Otimização na Simulação e Análise de Modelos de Difusão de Contaminantes em Água

Pedro Paulo F. M. Vianna, Victor Jorge C. Chaves, João Guilherme Iwo D. L. Costa

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

Abstract. *This work aims to create a simulation that models the diffusion of contaminants in a body of water (such as a lake or river), applying concepts of parallelism to speed up the calculation and observe the behavior of pollutants over time. The project will investigate the impact of OpenMP, CUDA and MPI on model runtime and accuracy. In this document we will address the implementation in OpenMP, parallelizing the code provided in the project proposal.*

Resumo. *Este trabalho tem como objetivo criar uma simulação que modele a difusão de contaminantes em um corpo d'água (como um lago ou rio), aplicando conceitos de paralelismo para acelerar o cálculo e observar o comportamento de poluentes ao longo do tempo. O projeto investigará o impacto de OpenMP, CUDA e MPI no tempo de execução e na precisão do modelo.*

1. Introdução

1.1. Motivação

A difusão de contaminantes em corpos d'água, como lagos, rios e oceanos, é um tema de grande relevância ambiental e social, dada a crescente preocupação com a qualidade da água e o impacto de atividades humanas no meio ambiente. A contaminação por substâncias químicas, resíduos industriais, efluentes domésticos e outros poluentes pode comprometer ecossistemas aquáticos e a saúde pública. Assim, entender e prever a propagação desses contaminantes é fundamental para a formulação de políticas de controle e mitigação.

A partir de simulações computacionais pode ser feito o estudo no comportamento de difusão de poluentes na água. As simulações permitem a análise de diferentes cenários e variáveis, oferecendo uma visão detalhada sobre o comportamento dos poluentes ao longo do tempo e em diferentes condições ambientais. Essas ferramentas proporcionam aos pesquisadores a capacidade de modelar e prever eventos de contaminação.

No entanto, pode ser alta o custo computacional ao simular processos de difusão em grande escala. Para superar esses desafios, métodos de otimização do algoritmo são necessárias para agilizar o tempo de execução das simulações, como também ser capaz de computar casos mais demandantes. Tecnologias como OpenMP (Open Multi-Processing), CUDA (Compute Unified Device Architecture) e MPI (Message Passing Interface) oferecem soluções poderosas para acelerar os cálculos, permitindo o processamento simultâneo de múltiplas operações e a distribuição de tarefas entre diferentes unidades de processamento.

Este estudo se concentra na implementação e comparação dessas técnicas de otimização em simulações de difusão de contaminantes, avaliando seu impacto no desempenho e na

precisão dos modelos. Através da utilização de OpenMP, CUDA e MPI, será buscado a redução do tempo de execução de um algoritmo existente de simulação do processo de difusão em corpos d'água.

1.2. Modelo de Difusão Utilizado

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

Onde:

- C é a concentração do contaminante,
- t é o tempo,
- D é o coeficiente de difusão, e
- $(\nabla)^2 C$ representa a taxa de variação de concentração no espaço.

Este modelo descreve a difusão de um contaminante em um meio homogêneo e isotrópico. A equação baseia-se na Segunda Lei de Fick, que afirma que a taxa de variação temporal da concentração em um ponto é proporcional à divergência do fluxo de difusão nesse ponto.

Para resolver esta equação, utiliza-se o método de diferenças finitas, discretizando o espaço e o tempo. A discretização leva à equação de diferença:

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

Onde:

- $C_{i,j}^{n}$ é a concentração no ponto (i, j) no tempo n,
- Δt é o passo temporal, e
- Δx é o tamanho da célula no espaço discreto.

Este esquema permite calcular a evolução temporal da concentração em uma grade 2D, sendo adequado para implementação em sistemas paralelos, como o modelo estudado neste trabalho.

