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—which describes how contaminants spread in water—using the finite difference method. Several implementations have been developed: a sequential version; a version parallelized with OpenMP; a GPU-accelerated version with CUDA; a distributed version using MPI; and finally, a hybrid implementation combining MPI, CUDA, and OpenMP. The objective is to compare the performance of each approach, highlighting the improvements achieved by parallelism. The report is written in accessible language so that readers with little background in these technologies can understand the concepts and results.*

Resumo. *Este relatório discute a simulação da equação de difusão, que descreve como contaminantes se espalham em corpos d'água, utilizando o método das diferenças finitas. Foram desenvolvidas diversas implementações: uma versão sequencial; uma paralelizada com OpenMP; uma acelerada com CUDA; uma distribuída com MPI; e, por fim, uma implementação híbrida que integra MPI, CUDA e OpenMP. O objetivo é comparar o desempenho de cada abordagem, evidenciando os ganhos obtidos com o uso do paralelismo. O relatório foi escrito de forma acessível para que leitores sem conhecimento avançado nas tecnologias possam compreender os conceitos e os resultados.*

1. Introdução

A equação de difusão é fundamental para entender o espalhamento de substâncias no tempo e no espaço. Contudo, sua resolução numérica demanda grande poder computacional, principalmente em simulações de alta resolução. Para acelerar os cálculos, diversas técnicas de paralelismo podem ser empregadas:

- **OpenMP:** Paraleliza a execução em múltiplos núcleos de uma CPU.
- **CUDA:** Explora o paralelismo massivo das GPUs.
- **MPI:** Distribui a carga de trabalho entre vários nós (computadores) de um cluster.
- **Híbrido (MPI+CUDA+OpenMP):** Combina as vantagens das três abordagens, distribuindo a simulação entre nós e, em cada nó, utilizando a GPU e os núcleos da CPU.

Este relatório apresenta cada implementação e compara seus desempenhos.

2. Estudo do Modelo de Difusão

A equação de difusão é expressa por:

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

onde:

- C é a concentração do contaminante;
- t é o tempo;
- D é o coeficiente de difusão;
- $\nabla^2 C$ representa a variação espacial da concentração.

Utilizando o método das diferenças finitas, a atualização da concentração em cada célula é aproximada por:

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

3. Configuração do Ambiente e Parâmetros da Simulação

Os parâmetros adotados para todas as implementações foram:

- **Tamanho da grade:** $N \times N$, com $N = 2000$.
- **Coeficiente de difusão:** $D = 0.1$.
- **Passo espacial:** $\Delta x = 1.0$.
- **Passo temporal:** $\Delta t = 0.01$.
- **Número de iterações:** $T = 500$.

A condição inicial determina que todas as células começam com concentração zero, exceto a célula central, que tem valor 1.0. As bordas da grade permanecem inalteradas, representando condições de contorno fixas.

4. Especificações das Máquinas Utilizadas

Os experimentos foram realizados em diferentes plataformas.

4.1. Computadores

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

5. Implementações

A seguir, descrevemos as diferentes abordagens desenvolvidas para a simulação.

5.1. Implementação Sequencial

Esta versão, desenvolvida em C, atualiza os valores de cada célula da matriz de forma linear, sem qualquer paralelismo. Serve como referência para a comparação dos ganhos obtidos com as implementações paralelizadas.

Disponível em: codigo_nao_paralelizado.c.

5.2. Implementação com OpenMP

Utilizando a biblioteca OpenMP, os laços que atualizam a matriz são paralelizados, permitindo a execução das operações em múltiplos núcleos da CPU. A diretiva `#pragma omp parallel for collapse(2)` é empregada para distribuir as iterações entre as threads.

Disponível em: codigo_paralelizado.c.

5.3. Implementação com CUDA

A versão CUDA utiliza a capacidade de processamento massivo das GPUs. Cada thread da GPU é responsável por atualizar uma célula interna da matriz. O código gerencia a transferência de dados entre a memória da CPU (host) e da GPU (device), e executa kernels que aplicam a fórmula das diferenças finitas.

Disponível em: diffusion_cuda.cu.

