Simulação de Modelos de Difusão de Contaminantes em OpenMPI
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* ou processos.

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 OpenMPI para agilizar o tempo de resposta do software. No fim, foi analisado e comparado o desempenho deste programa executado com um e múltiplos processadores, e então com soluções utilizando as ferramentas OpenMP e CUDA.

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 dividir o processamento desta tarefa entre múltiplos processos, o programa recebe como parâmetro o número de processos a serem utilizados com *MPI_Comm_size()*. Este valor é utilizado para calcular a quantidade de linhas que cada processo é responsável por calcular. Como o cálculo é feito utilizando elementos vizinhos da matriz, então é necessário que haja troca de mensagens entre os processos para que as linhas calculadas em um processo possam ser lidas pelos seus processos de *rank* imediatos. Para que essa troca de mensagens seja possível, são utilizadas as funções não-bloqueantes *MPI_Isend()* para enviar dados e *MPI_Irecv()* para receber dados, e então é chamada a função *MPI_Waitall()* para garantir que haja sincronização dos dados das requisições (até dois pares de *MPI_Isend()* e *MPI_Irecv()*) feitas e que os cálculos de difusão média não utilizem valores não atualizados. Com isso, cada processo consegue calcular a difusão dos elementos de suas linhas correspondentes.

A cada 100 iterações temporais, o processo de *rank* 0 imprime o valor da difusão média. Este valor, caso seja igual na versão serial e paralelizada, pode comprovar a compatibilidade entre as versões serial e paralelizadas.

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. Tempos de Execução Serial e Concorrente

A tabela abaixo demonstra os tempos de execução, speedup e eficiência para cada cenário de execução em relação à quantidade de processos:

Processos	Tempo(s)	Speedup	Eficiencia
1	25,559583	1,000000	100,00%
2	16,006543	1,596821	79,84%
4	11,076580	2,307534	57,69%
8	7,694598	3,321757	41,52%
16	7,607713	3,359693	21,00%

Figure 1. Processos X Tempo, Eficiência e SpeedUp

Podemos dizer que a execução com somente 1 processo é o mesmo cenário de uma execução serial. Com isso, é possível esboçar os gráficos de eficiência e ”speedup” x ”speedup linear”:

