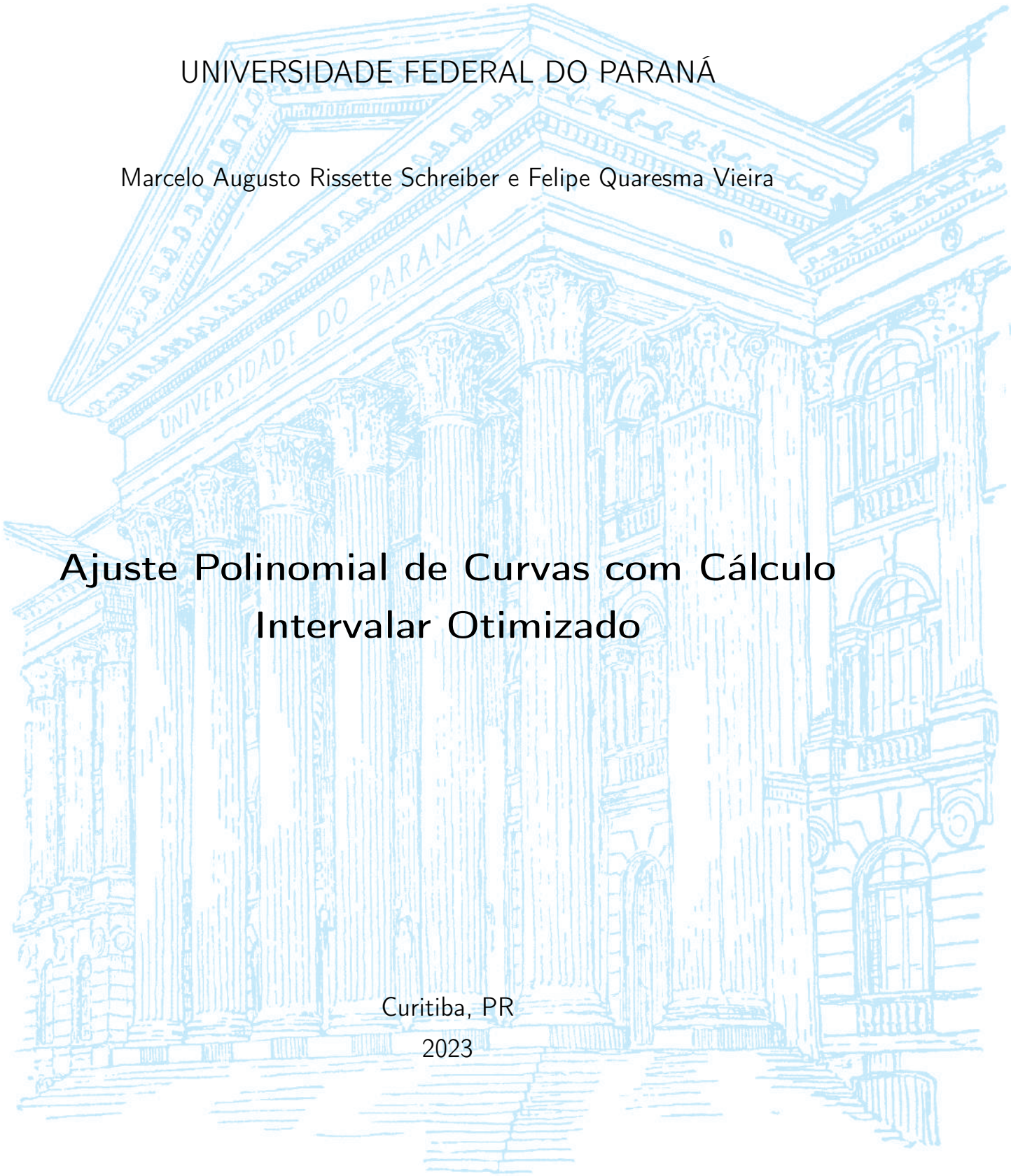
UNIVERSIDADE FEDERAL DO PARANÁ

Marcelo Augusto Rissette Schreiber e Felipe Quaresma Vieira



Ajuste Polinomial de Curvas com Cálculo Intervalar Otimizado

Curitiba, PR

2023

Marcelo Augusto Rissette Schreiber e Felipe Quaresma Vieira

Ajuste Polinomial de Curvas com Cálculo Intervalar Otimizado

Trabalho acadêmico para a disciplina CI1164 - Introdução à computação científica.

