Simulação e Análise de Modelos de Difusão de Contaminantes em Água

Amanda A. A. Zago[1], **Éric F. C. Yoshida**[1], **Henrique C. Garcia**[1]

[1]Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)
São José dos Campos – SP – Brazil

azago@unifesp.br, eric.fadul@unifesp.br, henrique.garcia@unifesp.br

Abstract. *This report discusses the simulation of the diffusion equation using the finite difference method, parallelized with OpenMP and CUDA. Initially, we present the diffusion model, highlighting the importance of the transient diffusion equation in describing the spread of contaminants in water bodies. We then detail the simulation environment configuration, including the two-dimensional grid, parameters such as the diffusion coefficient D, boundary/initial conditions, and the 1000 time iterations. Next, we describe CPU (OpenMP) and GPU (CUDA) implementations to parallelize calculations, optimize performance, and reduce execution time. The codes are presented and analyzed, emphasizing the improvements compared to the sequential version.*

Resumo. *Este relatório aborda a simulação da equação de difusão com método de diferenças finitas, paralelizada com OpenMP e CUDA. Apresenta-se o modelo de difusão, destacando a importância da equação de difusão transiente no espalhamento de contaminantes. Em seguida, descrevem-se as definições da grade 2D, parâmetros (coeficiente de difusão D, condições de contorno e iniciais) e 1000 iterações temporais. Por fim, mostra-se a implementação em CPU (OpenMP) e GPU (CUDA), ilustrando ganhos de desempenho em relação à versão sequencial.*

1. Estudo do Modelo de Difusão

A equação de difusão descreve como substâncias se espalham ao longo do tempo e do espaço. Em duas dimensões, a forma transiente é:

$$\frac{\partial C}{\partial t} = D \cdot \nabla^2 C, \tag{1}$$

onde C é a concentração do contaminante, t é o tempo e D é o coeficiente de difusão. O termo $\nabla^2 C$ representa a variação espacial da concentração. Usamos o método das diferenças finitas para discretizar as derivadas parciais em uma grade 2D, aproximando as variações contínuas por valores em células discretas.

2. Configuração do Ambiente e Parâmetros da Simulação

A aproximação por diferenças finitas centrais para resolver a equação de difusão em 2D é dada por:

$$C_{i,j}^{t+1} = C_{i,j}^{t} + D\,\Delta t \left(\frac{C_{i+1,j}^{t} + C_{i-1,j}^{t} + C_{i,j+1}^{t} + C_{i,j-1}^{t} - 4C_{i,j}^{t}}{\Delta x^2} \right). \tag{2}$$

Grade 2D:

- $N \times N$ células, $N = 16$.
- Cada célula representa a concentração de contaminantes em uma região do corpo d'água.

Parâmetros Principais:

- **Coeficiente de difusão** (D): 0,1.
- **Condições de contorno**: Neumann homogêneas (barreiras).
- **Condições iniciais**: concentração zero em toda a grade, exceto na região central ($C[N/2][N/2] = 1{,}0$).
- **Número de iterações**: $T = 1000$.
- **Passos espaciais e temporais**:
  - $\Delta x = 1{,}0$.
  - $\Delta t = 0{,}01$.

2.1. Máquinas Utilizadas

- **Máquina 1:** Intel Core i7-1360P (16 núcleos) a 2.2 GHz.
- **Máquina 2:** Intel Core i7-1165G7 (8 núcleos) a 2.8 GHz.
- **Máquina 3:** GPU NVIDIA T4 (16 GB de VRAM) via Google Colab.

3. Implementação com OpenMP (CPU)

Utilizamos OpenMP para paralelizar os loops que atualizam as concentrações em múltiplos núcleos.

3.1. Código não paralelizado

Disponível em: codigo_nao_paralelizado.c.

3.2. Implementação Paralela com OpenMP

Disponível em: codigo_paralelizado.c.

3.3. Explicação das Modificações

1. **Inclusão de `<omp.h>`** para funcionalidades OpenMP.
2. **Paralelização de loops** usando `#pragma omp parallel for collapse(2)` para inicialização e atualização de `C_new`.
3. **Redução** com `reduction(+:difmedio)` para somar as diferenças médias.
4. **Gerenciamento de memória** para liberar recursos ao final.

4. Implementação com CUDA (GPU)

Desenvolvemos uma versão CUDA que explora o paralelismo massivo das GPUs. A matriz de concentração é alocada na memória do host (CPU) e do dispositivo (GPU), e as principais rotinas (*kernels*) realizam o passo de difusão (`diffusion_step`) e a atualização (`update_matrix`).

