Simulação de Modelos de Difusão de Contaminantes em CUDA
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 vantagens dos computadores. Por isso, tarefas como simulações que demorariam diversas horas para serem realizadas por um humano podem ser concluídas por uma máquina em poucos segundos. Entretanto, mesmo com a execução computacional, tarefas de alta complexidade ou que envolvem um grande volume de dados podem apresentar um desempenho insatisfatório para aplicações que exigem respostas rápidas. Nesse contexto, a paralelização surge como uma solução para dividir o processamento entre múltiplos núcleos ou unidades de processamento.

Este trabalho aborda um problema de simulação de contaminantes em um corpo d'água. Para isso, foi desenvolvido um programa em C utilizando a tecnologia CUDA, que permite a execução paralela de tarefas em GPUs (Graphics Processing Units) para reduzir significativamente o tempo de resposta do software. Ao final, foi analisado e comparado o desempenho deste programa em sua execução 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, foram alocadas memórias no host de dimensão 2000x2000 para passar a matriz inicial e para receber a matriz resultado. Em seguida alocamos 2 memória no devido usando *CudaMalloc*, 1 matriz para copiar o *input*, 1 matriz para receber o *output* e um espaço para receber um valor *float*, que será o teste de compatibilidade, a difusão média da matriz

Em seguida é feito o laço *for* com as T iterações, neste caso 500. Dentro do laço são chamados dois *kernels*: *update_matriz* que faz o cálculo da difusão sobre a matriz e *update_dif_medio* que faz o somatório do valor absoluto da diferença entre o novo valor da matriz e o valor antes de ser feito o cálculo da difusão.

Para verificar a compatibilidade da implementação, a cada 100 iterações é impresso o valor da difusão média da matriz.

Por fim é feito a liberação dos espaços de memória tanto no *device* quanto no *host* usando *free* e *cudaFree*.

3. Resultados

3.1. Especificações da máquina utilizada para testes

GPU utilizada: NVIDIA Tesla T4

- Arquitetura: Turing
- CUDA Cores: 2,560
- Tensor Cores: 320
- VRAM: 16 GB GDDR6

3.2. Tempo de Execução Serial x Paralelizado

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

Threads	Tempo(s)	Speedup	Eficiencia
1	6,633510	1,000000	100,00%
2	4,784174	1,386553	69,33%
4	4,373969	1,516588	37,91%
8	4,286109	1,547677	19,35%
16	4,280073	1,549859	9,69%

Figure 1. Threads X Tempo, Eficiência e SpeedUp

A execução do código em série teve o mesmo tempo de execução que a execução do código paralelizado fazendo uso somente de 1 thread. Logo, podemos esboçar os seguintes gráficos de eficiência e "speedup" x "speedup linear":

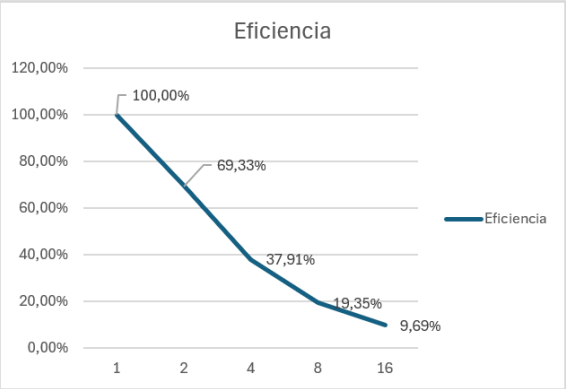

Figure 2. Gráfico de speedup de acordo com as métricas

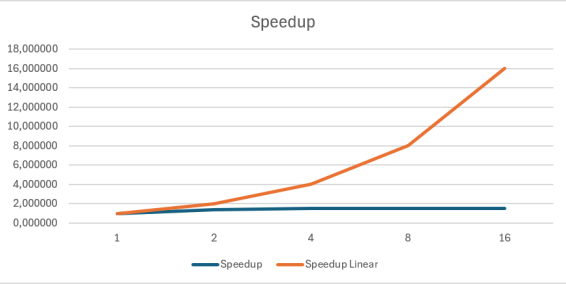

Figure 3. Gráfico de speedup de acordo com as métricas

3.3. Compatibilidade

A compatibilidade do software é comprovada pelos valores de diferença média no decorrer dos laços e concentração final serem equivalentes na versão serial e paralelizada, conforme abaixo:

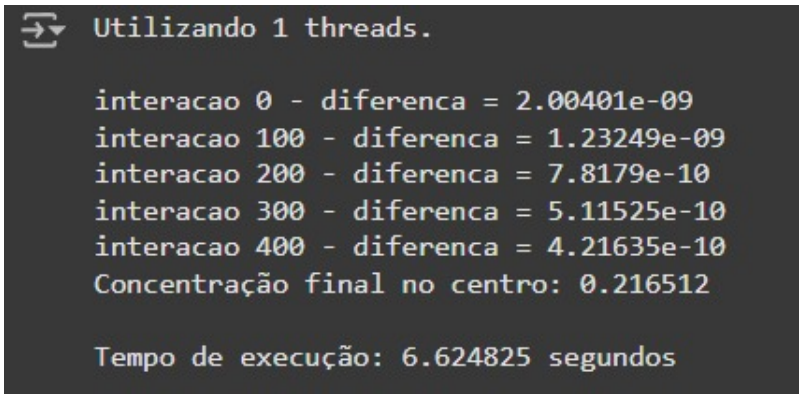

Figure 4. Diferença média na execução serial

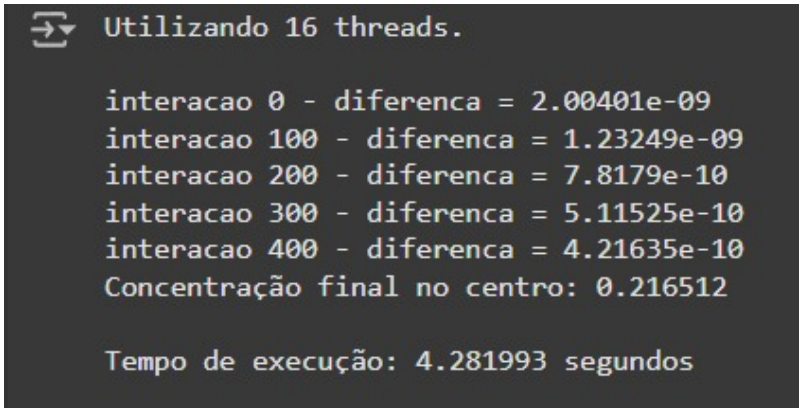

Figure 5. Diferença média na execução paralela

4. Conclusão

Ao analisar as informações obtidas, concluímos que a compatibilidade do software foi preservada, confirmando assim a funcionalidade da versão paralela. Com base nas métricas, observou-se que o código atingiu sua maior eficiência com quando executado com 1 thread e maior speedup quando executado com 16 threads paralelas. Além disso, o menor tempo de execução também foi alcançado ao utilizar 16 threads. Verificou-se que, ao realizar testes com mais de 16 threads, o tempo de execução permaneceu praticamente constante, o que pode ser atribuído à quantidade de threads suportadas pelo hardware utilizado neste estudo