2. Objetivo

O objetivo deste estudo é desenvolver uma simulação eficiente para modelar a difusão de contaminantes em corpos d'água, como lagos e rios, utilizando técnicas avançadas de paralelismo e otimização de código. A simulação visa não apenas prever o comportamento dos poluentes ao longo do tempo, mas também avaliar o desempenho computacional de diferentes abordagens de paralelização.

As principais metas incluem:

- Implementar e comparar técnicas de paralelismo utilizando OpenMP, CUDA e MPI para acelerar a simulação.
- Avaliar o impacto da otimização -O3 do GCC no desempenho da simulação.
- Analisar a eficiência e a escalabilidade de cada técnica em diferentes cenários de simulação, variando o tamanho da matriz e o número de iterações.

- Fornecer insights sobre a melhor abordagem para otimizar a execução de simulações de difusão de contaminantes em ambientes computacionais diversos.

Ao alcançar esses objetivos, o estudo pretende contribuir para a compreensão dos benefícios e limitações das diferentes técnicas de paralelismo e otimização, assim destacando qual método demonstrou a melhor performance para o problema proposto.

3. Metodologia

Para otimizar a simulação da difusão de contaminantes em corpos d'água, técnicas de paralelismo e otimização de código foram aplicadas. Nesta seção, são descritas as técnologias que implementam e facilitam a programação paralela, sendo elas: CUDA, OpenMP, MPI e a otimização -O3 do GCC, com o objetivo de melhorar o desempenho computacional da simulação.

3.1. CUDA (Compute Unified Device Architecture)

CUDA é uma plataforma de computação paralela desenvolvida pela NVIDIA, que permite o uso de GPUs (Graphics Processing Units) para realizar cálculos computacionais intensivos. Ao distribuir as operações de cálculo em milhares de núcleos da GPU, CUDA possibilita uma aceleração significativa em tarefas que envolvem grandes volumes de dados ou computações repetitivas, como na simulação de difusão de contaminantes.

3.2. OpenMP (Open Multi-Processing)

OpenMP é uma API que suporta a programação paralela em ambientes compartilhados de memória. Com uma sintaxe baseada em diretivas, OpenMP facilita a paralelização de loops e outras estruturas de controle em linguagens como C, C++ e Fortran. A utilização de OpenMP nesta simulação permite a execução paralela em múltiplos threads, aproveitando melhor os recursos de CPUs multicore.

3.3. MPI (Message Passing Interface)

MPI é uma biblioteca padrão para programação paralela em sistemas de memória distribuída. Ao contrário do OpenMP, que opera em memória compartilhada, MPI permite a comunicação entre diferentes processos que podem estar executando em máquinas distintas. Na simulação de difusão de contaminantes, o MPI é utilizado para distribuir partes do domínio de cálculo entre múltiplos processos em uma mesma máquina, sincronizando e agregando os resultados.

3.4. Otimização -O3 do GCC

A flag -O3 do compilador GCC (GNU Compiler Collection) ativa uma série de otimizações agressivas de código que visam maximizar o desempenho de programas compilados.

4. Resultados e Discussão

Nesta seção, são apresentados os resultados obtidos para cada técnica de otimização aplicada, assim com os resultados do algoritmo sem otimização, será feito uma análise de speedup para cada técnica.

Para cada técnica, foi conduzida uma série de simulações utilizando combinações variadas de tamanhos de matriz e quantidades de iterações, conforme descrito a seguir:

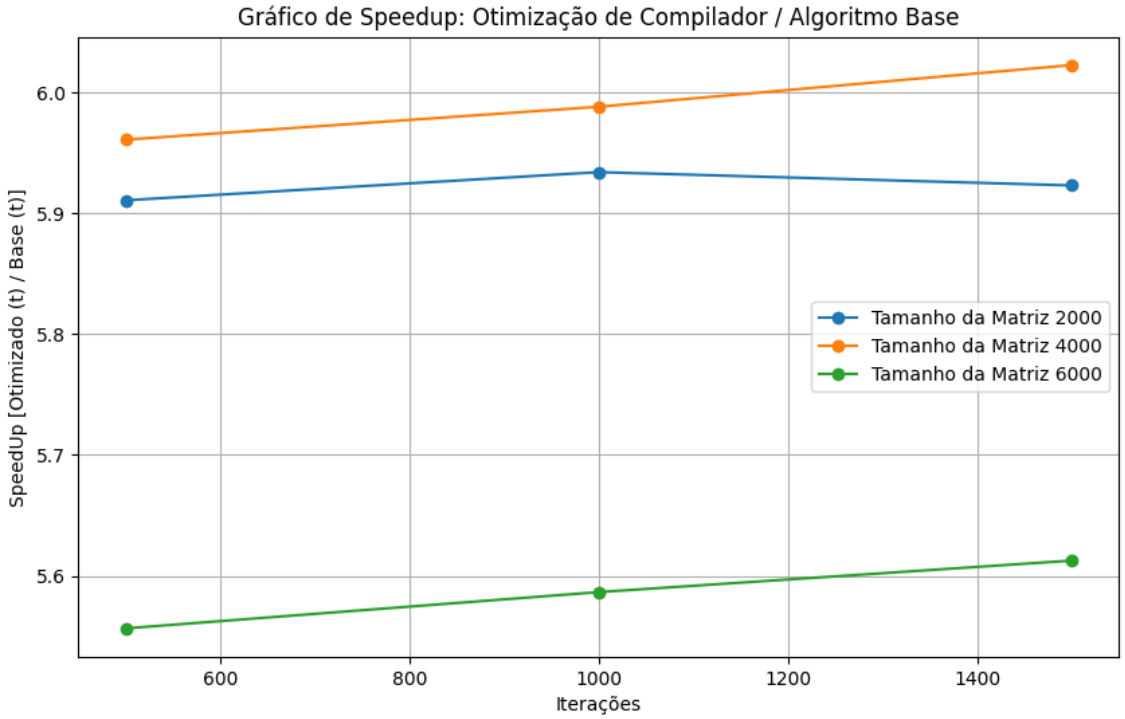
- **Tamanho da Matriz:** 2000, 4000, 6000
- **Quantidade de Iterações:** 500, 1000, 1500

Essas combinações foram escolhidas para simular diferentes cenários de complexidade computacional, permitindo uma análise detalhada do desempenho de cada técnica em condições variadas. Os resultados focam nas métricas de tempo de execução da simulação.

O ambiente de execução de todos os algoritmos (exceto do algoritmo em CUDA) tem as seguintes características:

- Máquina: Notebook IdeaPad Lenovo
- CPU: Ryzen 5 5600G
- Memória RAM: 12 GB
- Compilador: GCC