Universidade Federal do Paraná (UFPR)
Departamento de Informática

Curitiba, PR
2023

RESUMO

Este trabalho visa aprimorar e avaliar o desempenho do Ajuste Polinomial com Cálculo Intervalar desenvolvido anteriormente. Foram implementadas diversas otimizações, como o uso de vetores unidimensionais, reutilização de diagonais da matriz, aproveitamento de cálculos exponenciais, entre outros. Os testes de desempenho foram realizados com o GCC e as opções especificadas, utilizando Likwid para análises detalhadas. Os resultados são apresentados em gráficos para diferentes parâmetros, demonstrando melhorias significativas em relação à versão anterior.

Palavras-chave: Ajuste Polinomial, Cálculo Intervalar, Otimização, Desempenho, Likwid.

ABSTRACT

This work aims to improve and evaluate the performance of the Polynomial Fitting with Interval Calculation developed in a previous study. Various optimizations were implemented, such as the use of one-dimensional vectors, reuse of matrix diagonals, exploitation of exponential calculations, among others. Performance tests were conducted with GCC and the specified options, using Likwid for detailed analyses. The results are presented in graphs for different parameters, demonstrating significant improvements compared to the previous version.

Keywords: Polynomial Fitting, Interval Calculation, Optimization, Performance, Likwid.

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo, apresentaremos uma visão geral do trabalho, descrevendo os capítulos e as seções

No Segundo capítulo, abordaremos as otimizações realizadas, especificadas nas seções 2.1, vetores unidimensionais, 2.2, aproveitamento de cálculos exponenciais, 2.3, reaproveitamento de cálculo de diagonais.

No terceiro capítulo, falaremos sobre análise de desempenho. Detalhada nas seções 3.1, criação da matriz, 3.2, cálculo do resíduo e 3.3 resolução do sistema.

2 OTIMIZAÇÕES REALIZADAS

2.1 VETORES UNIDIMENSIONAIS

Uma das principais otimizações realizadas foi a transição de matrizes bidimensionais para vetores unidimensionais. Essa alteração visa reduzir o consumo de memória e otimizar o acesso aos elementos da matriz. Utilizar vetores unidimensionais elimina a necessidade de alocação de memória para a segunda dimensão e reaproveita as linhas de cache que são puxadas ao acessar algum elemento da matriz, proporcionando uma gestão mais eficiente dos recursos.

2.2 APROVEITAMENTO DE CÁLCULOS EXPONENCIAIS

Outra otimização crucial foi o aprimoramento do aproveitamento de cálculos exponenciais durante a geração da matriz. Em vez de calcular repetidamente as potências de x para cada elemento da matriz, implementamos uma abordagem que reutiliza o resultado anterior. Isso é particularmente eficaz em termos de operações aritméticas, pois evitamos o custo computacional associado a múltiplas chamadas da função `interval_pow`. A utilização de um termo previamente calculado para derivar termos subsequentes reduz o tempo de execução e melhora a eficiência do algoritmo.

2.3 REAPROVEITAMENTO DE CÁLCULO DE DIAGONAIS

A reutilização de diagonais da matriz durante o cálculo do ajuste polinomial é outra estratégia adotada. Ao reutilizar valores previamente calculados, evitamos redundâncias e reduzimos a carga computacional, pois as diagonais (da direita pra esquerda) contém o mesmo valor de coeficientes.

3 METODOLOGIA

3.1 ESPECIFICAÇÕES

- CPU: AMD Ryzen 5 3500U @ 2.100GHz
- Sistema Operacional: Linux Pop!_OS 22.04 LTS x86_64
- Memória RAM: 5861MiB
- Compilador: GCC 11.4.0
- Tamanho das entradas: Variando de 64 a 100 milhões.
- Os grupos do likwid utilizados foram FLOPS_DP CACHE e MEM com os valores, respectivamente, de RETIRED_SSE_AVX_FLOPS_DOUBLE_ALL, data cache miss ratio e Memory Bandwidth (FLOPS_DP AVX é indisponível em processadores AMD).
- Mais especificações no arquivo **likwid-topology.txt**

