Insper

SuperComputação

Aula 3 – Modelo fork-join

2018 – Engenharia

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

Tarefa 3

- Objetivo: comparar desempenho de código vetorizado com código “normal”
  - Diversas funções em *tarefa3.cpp*
  - Realizem testes e escrevam um relatório mostrando seus resultados

Entrega via Blackboard para a próxima aula

Traga duas cópias impressas do seu relatório

Tarefa 3

1. Cada um receberá dois relatórios e duas fichas de avaliação;

2. Dediquem 20 minutos a cada relatório;

3. Tentem reproduzir alguns dos resultados apresentados.

Critérios

1. Descrição do problema
2. Descrição da implementação/testes
3. Reprodutibilidade
4. Tamanho das instâncias de teste
5. Medidas de desempenho
6. Facilidade de leitura/interpretação

Como avaliar um trabalho

1. Leia a rubrica com atenção

2. Leia o relatório inteiro uma vez

3. Avalie cada item da rubrica de maneira individual, consultando o relatório sempre que necessário.

    - No item Reprodutibilidade, tente reproduzir os testes feitos para a função *gauss*

4. Faça comentários construtivos no texto.

Paralelismo e concorrência

- **Concorrência**: condição de um sistema na qual múltiplas tarefas estão logicamente ativas ao mesmo tempo.
- **Paralelismo**: condição de um sistema na qual as múltiplas tarefas estão realmente ativas ao mesmo tempo.

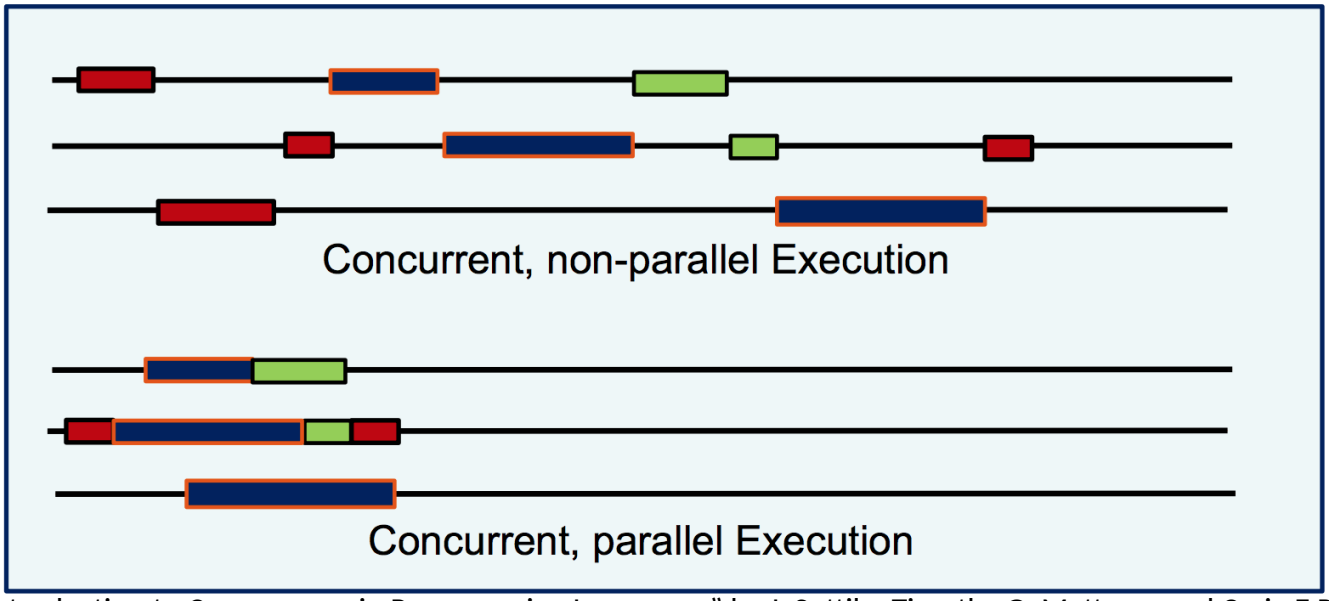

Figure from “An Introduction to Concurrency in Programming Languages” by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010