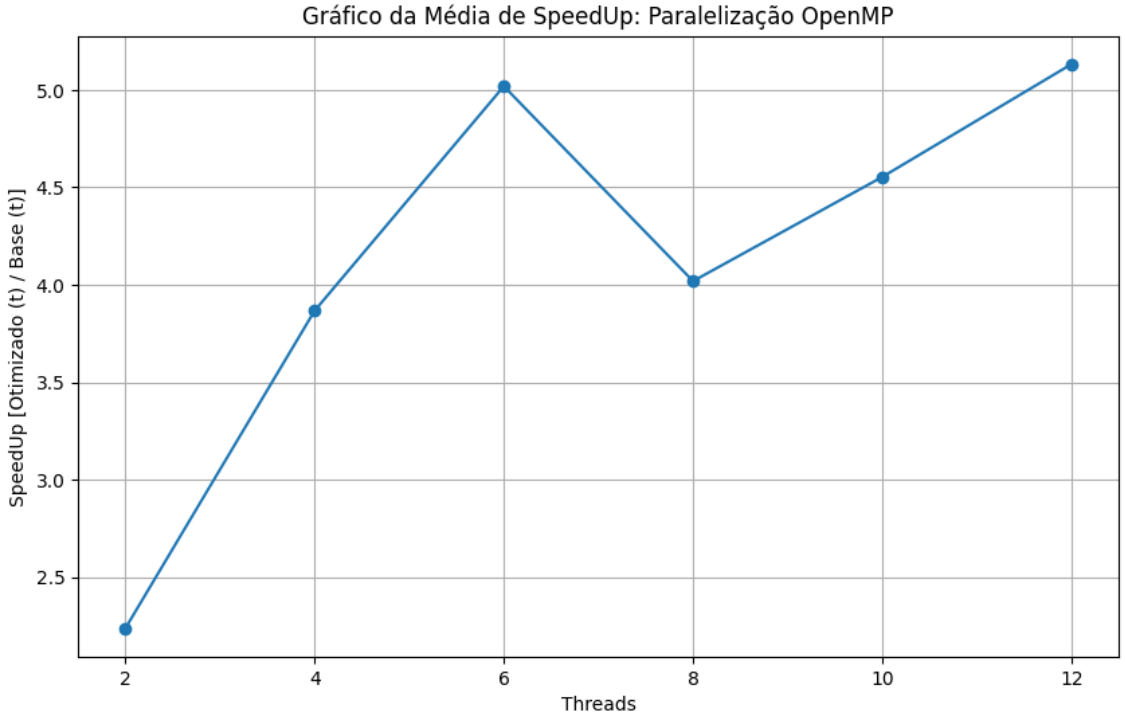
4.1. Resultados do Algoritmo Compilado com a Flag -O3



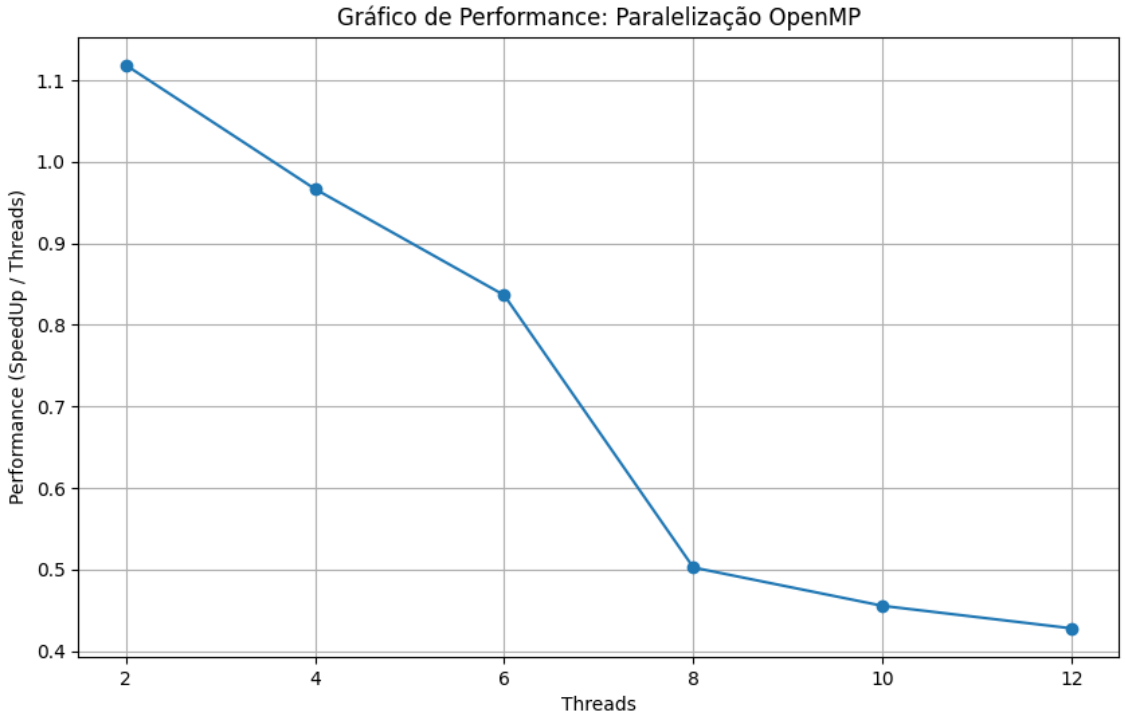
4.2. Resultados do Algoritmo Paralelizado com OpenMP

No gráfico abaixo, sobre a média de SpeedUp, os dois casos com o melhor SpeedUp é com 12 e 6 threads, invés de ser 12 e 10 threads, apesar de não ser encontrado um

motivo para isso, pode ser que na configuração de 6 processos tem o menor overhead em comparação ao tempo ganho com a paralelização.

