Condição
- Representa o número de unidades de um recurso
    - supondo que Up() seja seguido de Down()

API para concorrência

OpenMP

- API de alto nível para Fork-Join
- Possível usar para outros fins, mas pode conter poucos recursos
- Oferece pouco controle do progresso das threads

Primitivas de sincronização - OpenMP

- Mutex (*omp_lock_t, critical*)
- Barreira (*barrier, implícitas*)
- ~~Variável de condição~~
- ~~Semáforo~~

API para concorrência

C++ threads

- Controle explícito das threads
- Bom quando precisamos de threads que rodam durante muito tempo
- Bom para sincronizar progresso das threads

Primitivas de sincronização - C++

- Mutex (*unique_lock, shared_lock, lock_guard*)
- Barreira – fácil de implementar
- Variável de condição (`condition_var`)
- Semáforo – fácil de implementar

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

Atividade:

- Prática com sincronização em C++
- Implementação do modelo produtor-consumidor
- Entrega: 24/09

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