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.

Uma abordagem eficaz para estudar a difusão de contaminantes é através de simulações computacionais. 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, auxiliando na tomada de decisões mais informadas e eficazes.

No entanto, a simulação de processos de difusão em grande escala pode ser computacionalmente intensiva, especialmente quando se considera a necessidade de alta precisão e a complexidade dos modelos envolvidos. Para superar esses desafios, a otimização do código e a utilização de técnicas de paralelismo se tornam essenciais. 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, oferecendo uma base sólida para futuros trabalhos na área de simulação ambiental e modelagem computacional.

3. Metodologia

Para otimizar a simulação da difusão de contaminantes em corpos d'água, diversas técnicas de paralelismo e otimização de código foram empregadas. Nesta seção, são descritas as metodologias relacionadas ao uso de 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. A implementação em CUDA envolve a escrita de kernels, que são funções executadas em paralelo pela GPU, e a gestão eficiente da memória entre a CPU e a GPU.

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. A abordagem é especialmente útil para dividir o trabalho computacional entre os núcleos do processador, reduzindo o tempo total de execução.

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 nós de um cluster de computadores, sincronizando e agregando os resultados. Isso permite o escalonamento eficiente da simulação para conjuntos de dados muito grandes.

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. Entre as otimizações realizadas estão a unrolling de loops, a substituição de expressões comuns, a eliminação de código redundante, e a inlining de funções. Essas otimizações são aplicadas automaticamente pelo GCC durante a compilação do código, reduzindo o tempo de execução e melhorando a eficiência do programa sem a necessidade de modificações manuais no código-fonte.

4. Desenvolvimento e Resultados

Nesta seção, são apresentados os resultados obtidos para cada técnica de otimização aplicada, assim como os resultados do algoritmo sem otimização. A análise considera diferentes configurações de simulações, avaliando o impacto de cada técnica no desempenho computacional e na eficiência do modelo.

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:** 400, 800, 1200, 1600, 2000
- **Quantidade de Iterações:** 200, 400, 600, 800, 1000

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.

4.1. Execução do Algoritmo Sem Otimização

4.1.1. Configuração do Ambiente de Execução

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

4.2. Execução do Algoritmo Compilado com a Flag -O3

4.2.1. Configuração do Ambiente de Execução

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

4.3. Execução do Algoritmo Paralelizado com OpenMP

4.3.1. Configuração do Ambiente de Execução

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

4.4. Execução 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)

4.5. Execução do Algoritmo Distribuído com MPI

4.5.1. Configuração do Ambiente de Execução

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

Os dados usados nesta análise representam os tempos de execução e os resultados de concentração para uma simulação paralelizada de difusão de contaminantes na água. Os parâmetros variáveis incluem o número de threads, o tamanho da grade e as iterações do algoritmo. Os principais objetivos da coleta de dados foram avaliar:

- A precisão da simulação comparando valores de concentração entre as configurações.
- A escalabilidade do desempenho do código medindo o tempo de execução com o aumento do número de threads.
- A eficiência da estratégia de paralelização.

Os dados foram coletados sistematicamente executando simulações sob condições controladas e garantindo consistência entre as configurações.

5. Análise Estatística

A análise estatística foca em entender a relação entre o tempo de execução(podem ser vistas das figuras **??** a **??**) e as variáveis: threads, tamanho da grade e iterações.