5.4. Implementação com MPI

Na implementação MPI, o domínio global é dividido entre os processos, onde cada processo trata de um bloco de linhas da matriz. São alocadas linhas extras (linhas halo) para facilitar a troca de informações com os processos vizinhos, utilizando chamadas MPI (como `MPI_Sendrecv`). Essa abordagem permite distribuir a simulação em vários nós de um cluster ou supercomputador.

Disponível em: codigo_mpi.c.

5.5. Implementação Híbrida: MPI + CUDA + OpenMP

Esta abordagem combina:

- **MPI:** Divide a grade global entre os processos, distribuindo a carga entre nós.
- **CUDA:** Em cada processo, a GPU é utilizada para acelerar a atualização dos valores da matriz.
- **OpenMP:** No host, é empregado para acelerar operações auxiliares, como a inicialização e a redução dos resultados.

Essa versão híbrida integra os pontos fortes de cada tecnologia, permitindo a execução eficiente da simulação em ambientes de alto desempenho.

Disponível em: codigo_hibrido.c.

6. Resultados e Discussão

Foram realizados experimentos com todas as implementações, e os resultados foram comparados executando as simulações nas diferentes plataformas (Máquina 1, Máquina 2 e Máquina 3) e, para a implementação híbrida, em um supercomputador.

6.1. Resultados em computadores

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

Tabela 1. Tempos de execução para OpenMP em diferentes máquinas.

Threads	Máquina 1	Máquina 2
1	12.45	34.00
2	14.825	18.20
4	10.56	13.00
8	6.14	12.78
16	5.85	30.83

Tabela 2. Tempos de execução para MPI em diferentes máquinas.

Threads	Máquina 1	Máquina 2
1	12.45	34.00
2	x	23.45
4	x	13.00
8	x	12.03
16	x	5.96

Tabela 3. Tempos de execução para CUDA em diferentes máquinas.

Código	Máquina 2
Sequencial	34.00
OpenMP	8.77
CUDA	12.03
MPI	12.78
Hibrido	7.61

Tabela 4. Comparação geral entre as abordagens.

6.1.1. 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.

Além da tabela, foram gerados gráficos de *speedup* e eficiência que demonstram o ganho de desempenho obtido com cada tecnologia. De forma resumida:

- **Sequencial:** Executa de forma linear, sendo a versão mais lenta.
- **OpenMP:** Distribui as tarefas entre os núcleos da CPU, reduzindo o tempo de execução.

- **CUDA:** Utiliza a GPU para acelerar os cálculos, obtendo ganhos significativos.
- **MPI:** Distribui a simulação entre vários nós, o que é vantajoso em sistemas com muitos recursos.
- **MPI + CUDA + OpenMP:** A abordagem híbrida integra todas as vantagens, proporcionando o melhor desempenho, especialmente em ambientes de supercomputação.

6.2. Análise Final

A comparação final, baseada em testes realizados nas máquinas individuais e no supercomputador, revelou que:

- Todas as versões paralelizadas apresentaram reduções significativas no tempo de execução em comparação à versão sequencial.
- A implementação híbrida (MPI+CUDA+OpenMP) demonstrou o maior ganho de desempenho, aproveitando ao máximo os recursos disponíveis em um ambiente de alto desempenho.
- A escolha da abordagem ideal depende do ambiente computacional disponível e da escala do problema a ser resolvido.

7. Conclusão

Este relatório apresentou a simulação da equação de difusão utilizando o método das diferenças finitas, com várias implementações: sequencial, OpenMP, CUDA, MPI e híbrida (MPI+CUDA+OpenMP). Cada abordagem foi explicada de forma simples para facilitar o entendimento, mesmo por leitores sem experiência prévia em computação paralela.

Os resultados demonstraram que o uso do paralelismo (seja em nível de CPU, GPU ou distribuído) permite reduzir drasticamente o tempo de execução da simulação. Em especial, a implementação híbrida mostrou-se a mais eficiente, sendo a melhor opção para ambientes com recursos avançados, como supercomputadores.

8. Referências

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*. Disponível em: `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf`

[4] NVIDIA Corporation. (2020). *CUDA C Programming Guide*. Disponível em: `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`

[5] MPI Forum. (2015). *MPI: A Message-Passing Interface Standard*. Disponível em: `https://www.mpi-forum.org/docs/`