4 ANÁLISE DE DESEMPENHO

4.1 CRIAÇÃO DA MATRIZ

4.1.1 MFLOP

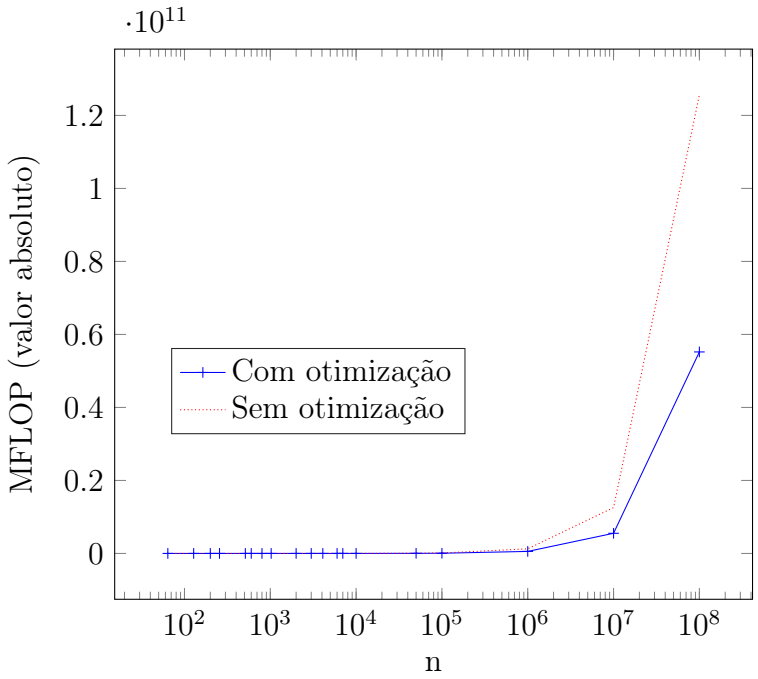


Figura 1 – Gráfico com a relação de MFLOP por tamanho da entrada na geração da matriz.

No contexto do Gráfico 1, a estratégia de copiar a diagonal em vez de calcular cada valor individual apresentou uma redução significativa na carga computacional. Na maior entrada, a quantidade de MFLOPS necessários diminuiu de 125.6 bilhões para 55.2 bilhões, destacando uma eficiência notável ao otimizar o processo por meio da exploração da simetria intrínseca da matriz no contexto do método dos mínimos quadrados.

4.1.2 TAXA DE CACHE MISS NA L2

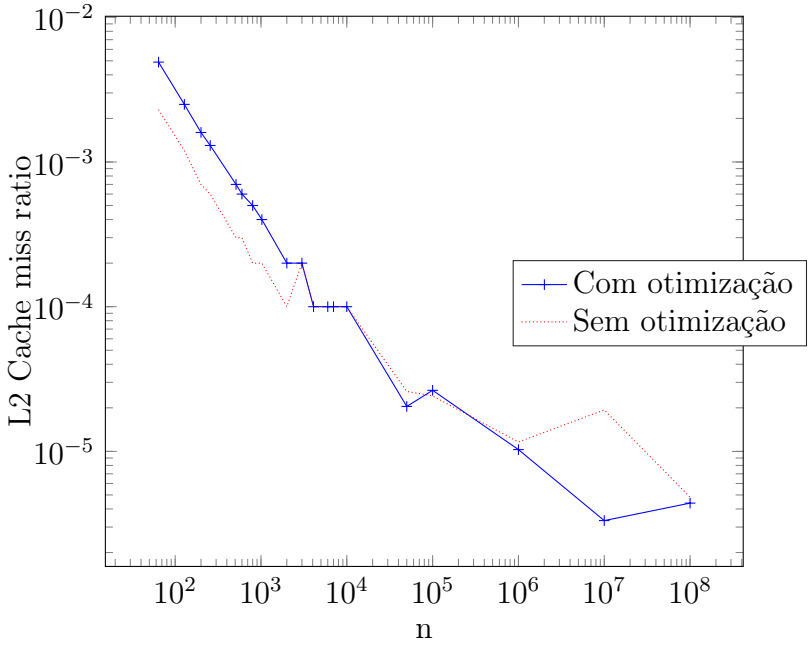


Figura 2 – Gráfico com a relação de l2 cache miss ratio por tamanho da entrada na geração da matriz.

No gráfico de cache miss da L2 (Figura 2), a transição da matriz 2D para um vetor unidimensional com o quadrado do tamanho resultou em melhoria perceptível, especialmente em tamanhos maiores. No entanto, a redução nos cache misses foi menos substancial, principalmente em tamanhos menores. A mudança indicou uma otimização moderada, sugerindo que, embora tenha havido aprimoramento no acesso à L2 cache, a diferença não foi tão significativa como inicialmente previsto. Esse impacto mais modesto na eficiência do cache, em comparação com a otimização da carga computacional, destaca a especificidade do contexto deste estudo.