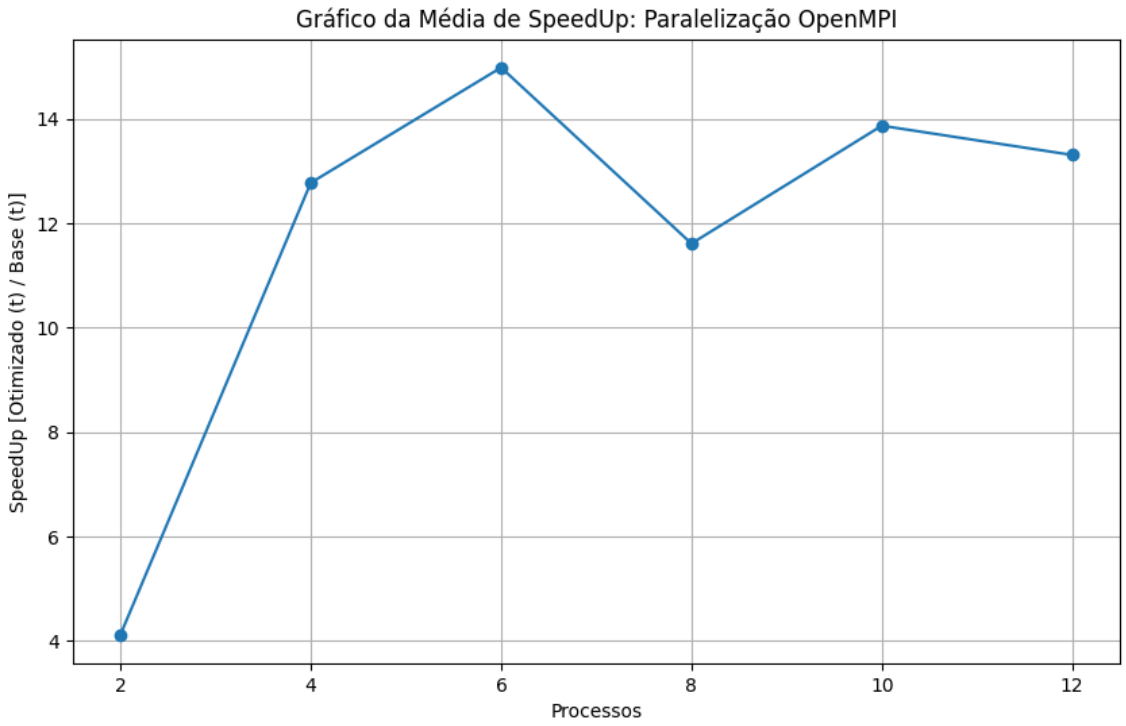


No gráfico abaixo, sobre a performance, é relevante a queda de performance conforme mais threads são usadas, indicando assim que quanto mais threads menos eficiente é o uso das CPUs.