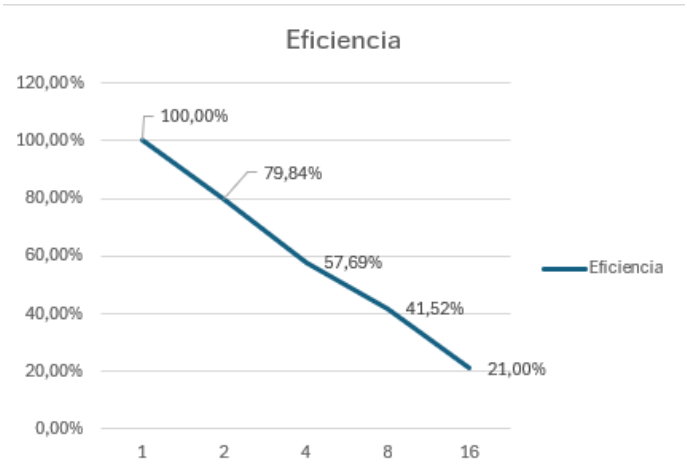

Figure 2. Gráfico de eficiência de acordo com as métricas

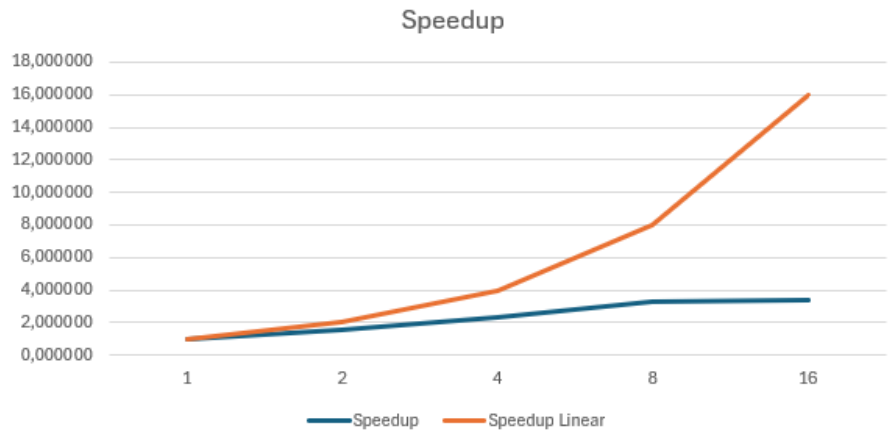

Figure 3. Gráfico de speedup de acordo com as métricas

Também podemos traçar comparativos das métricas de tempo, speedup e eficiência com as soluções desenvolvidas anteriormente utilizando paralelismo com OpenMP e CUDA:

