



Introdução a Paralelização de Aplicações com OpenMP

Luan Teylo

Escola Regional de Alto Desempenho do Rio de Janeiro 2021



Luan Teylo (Foto ilustrativa)

- ▶ Bacharel em Ciência da Computação (UFMT - 2015)
- ▶ Mestrado em Computação (UFF - 2017)
- ▶ Doutorado em Computação (UFF - 2021)
- ▶ Pós doutorado (Inria Bordeaux - Atual)

Sobre esse minicurso

- ▶ IV DevWeek 2016 (Leonardo Araujo e Luan Teylo)
- ▶ ERAD-RJ 2017 (Leonardo Araujo e Luan Teylo)
- ▶ ERAD-RJ 2018 (Rodrigo Alves)
- ▶ ERAD-RJ 2020 (Luan Teylo)

Códigos disponíveis em:

<https://github.com/luanteylo/minicurso-OpenMP>

Não importa o quão rápido os computadores são. Neste momento, novas tecnologias estão sendo desenvolvidas para que eles sejam ainda mais rápidos. Nosso apetite por processamento e capacidade de memória parece insaciável.

Chapman *et al.* (2008)

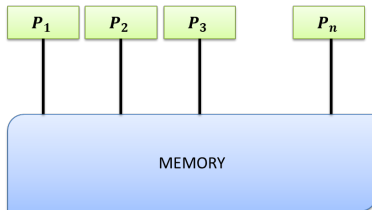


Quase duas décadas de evolução gráfica do jogo Tomb Raider (1996 à 2013).

A indústria dos jogos eletrônicos é um exemplo "visual" da demanda **constante** por mais capacidade de processamento.

Paralelismo

Na década de 1980 vários fornecedores começaram a produzir computadores que exploravam um paralelismo arquitetural. As máquinas eram construídas com **vários processadores completos** que **compartilhavam** a mesma memória principal.



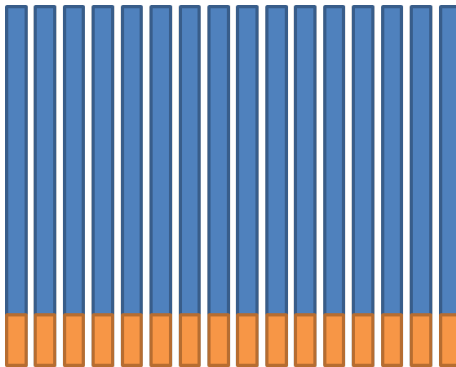
Arquitetura de memória compartilhada.

O Processo de Programação Paralela



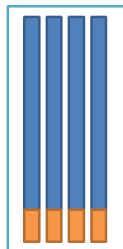
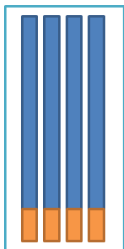
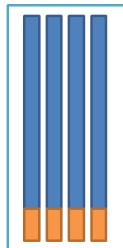
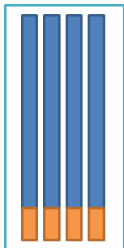
Problema Original

O Processo de Programação Paralela



Definir paralelismo

O Processo de Programação Paralela



Estratégia Algorítmica

Por que deveríamos paralelizar?

- ▶ Quantos cores tem o seu computador?
- ▶ Os programas que você escreve aproveitam TODO o poder de processamento disponível?
- ▶ Você quer ganhar desempenho?