Paralelismo e concorrência

- **Paralelismo**: usado para
  - fazer mais trabalho em menos tempo
  - tratar problemas grandes

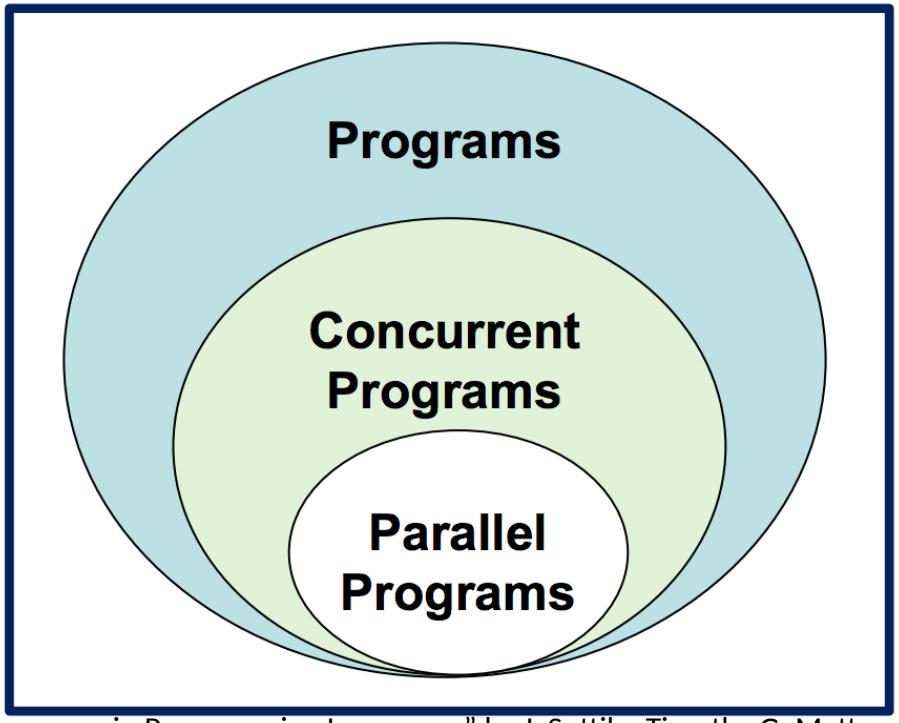

Figure from “An Introduction to Concurrency in Programming Languages” by J. Sottile, Timothy G. Mattson, and Craig E Rasmussen, 2010

Computação paralela atual

- Multicore com OpenMP
- Cluster com MPI
- GPGPU com CUDA ou OpenCL
- FPGA (Google TPU) com OpenCL (não veremos)

Paralelismo raiz

- Quais mecanismos temos para executar tarefas em paralelo?

- Quais suas diferenças?

Processos

Principal mecanismo de execução de tarefas em um SO

Tarefas completamente isoladas

É possível a comunicação entre processos via

- Entrada e saída padrão
- Passagem de mensagens
- Memória **(explicitamente)** compartilhada

Processos

Criação (POSIX):

```
pid_t fork();
```

Cria um novo processo duplicando todos os dados do processo chamador (pai). Retorna duas vezes.

1. No pai: pid do filho

2. No filho: 0

Threads

Cada processo tem um ou mais threads

Compartilham recursos entre si

Problema com sincronização ao acesso a objetos compartilhados

Threads

Em C: POSIX Threads (obsoleto?)