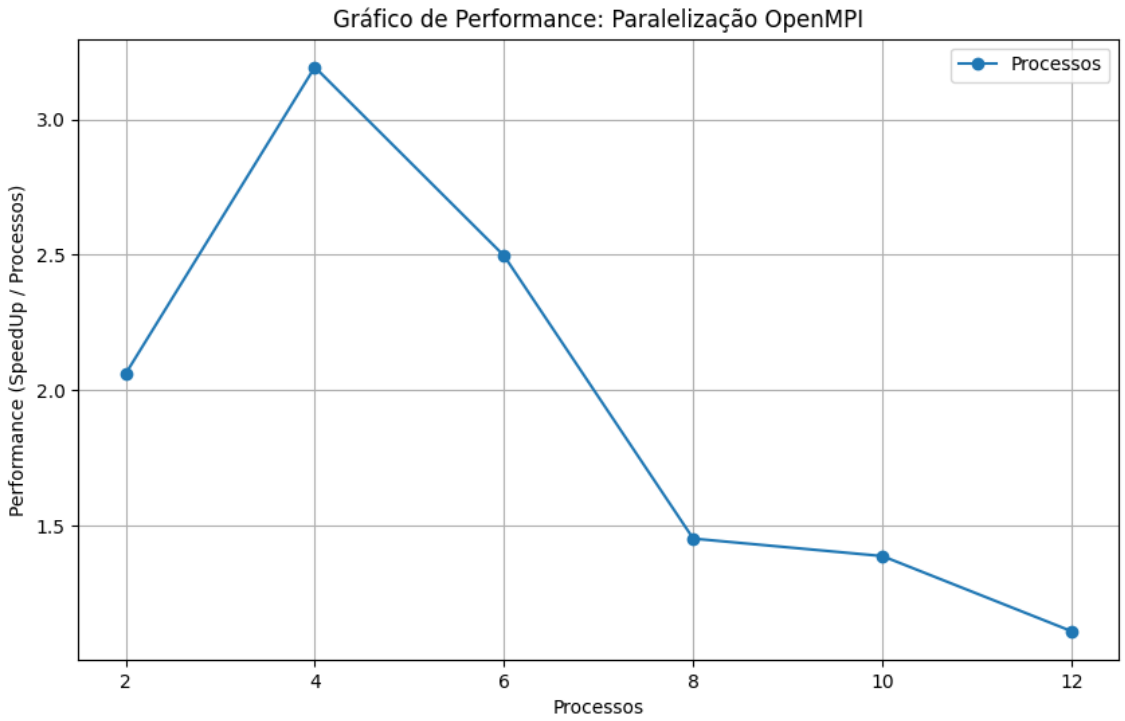


4.3. Resultados do Algoritmo Paralelizado com OpenMPI e com Otimização de Compilação

No gráfico abaixo, sobre a média de SpeedUp, os dois casos com o melhor SpeedUp é com 6 (a melhor) e 10 processos, invés de ser 12 e 10 processos, isso pode ocorrer por diminuir a quantidade de sincronização necessária entre processos enquanto há uma divisão adequada de trabalho entre os processos.



No gráfico abaixo, sobre a performance, é relevante notar que em comparação a paralelização com OpenOMP, a performance com MPI sempre está acima que 1, enquanto do OpenOMP esteve em sua maioria, abaixo de 1, assim indicando uma ótima utilização dos recursos do computador quando a atividade é dividida entre processos que em threads.