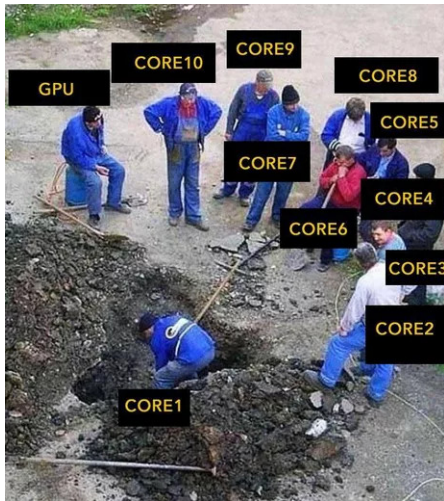
Paralelismo

Expectativa

Programas cada vez mais rápidos aproveitando todos os processadores

Paralelismo

Realidade:



Introdução ao OpenMP

Open specification for Multiprocessing

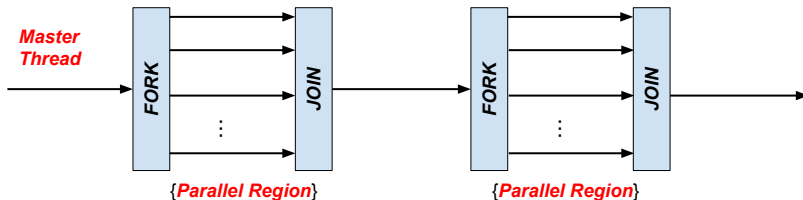
O que é OpenMP?

- ▶ API de programação multithreading com memória compartilhada
- ▶ Especificação colaborativa: indústria, governo e academia
- ▶ Composto por Diretivas, Funções e Variáveis de Ambiente
- ▶ <http://www.openmp.org/>



Sobre OpenMP

- ▶ Linguagens suportadas: Fortran e C/C++
- ▶ Mesmo código para implementação serial e paralela
- ▶ Implementação portátil e intuitiva
- ▶ Requer compilador (não é apenas uma biblioteca)



Principais Componentes do OpenMP

► Diretivas

- *#pragma omp parallel*
- *#pragma omp master*
- *#pragma omp single*
- *#pragma omp for*
- *#pragma omp critical*
- *#pragma omp atomic*
- Etc

► Biblioteca

- *omp_get_num_threads()*
- *omp_set_num_threads()*
- *omp_get_thread_num()*

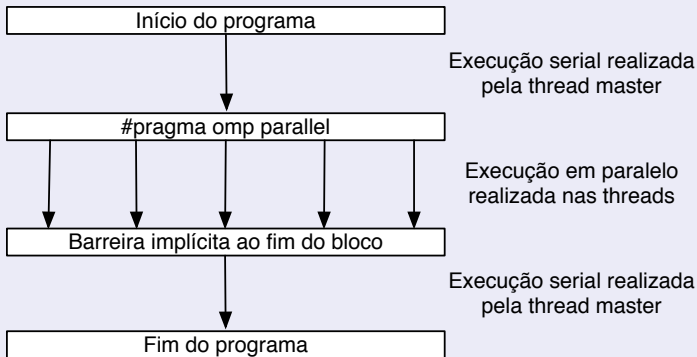
► Variáveis de ambiente

- OMP_NUM_THREADS

Expressando paralelismo

Bloco paralelo

- ▶ *#pragma omp parallel*
- ▶ Inicia bloco de código que será executado por todas as *threads* disponíveis



Exemplo

Hello World Paralelo

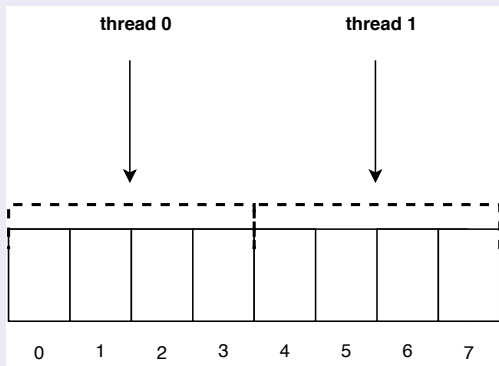
- ▶ A execução é inicializada na *thread master*
- ▶ A diretiva `#pragma omp parallel` inicia um bloco paralelo (*fork*)
- ▶ Cada thread executa os comandos contidos no bloco paralelo
- ▶ Barreira implícita ao fim do bloco (*join*)
- ▶ *Thread master* retoma o processamento serial

para compilar: `$ gcc -o hello -fopenmp hello.c`

Pergunta: O que acontece na linha 11?