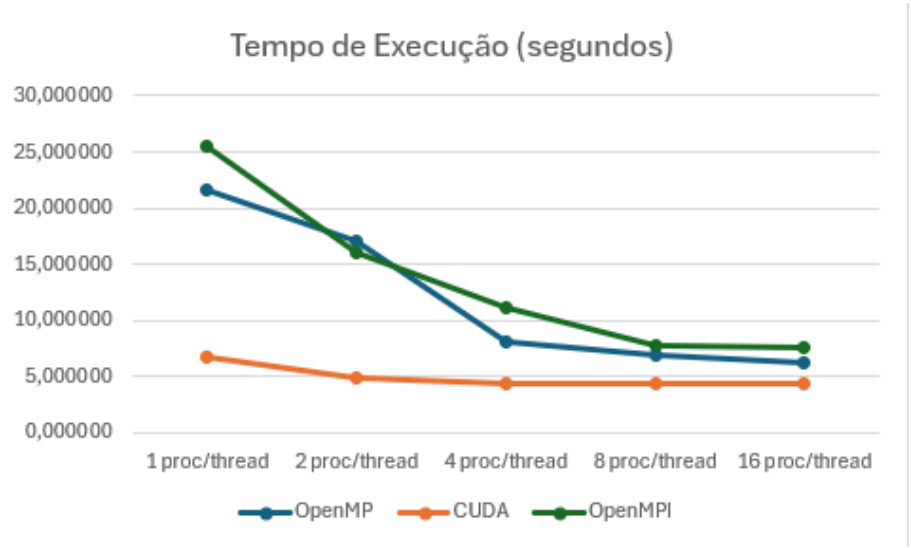

Figure 4. Tempo de Execução OpenMP X CUDA X OpenMPI

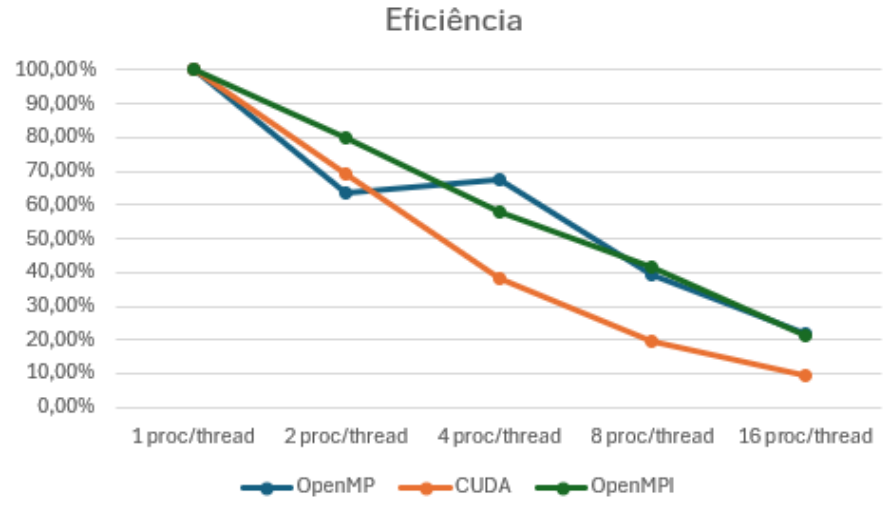

Figure 5. Eficiência OpenMP X CUDA X OpenMPI

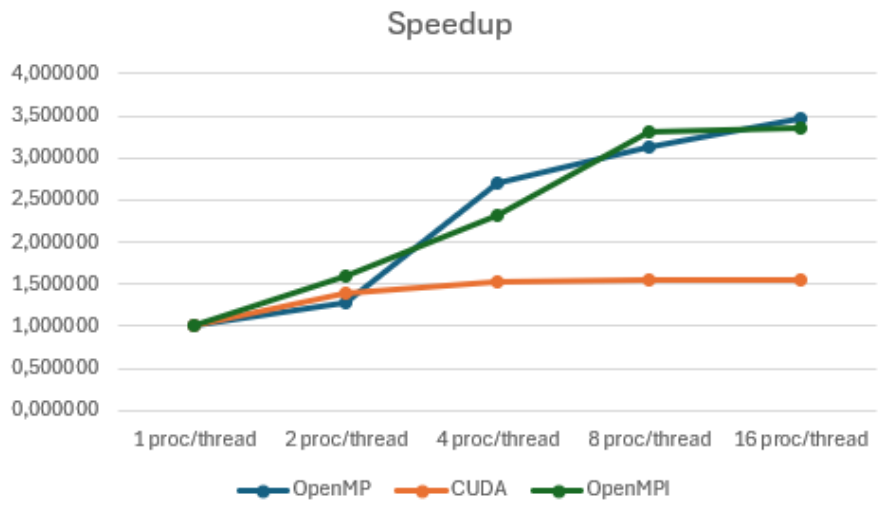

Figure 6. Speedup OpenMP X CUDA X OpenMPI

3.3. Compatibilidade

A compatibilidade da solução desenvolvida é comprovada pelos valores de diferença média e concentração final serem iguais nas execuções com 1 e 16 processos:

```
evandro@evandro-IdeaPad-3-15ALC6:~/Documentos/ProjetoPCD$ mpirun -np 1 ./main
interação 0 - diferenca = 2.00401e-09
interação 100 - diferenca = 1.23248e-09
interação 200 - diferenca = 7.81794e-10
interação 300 - diferenca = 5.11528e-10
interação 400 - diferenca = 4.21632e-10
Concentração final no centro: 0.216512
Tempo de execução: 25.289895 segundos
```

Figure 7. Diferença média e concentração final - 1 Processo

```
evandro@evandro-IdeaPad-3-15ALC6:~/Documentos/ProjetoPCD$ mpirun --oversubscribe -np 16 ./main
interação 0 - diferenca = 2.00401e-09
interação 100 - diferenca = 1.23248e-09
interação 200 - diferenca = 7.81794e-10
interação 300 - diferenca = 5.11528e-10
interação 400 - diferenca = 4.21632e-10
Concentração final no centro: 0.216512

Tempo de execução: 7.599090 segundos
```

Figure 8. Diferença média e concentração final - 16 Processos

4. Conclusão

Ao analisar as métricas obtidas somente do programa executado de forma concorrente de acordo com a quantidade de processos, é possível observar que há uma queda aproximadamente linear na eficiência, conforme aumenta a quantidade de processos utilizados. Já seu speedup aumenta consideravelmente até a execução com 8 processos; após isso, é possível observar que a tendência é o valor se manter estável. É importante ressaltar,

todavia, que apesar da eficiência ter redução com o aumento do número de processos, até 8 processos tem-se a diminuição do tempo de execução.

Comparando-se o código concorrente com as versões paralelas utilizando OpenMP e CUDA, pode-se observar que o tempo de execução em CUDA é significativamente menor que as outras opções, entretanto a taxa de redução é relativamente baixa com o aumento da quantidade de threads em uso. Contudo ao comparar com as versões OpenMP e OpenMPI, pode-se observar que a taxa de redução do tempo é muito maior com o aumento de processos/threads. Já a eficiência diminui de forma semelhante nas três modalidades, com CUDA tendo uma métrica relativamente menor que as outras opções para quantidades de threads/processos maior que 2. Por fim, ao analisar o speedup comparativo, é observado que para até 2 processos/threads a métrica cresce de maneira semelhante e para quantidades superiores as versões em OpenMP e OpenMPI crescem de maneira muito superior à versão CUDA, que se mantém quase estável.