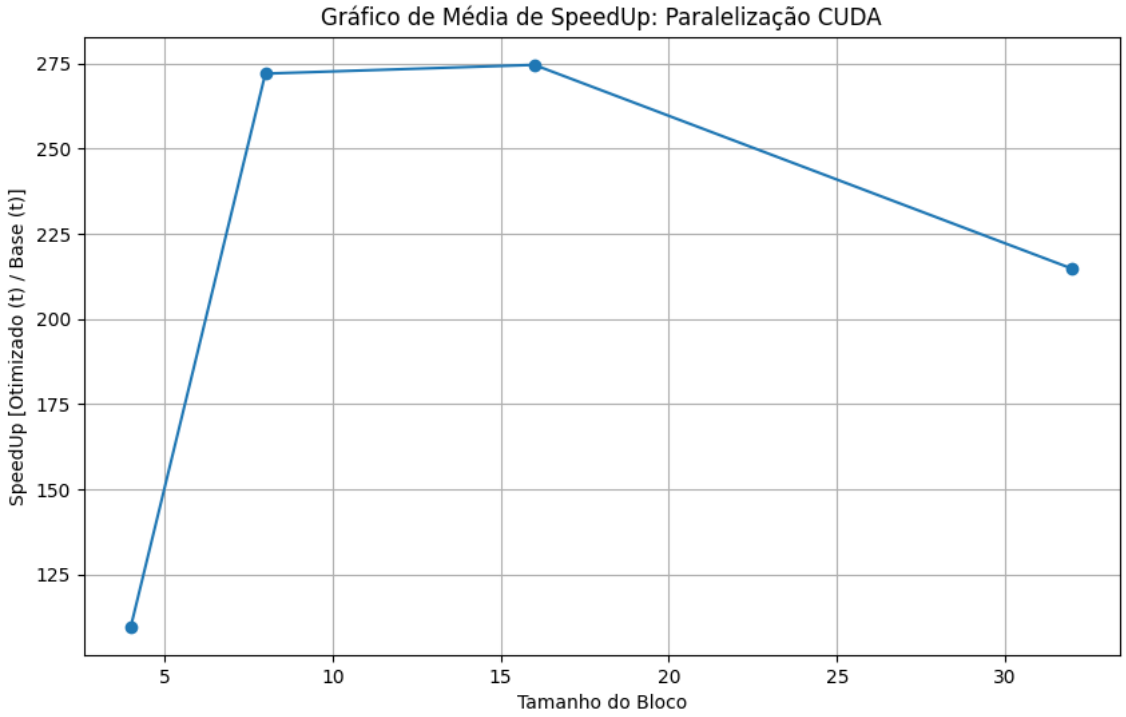


4.4. Resultados do Algoritmo Adaptado para CUDA

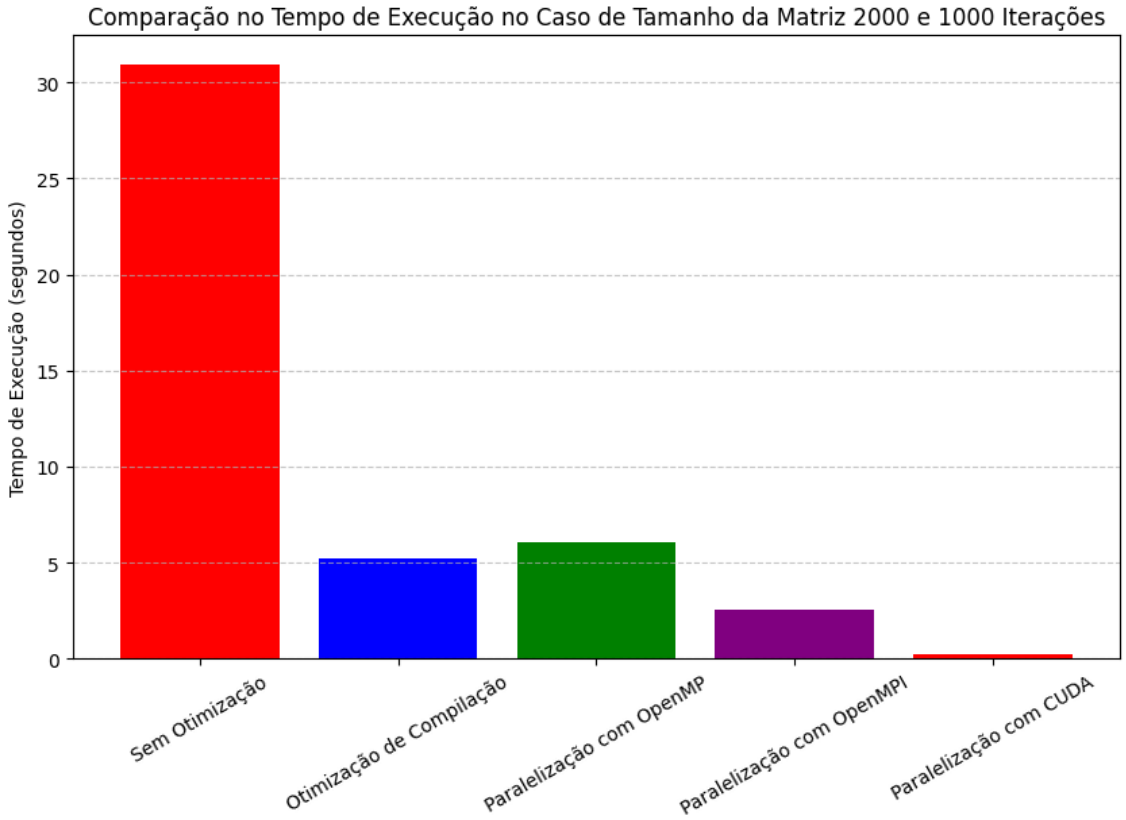
4.4.1. Configuração do Ambiente de Execução