Expressando paralelismo

Divisão de *loop*

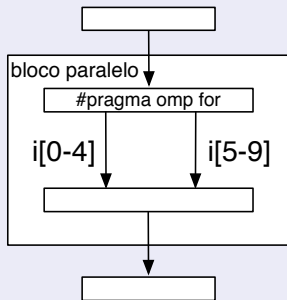


Expressando paralelismo

Divisão de *loop*

- ▶ *#pragma omp for*
- ▶ Divide as iterações de um *loop* entre as *threads* disponíveis

```
int i[10];  
omp_set_num_threads(2);
```



Contexto dos Dados

- ▶ Contexto das threads
 - ▶ Variáveis locais
 - ▶ Variáveis globais
- ▶ Variáveis privadas
 - ▶ Private
 - ▶ Firstprivate
- ▶ Variáveis compartilhadas
 - ▶ Default
 - ▶ Shared
- ▶ Redução
 - ▶ Reduction (+,-,*,MAX,MIN,etc)

Exemplo

Contexto de variáveis

- ▶ Inicialização da variável X
- ▶ Criação de 20 threads

Pergunta: O que acontece em cada um dos blocos paralelos?

Sincronização

Como estabelecer uma ordem na execução das *threads*?

Barreira

- ▶ *#pragma omp barrier*
- ▶ Todas as *threads* devem alcançar a barreira para que a execução possa seguir
- ▶ Garante consistência

Sessão crítica

- ▶ *#pragma omp critical*
- ▶ Trecho executado por somente uma *thread* por vez
- ▶ Garante exclusão mútua, evita condição de corrida

Exemplo

Redução de um vetor

- ▶ Dado um vetor de N posições
- ▶ Somar os valores contidos em cada uma das N posições

Pergunta: Como dividir este problema?

Exemplo

Contar números primos

- ▶ Dado um valor N
- ▶ Qual o número de primos no intervalo $[1, N]$?

Perguntas:

- ▶ Como dividir este problema?
- ▶ Esta solução está correta?
- ▶ Todas as *threads* terão o mesmo trabalho?

Escalonamento de Iterações

Escalonamento Estático

- ▶ Escalonamento default
- ▶ Chunksize default = número de iterações / número de threads
- ▶ *`#pragma omp parallel for schedule(static[,chunksize])`*

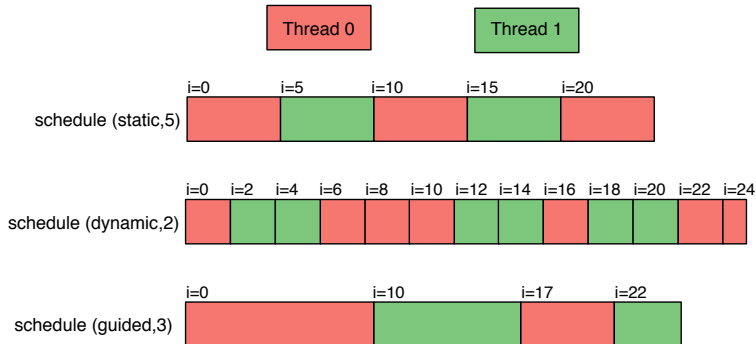
Escalonamento Dinâmico

- ▶ Chunksize default = 1
- ▶ *`#pragma omp parallel for schedule(dynamic[,chunksize])`*

Escalonamento Guiado

- ▶ Compilador define tamanho dos chunks, de forma decrescente
- ▶ Lastchunksize define o tamanho do último chunk
- ▶ *`#pragma omp parallel for schedule(guided[,lastchunksize])`*

Escalonamento de Iterações



créditos

Minicurso produzido originalmente por:

- ▶ leonardoaj@ic.uff.br
- ▶ luanteylo@ic.uff.br