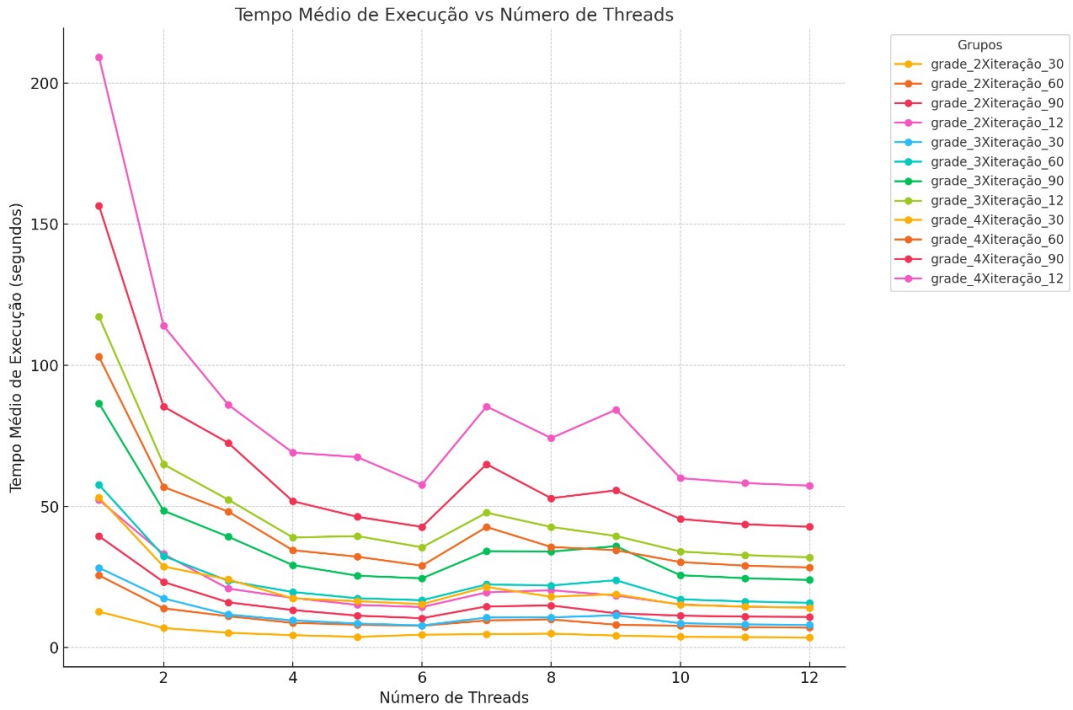

Figure 1. Tempo médio de execução para cada grupo de grade e iterações com base no número de threads.(OpenMP)

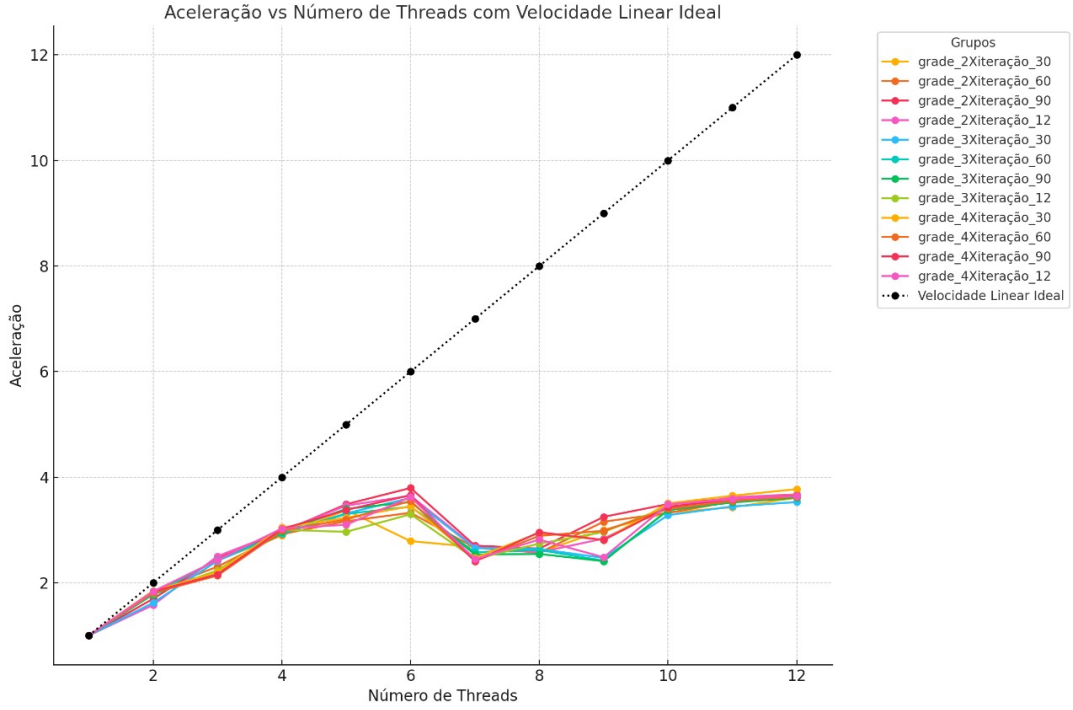

Figure 2. Speedup médio para cada combinação de grupos e threads para avaliação na seção(OpenMP) ??.

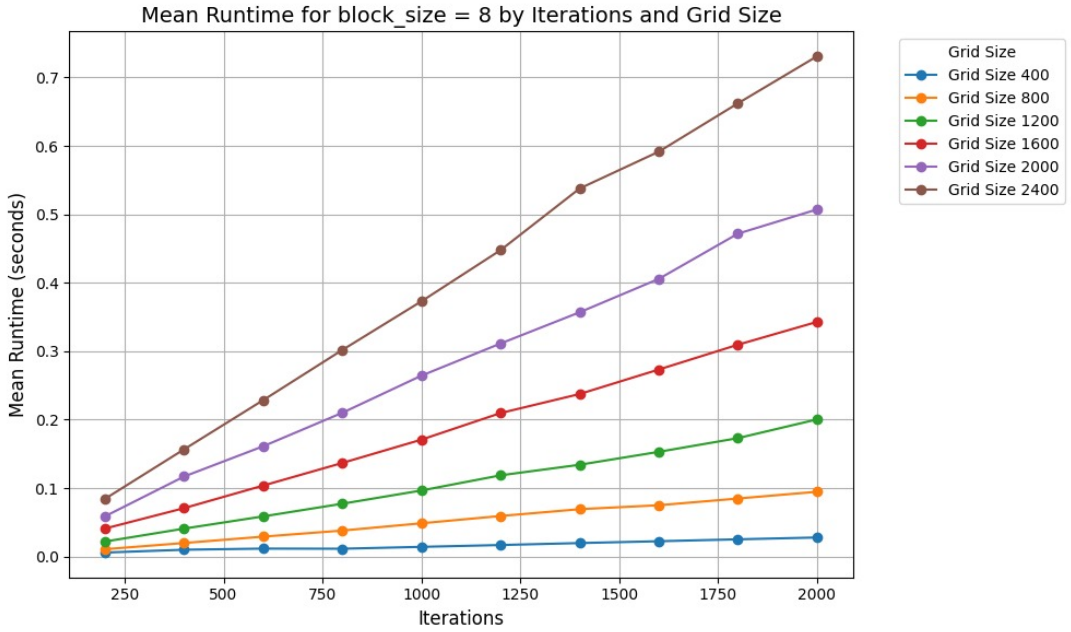

Figure 3. Tempo de execução por iterações relacionado ao tamanho da grade.(CUDA)

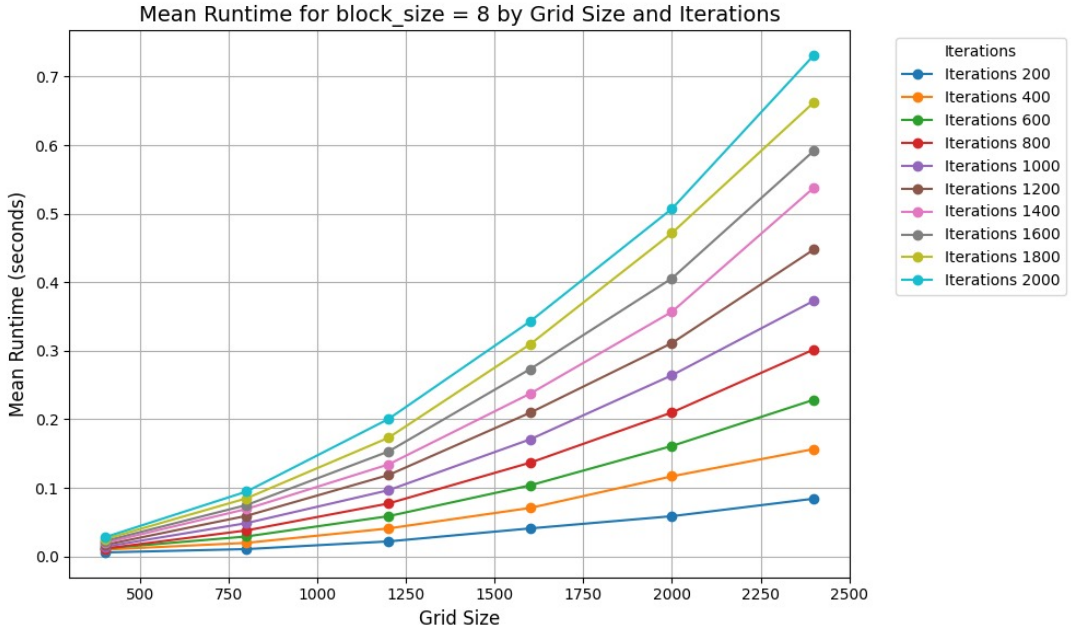

Figure 4. Tempo de execução pelo tamanho da grade relacioando a iterações.(CUDA)

As principais conclusões foram:

- **Threads:** O aumento no número de threads reduz o tempo de execução até certo ponto, com retornos decrescentes devido ao overhead, no caso da CUDA, o crescimento é exponencial, conforme o esperado.

- **Tamanho da Grade:** Grades maiores aumentam consistentemente o tempo de execução
- **Interações:** Interações significativas foram identificadas, como threads mitigando o impacto do tamanho da grade e iterações no tempo. Em CUDA obteve-se um resultado positivo, o crescimento linear em relação ao aumento de iterações.

5.1. Analise de Regressão

5.1.1. OpenMP

Para compreender o impacto das variáveis no tempo de execução, foi desenvolvido um modelo de regressão linear múltipla. O modelo inclui como preditores o número de threads, tamanho da grade, iterações e suas interações. Os resultados da análise de regressão estão resumidos abaixo:

- **R-squared:** 0,775 – O modelo explica 77,5% da variação no tempo de execução.
- **R-squared Ajustado:** 0,765 – Ajustado para o número de preditores.
- **Preditores Significativos:**
  - **Threads:** Efeito positivo no tempo de execução ($\beta = 5.6200, p < 0.001$).
  - **Tamanho da Grade:** Efeito positivo no tempo de execução ($\beta = 0.0122, p = 0.008$).
  - **Threads ×Tamanho da Grade:** Interação negativa ($\beta = -0.0019, p < 0.001$).
  - **Threads ×Iterações:** Interação negativa ($\beta = -0.0040, p < 0.001$).
  - **Tamanho da Grade ×Iterações:** Interação positiva ($\beta = 2.653 \times 10^{-5}, p < 0.001$).
- **Preditores Não Significativos:**
  - Iterações ($\beta = -0.0102, p = 0.504$) – Esta variável não impacta significativamente o tempo de execução isoladamente.
- **Multicolinearidade:** Um número de condição alto (3.2×10^7) indica possíveis problemas de multicolinearidade entre os preditores.

Os resultados indicam que o tempo de execução é significativamente influenciado pelo número de threads, tamanho da grade e as interações entre essas variáveis. Contudo, problemas de multicolinearidade sugerem a necessidade de refinamento do modelo.

6. Avaliação de Perfórmance

6.0.1. OpenMP

A avaliação de desempenho foi realizada calculando o speedup e comparando os resultados reais com o speedup linear ideal. A análise mostrou:

- **Speedup:** O desempenho melhora com mais threads, mas o speedup observado se desvia do caso linear ideal, especialmente com maior número de threads.
- **Eficiência:** A eficiência diminui com o aumento do número de threads, indicando possível overhead ou contenção de recursos.
- **Escalabilidade:** Grades menores e menos iterações mostram escalabilidade limitada, enquanto configurações maiores se beneficiam mais da paralelização.

6.0.2. CUDA

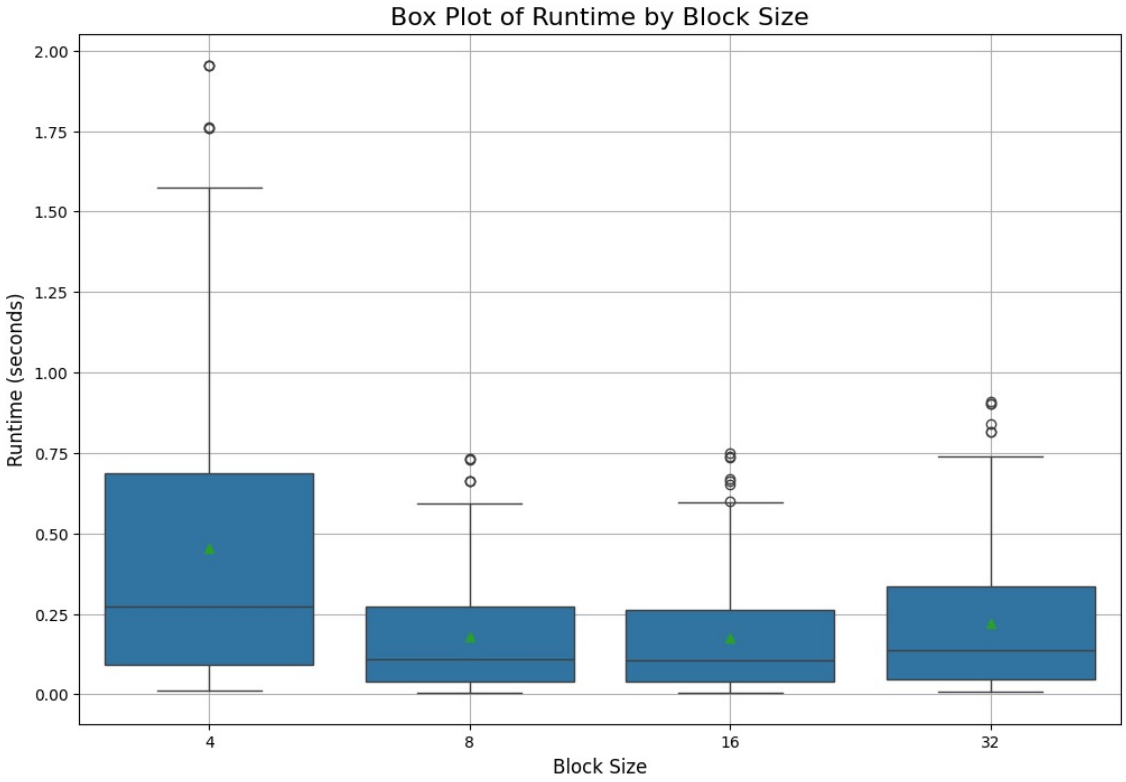

Figure 5. Tempo de processamento para cada tamanho do bloco usado.

Podemos observar que CUDA obteve seus melhores tempos com blocos de tamanho de 8 e 16, mesmo não sendo feito nenhum calculo preciso para a otimização do bloco em relação a grade.

Comparativamente ao OpenMP, a **Escabilidade** teve uma melhora imensa, ao desenhar a linha do aumento do tamanho da grade visualizamos um crescimento exponecial, que já é o esperado. Fazendo o contrário, chega-se no crescimento linear do aumento das iterações, forte indício que a paralelização obteve impacto no problema.

A **eficiência** foi mais controlada, sem estouros do tempo de execução por conta de um aumento dos blocos ou iterações. O pior caso para T4, com grade 2000x2000 e 1000 iterações foi aproximadamente 0.25 segundos, nos melhores casos, 0.24, sem distonâncias de valor de tempo.

7. Resultados Visuais e Tabulares

A análise de dados é apoiada por diversas visualizações:

- Gráficos de linhas mostrando o tempo de execução para cada grupo de tamanho de grade e iterações. Um Boxplot para visualisação do melhor tempo de CUDA.
- Gráficos de speedup comparando o desempenho real com o speedup linear ideal. O comparativo dos gráficos reais de CUDA com o esperado para análise comparativa com OpenMP.

- Resumos do modelo de regressão ilustrando os efeitos das variáveis e suas interações no tempo de execução.

Os resultados em formato tabular incluem estatísticas resumidas e coeficientes do modelo de regressão, fornecendo insights detalhados sobre as tendências observadas.

8. Conclusão

Nesse primeiro momento foi feito apenas uma tradução do código sequencial para uma implementação em OpenMP com pouco foco para otimização do algoritmo, visando avaliar exclusivamente o impacto de usa-lo para paralelização. Nesse sentido temos uma melhora, mas abaixo do ideal teórico esperado.

Para CUDA, se manteve o cuidado de traduzir do código sequencial, então pode-se inferir uma equivalência entre as implementações. Ao comparar ambas, observa-se que a implementação ”simples” de CUDA atinge um desempenho muito melhor. Além disso a escabilidade em CUDA se comportou da forma esperada com ganho significativos para aumentos no tamanho da *grid*.