4.1. Implementação em CUDA

Disponível em: diffusion_cuda.cu.

4.2. Observações

- Diferença principal: alocação/transferência de dados entre CPU e GPU.
- Cada iteração é realizada por milhares de *threads* na GPU.
- Usamos blocos 2D de 16×16 para cobrir toda a malha de $N \times N$.

5. Resultados (OpenMP)

Testamos em duas máquinas, com tempos de execução para diferentes números de *threads*. Os resultados mostram redução significativa de tempo até o limite dos núcleos físicos.

Threads	Máquina 1	Máquina 2
1	12.45	34.00
2	6.84	18.90
4	6.20	9.55
8	3.94	8.77
16	3.66	32.39

5.1. Gráficos de Speedup e Eficiência (OpenMP)

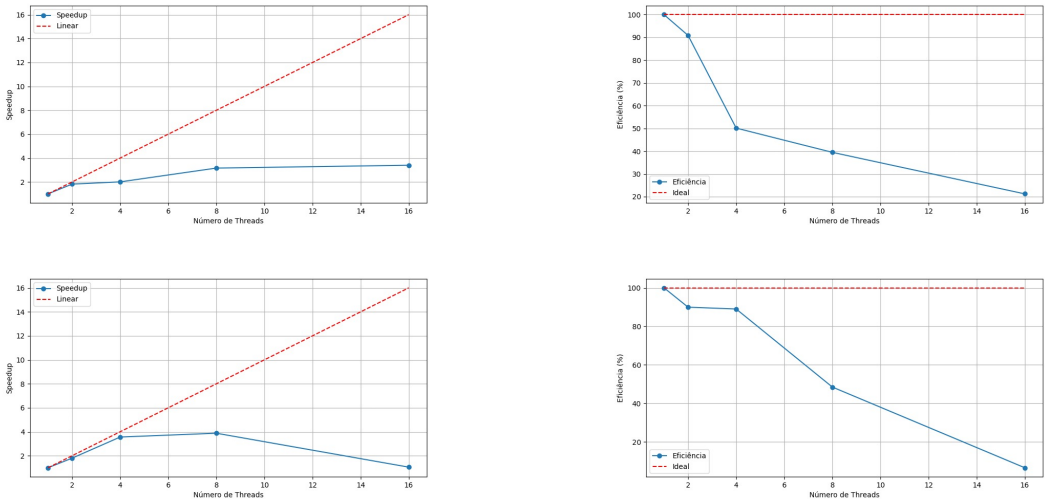

Figura 1. Desempenho nas Máquinas 1 e 2 (OpenMP)

5.2. Análise

O *speedup* cresce com o número de *threads* até o número de núcleos físicos. Na Máquina 1 (16 núcleos), o melhor desempenho ocorreu com 8 *threads*, pois o *hyper-threading* não agregou ganhos além desse ponto. Na Máquina 2 (8 núcleos), o uso de 16 *threads* aumentou o tempo devido à sobrecarga de gerenciamento. A eficiência permaneceu em torno de 90% até o limite de núcleos físicos.

6. Resultados (CUDA) na Máquina 3

Executando a versão CUDA na GPU T4 (Máquina 3) para $N = 16$ e $T = 500$:

Iteração	Diferença Média
0	2.00401×10^{-9}
100	1.23248×10^{-9}
200	7.81794×10^{-10}
300	5.11528×10^{-10}
400	4.21632×10^{-10}
Concentração final no centro: 0.216512 Tempo de execução (CUDA): 0.569653 segundos	

A diferença média diminui conforme a simulação avança, indicando convergência. A concentração final no centro foi de aproximadamente 0.216512 após 500 iterações, e o tempo total de cerca de 0.57s confirma o ganho de velocidade em GPUs.

7. Conclusão

A simulação da equação de difusão por diferenças finitas, paralelizada em CPU (OpenMP) e GPU (CUDA), obteve ganhos significativos de desempenho. Em CPU, a eficiência alcançou cerca de 90% até o limite de núcleos físicos. A versão CUDA demonstrou alto potencial de aceleração, como visto na GPU T4, que finalizou a simulação (malha de 16×16, 500 iterações) em aproximadamente 0.57s. Assim, tanto OpenMP quanto CUDA se mostram valiosas para reduzir tempos de execução em problemas de grande escala.

8. Referências

[1] Chapra, S. C., & Canale, R. P. (2011). *Métodos Numéricos para Engenharia* (7ª ed.). McGraw-Hill.

[2] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3ª ed.). Cambridge University Press.

[3] OpenMP Architecture Review Board. (2018). *OpenMP Application Programming Interface Version 5.0*. `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf`