Simulação de Modelos de Difusão de Contaminantes em OpenMP
Análise de Desempenho da Versão Serial x Paralela

Arthur L. A. Silva, Evandro K. Kayano, Yuri S. Bastos

[1]Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)

1. Introdução

A velocidade de execução de cálculos é uma das maiores qualidades dos computadores. Por isso, tarefas como simulações que demoraria diversas horas para um humano pode ser executada por uma máquina em poucos segundos. Entretanto, mesmo na execução de máquina, uma tarefa de complexidade muito grande ou com um número grande de amostragem pode ter um desempenho não desejado para tarefas que exijam respostas rápidas. Para isso, pode-se muitas vezes dividir a execução desta tarefa, paralelizando-a em *threads*.

Este trabalho aborda um problema de simulação de contaminantes em um corpo d'água. Para isso, foi utilizado um programa em C com uso de lógicas de paralelização com o uso da biblioteca OpenMP para agilizar o tempo de resposta do software. No fim, foi analizado e comparado o desempenho deste programa executado de forma serial e paralelizada.

2. Desenvolvimento

Foi utilizado como base para este projeto o código de simulação em execução serial em C disponibilizado pelos professores. Este programa base discretiza uma equação diferencial de difusão/transporte de contaminantes em um copo d'água utilizando diferenças finitas em uma grade bidimensional. Para isso cada valor da matriz atualiza seu valor a cada iteração com base nos valores de posições vizinhas, e as iterações indicam o avanço do tempo.

Por ser uma estrutura de dados grande (matriz de 2000x2000), e serem feitas 500 iterações temporais, a execução serial do código pode não ter o desempenho desejado. Para ser feita a paralelização, as iterações do laço de linhas (e colunas, por estarem em um laço interno) foram dividas igualmente entre as threads a serem utilizadas. Para isso, foi utilizada a diretiva *#pragma omp parallel for collapse(2)* no primeiro conjunto de for e *#pragma omp parallel for collapse(2) reduction(+:difmedio)* no segundo conjunto. A cláusula *reduction(+:difmedio)* faz com que a variável difmedio, responsável por armazenar a diferença média entre os dados finais e iniciais em uma iteração, combine os resultados parciais de cada *thread* em um único valor global para evitar condições de corrida na operação de somatória da diferença média.

A cada 100 iterações temporais é impresso o valor da diferença média. Este valor, caso seja igual na versão serial e paralelizada, pode comprovar a compatibilidade entre as versões.

3. Resultados

3.1. Especificações da máquina utilizada para testes

Para a realização dos testes deste trabalho, foi utilizado um processador *AMD Ryzen 7 5700u @ 1.8GHz* [8 núcleos reais e 16 *threads*]

3.2. Tempo de Execução Serial x Paralelizado

Abaixo, a tabela demonstra os tempos de execução para cada cenário de execução em relação à quantidade de threads assim como as métricas de eficiência e speedup:

Table 1. Threads X Tempo, Eficiência e SpeedUp

Threads	Tempo (s)	Speedup	Eficiencia
1	21,6584	1,000000	100,00%
2	17,0364	1,271301	63,57%
4	7,9932	2,709603	67,74%
8	6,9052	3,136535	39,21%
16	6,263	3,458151	21,61%

Além disso, a execução do código serializado teve o mesmo tempo aproximado de execução que a execução do código paralelizado com 1 thread. Com essas informações, é possível traçar os seguintes gráficos de eficiência e ”speedup” x ”speedup linear”:

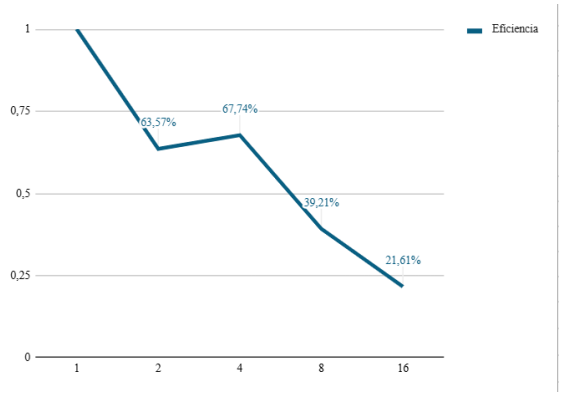

Figure 1. Gráfico de eficiência de acordo com as métricas

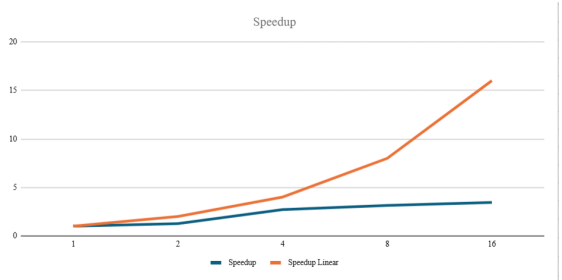

Figure 2. Gráfico de speedup de acordo com as métricas

3.3. Compatibilidade

A compatibilidade do software é comprovada pelos valores de diferença média e concentração final serem equivalentes na versão serial e paralelizada.

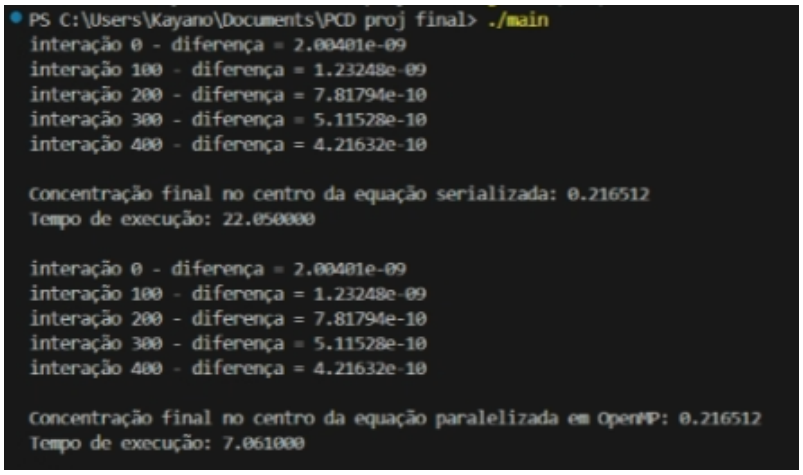

Figure 3. Comparação de diferença média na execução serial e paralela

4. Conclusão

Com isso, analisando as informações obtidas, podemos concluir que a compatibilidade do software foi mantida, assim confirmando a funcionalidade da versão paralela. Dadas as métricas, é possível verificar que o código teve sua maior eficiência quando executado de forma paralelizada com 4 threads, enquanto seu maior speedup foi obtido quando executado de forma paralelizada com 8 threads; além disso, foi observado que o melhor tempo foi obtido com a execução paralelizada em 16 threads. Também foi possível verificar que, ao tentar realizar testes de forma paralelizada com quantidade de threads superior a 16, o tempo de execução se manteve aproximadamente estável, o que pode ser explicado pela quantidade de threads disponíveis no hardware utilizado para realização desse estudo.