- Máquina: Ambiente Virtual do Google Colab
- CPU:
- Memória RAM: 16 GB
- GPU: T4
- Compilador: NVCC (NVIDIA CUDA Compiler)

No gráfico abaixo de SpeedUp, comparando cada caso com um tamanho de bloco diferente, é notável que com 16 de tamanho de bloco é o que apresentou o melhor SpeedUp. Esse comportamento pode ser explicado pelo balanceamento entre a ocupação da GPU e a minimização da sobrecarga associada à troca de contexto entre warps. Blocos muito pequenos podem não explorar eficientemente a capacidade dos multiprocessadores da GPU, enquanto blocos muito grandes podem causar contenção de recursos e maior latência na comunicação entre threads.



4.5. Comparativo com todos os Métodos



Os resultados obtidos evidenciam que a implementação CUDA apresentou o melhor desempenho, mas houve uma falha na obtenção de mais dados, o que limitou a análise mais aprofundada da escalabilidade da solução.

Entre as otimizações com CPU, OpenMPI apresentou o melhor resultado utilizando de 6 processos. E curiosamente, a versão sequencial otimizada com a flag -O3 do GCC superou a implementação OpenMP com 12 threads. Isso sugere que a otimização agressiva aplicada pelo compilador, foi suficiente para melhorar o desempenho sem a necessidade do overhead de gerenciamento de múltiplas threads, indicando que problemas de itensivo iteração pode ser otimizados em compilação.

5. Conclusão

Os resultados indicam que a utilização de GPUs com CUDA se mostra particularmente eficiente para problemas do tipo stencil 2D, alcançando uma execução 275 vezes mais rápido que o algoritmo original. O processamento de células pode ser altamente paralelizado devido à sua natureza estruturada e ao alto grau de reutilização de dados vizinhos. Essa característica permite um melhor aproveitamento da hierarquia de memória da GPU, reduzindo a latência e aumentando a largura de banda efetiva. Trabalhos futuros podem explorar diferentes estratégias de acesso à memória global e compartilhada para melhorar ainda mais a eficiência da computação stencil em arquiteturas CUDA.

Outra alternativa utilizando apenas de CPU é a execução em processos via OpenMPI aproveitando de uma otimização de compilação, com esse conjunto de técnicas é possível reduzir ainda mais o tempo de execução, alcançando um SpeedUp de 15, mas comparando com a execução de GPU, é 18 vezes mais lenta.

6. References

NVIDIA Corporation. *CUDA C Programming Guide*. Santa Clara, CA: NVIDIA, 2024. Disponível em: `https://docs.nvidia.com/cuda/cuda-c-programming-guide/`. Acesso em: 10 fev. 2025.

Carleton University. *Introduction to MPI*. Disponível em: `https://carleton.ca/rcs/rcdc/introduction-to-mpi/`. Acesso em: 10 fev. 2025.

Calvin University. *Matrices*. Disponível em: `https://cs.calvin.edu/courses/cs/112/labs/10/matrices/`. Acesso em: 10 fev. 2025.