Em C++11: cabeçalho <thread>

1. std::thread contém implementação simples de threads

2. std::this_thread é usado para referenciar o thread atual em uma função

Threads vs Processos

Threads:

- Ambientes multi-core
- Modelo fork-join

Processos:

- Pode ser usado em ambientes distribuídos
- Compartilhamento explícito de dados

Modelo fork-join

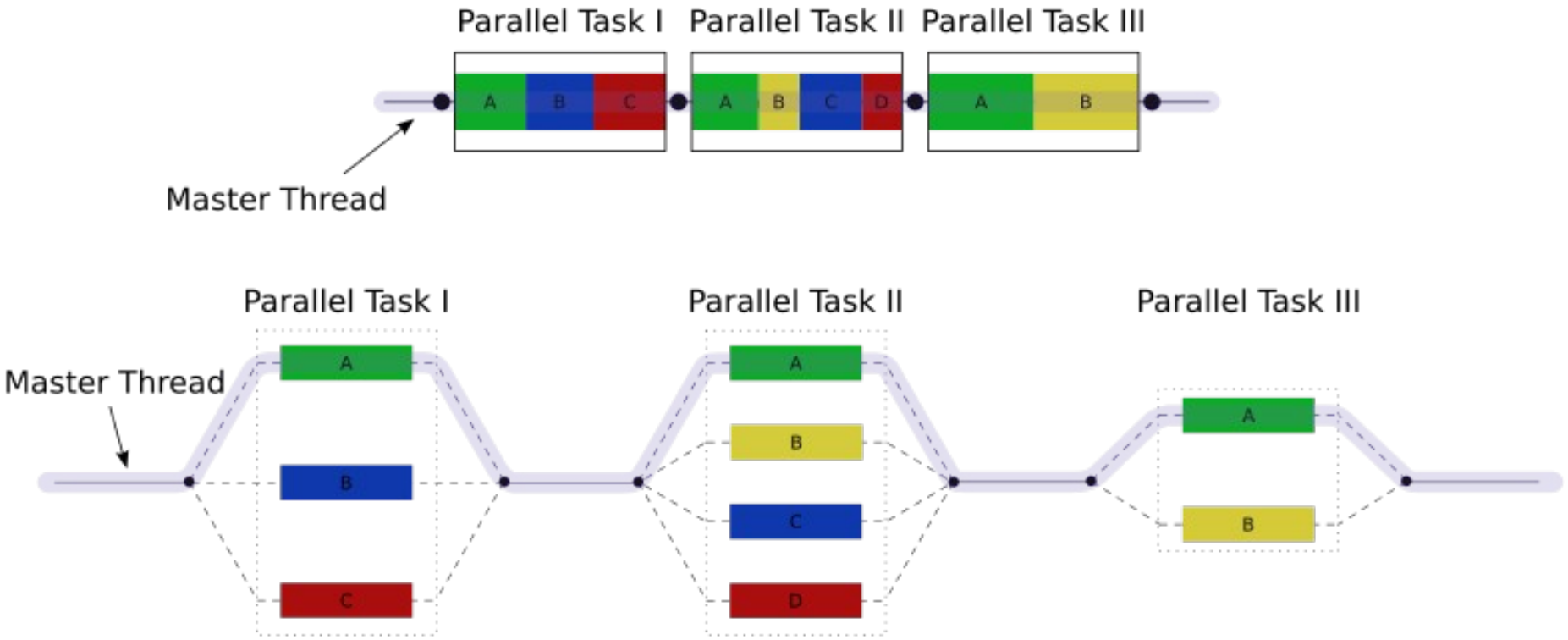

Figura: https://en.wikipedia.org/wiki/File:Fork_join.svg

Modelo fork-join raiz

Atividade de hoje: implementar modelo fork-join na mão usando std::thread.

Roteiro 1 – módulo multi-core

Referências

- Livros:
  - Hager, G. ; Wellein, G. **Introduction to High Performance Computing for Scientists and Engineers**. 1ª Ed. CRC Press, 2010.
- Internet:
  - https://en.wikipedia.org/wiki/Fork%E2%80%93join_model
  - https://en.cppreference.com/w/cpp/thread/thread
  - https://en.cppreference.com/w/cpp/thread

Insper

www.insper.edu.br