
 <p>Universidade do Minho</p>	<p>Módulo 10</p> <p>Processamento Paralelo</p> <p>10/Jan/2013</p>	
--	---	---

Diretivas do OpenMP:

- **omp parallel** → Faz com que todos os núcleos disponíveis executem o bloco delimitado pela diretiva ⇔ cria atividades paralelas (*threads*), cada uma alocada a um núcleo.
- **omp parallel for** → Quando o bloco delimitado pela diretiva é um ciclo *for*, esta diretiva dita a divisão do trabalho (i.e., as iterações do ciclo) pelos vários núcleos/*threads*.

Funções do OpenMP:

- A função **omp_get_thread_num** devolve um inteiro diferente (identificador) para cada núcleo/*thread*.
- **omp_get_num_threads** devolve o total de *threads* em execução.
- A função **omp_set_num_threads(nThreads)** define o número de *threads* que vão ser usadas na execução paralela.

(a)

- Importa verificar a forma como as iterações são mapeadas nos vários núcleos.
- Nota: usa-se o termo **núcleo**, assumindo, para simplificar, que uma **thread** é executada sempre no mesmo núcleo.
- Assumindo que dispomos de 2 núcleos, o núcleo 0 processa as iterações/colunas 1...127 e o núcleo 1 as iterações 128...256.
- Cada núcleo executa a sua gama de iterações de forma sequencial.
- Como o processamento é feito em paralelo, o "output" gerado pelos vários núcleos aparece misturado e com cadências diferentes, influenciadas pela carga computacional de cada núcleo.

(b) Nota: deve medir-se o tempo de execução retirando o **printf** introduzido em (a).

Resultados (com 4 núcleos):

$T_{\text{exec_sequencial}} \approx 687 \mu\text{s}$

$T_{\text{exec_paralelo}} \approx 457 \mu\text{s}$

$\text{ganho}_{\text{par sobre seq}} = T_{\text{exec_sequencial}}/T_{\text{exec_paralelo}} \approx 687/457 \approx 1,5$

Este ganho está longe do valor ótimo, que seria 2,0. Em geral é difícil conseguir o valor ótimo. Um dos responsáveis é a sobrecarga da gestão do paralelismo: *overhead* associado à criação de *threads*/tarefas, etc.

- (c) Esta alínea ilustra a lei de Amdahl.

Lei de Amdahl

A **aceleração** de um programa utilizando múltiplos processadores é limitada pelo tempo necessário para executar a parte sequencial do programa. Por exemplo, se um programa necessita de **20 horas** utilizando um único núcleo de processador, e uma determinada parte do programa (que demora a executar **1 hora**) não poder ser paralelizada, mas o resto do programa (que demora **19 horas**, ou seja, 95%) poder ser paralelizada, então, independentemente do número de processadores utilizados na execução paralela do programa, o tempo de execução mínimo não pode ser menor do que a tal 1 hora crítica. Daqui resulta que a aceleração está limitada a **20**.

A lei de Amdahl estabelece que se **P** é a parte dum programa que pode ser paralelizada e **(1 - P)** é a parte que não pode ser paralelizada, então a aceleração máxima **S** que se pode atingir com a utilização de **N** processadores é:

$$S(N) = \frac{1}{(1 - P) + \frac{P}{N}}$$

No limite, quando **N** tende para infinito, a aceleração máxima **S** tende para **1/(1-P)**.

Resultados para a execução global do programa, com **convolve3x1** sequencial OU paralelo (com 4 núcleos):

$$T_{\text{exec_global_convolve3x1_seq}} \approx 702 \mu\text{s}$$

$$T_{\text{exec_global_convolve3x1_paralelo}} \approx 565 \mu\text{s}$$

$$\text{ganho}_{\text{par sobre seq}} = T_{\text{exec_global_convolve3x1_seq}} / T_{\text{exec_global_convolve3x1_paralelo}} \approx 702/565 \approx 1,24$$

Apesar do ganho em **convolve3x1** ser **1,5**, o ganho **global** é apenas **1,24** porque o resto do código é sequencial. Por exemplo, a leitura e escrita em ficheiro não são efetuadas em paralelo.

- (d) Trocando a ordem dos ciclos for (ciclo em Y no exterior), a versão sequencial fica mais rápida que a usada na alínea (b). Isto ocorre porque a nova versão (d) é mais amigável da memória do que (b): em (d) percorre-se a matriz **I[[]]** por linhas em vez de colunas. Embora a paralelização ainda acelere a execução sequencial, ao usar 4 núcleos o ganho é menor que em (b). O que ocorre com 2 núcleos?

Resultados (com 4 núcleos):

$$T_{\text{exec_sequencial}} \approx 305 \mu\text{s}$$

$$T_{\text{exec_paralelo}} \approx 293 \mu\text{s}$$

$$\text{ganho}_{\text{par sobre seq}} = T_{\text{exec_sequencial}} / T_{\text{exec_paralelo}} \approx 305/293 \approx 1,04$$

(e)

Ordem de grandeza dos resultados (com 4 núcleos):

$T_{\text{exec_sequencial}} \approx 305 \mu\text{s}$

$T_{\text{exec_paralelo}} \approx 4000 \mu\text{s}$ *(é muito variável)*

Ao paralelizar os 2 ciclos *for*, o tempo de execução aumenta bastante porque temos mais tarefas, logo temos maior sobrecarga de gestão.

(f) Ao acrescentar a cláusula **schedule(dynamic)** na diretiva **#pragma omp parallel for**, as iterações do ciclo são atribuídas aos núcleos de forma dinâmica (ao que estiver mais livre). Esta opção implica um custo de gestão maior, logo o tempo de execução aumenta. Ou seja, aqui também não temos ganho em T_{exec} . Esta opção seria útil para balancear melhor a carga dos núcleos (por exemplo, se o tempo de execução da cada iteração não fosse homogéneo).

Ordem de grandeza dos resultados (com 4 núcleos):

$T_{\text{exec_sequencial}} \approx 644 \mu\text{s}$

$T_{\text{exec_paralelo_dinâmico}} \approx 3400 \mu\text{s}$ *(é muito variável)*