4.1.3 LARGURA DE BANDA DE MEMÓRIA

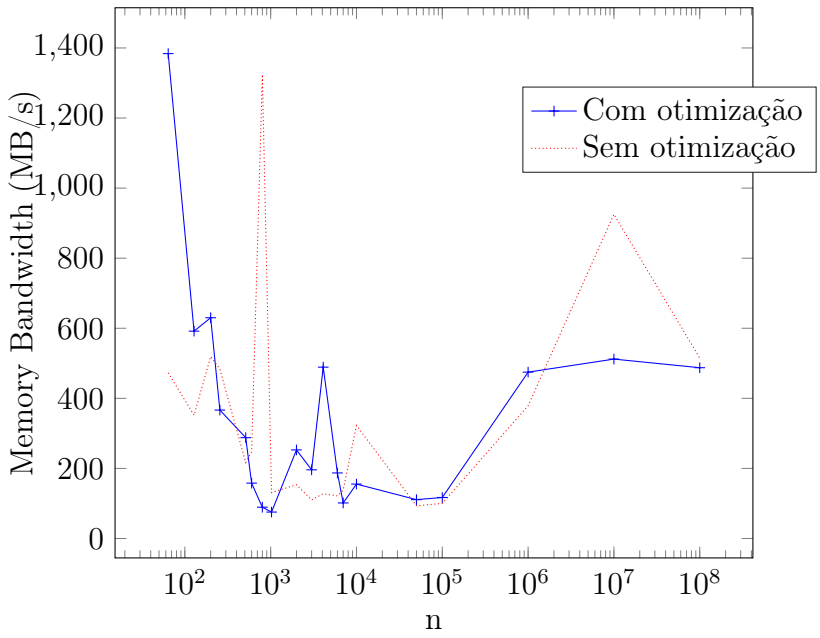


Figura 3 – Gráfico com a relação da largura de banda em MB/s por tamanho da entrada na geração da matriz.

No gráfico 3, que representa a largura de banda, a transição da matriz 2D para um vetor unidimensional inicialmente apontava para uma melhoria, mas os resultados revelaram uma oscilação notável e uma melhoria bem enxuta. A mudança na estrutura de dados não teve o impacto esperado, indicando complexidades adicionais que influenciam a eficiência do acesso à largura de banda, uma vez que o tempo de execução também pode afetar esse dado (MB/s).

4.1.4 TEMPO DE EXECUÇÃO

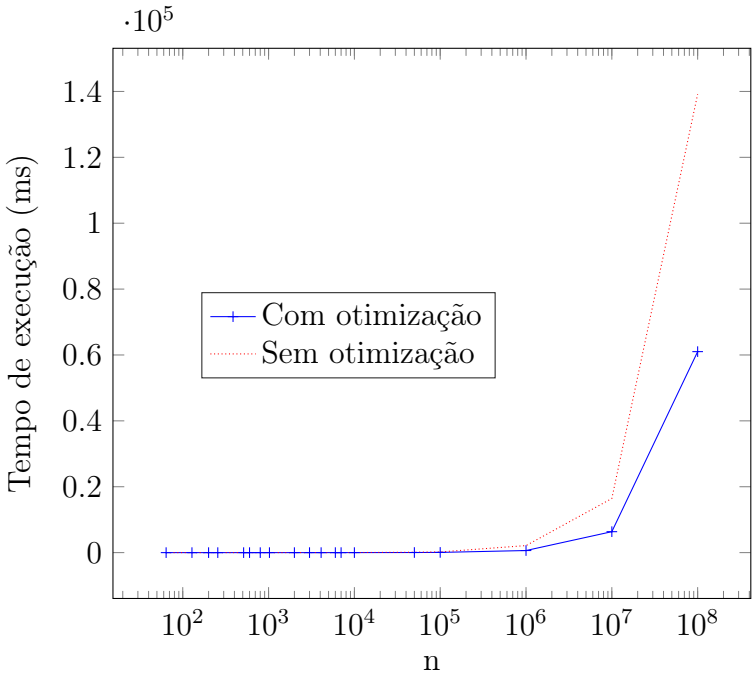


Figura 4 – Gráfico com a relação do tempo de execução em ms por tamanho da entrada na geração da matriz.

O tempo de execução, conforme demonstrado na Figura 4, apresentou uma notável melhoria para entradas de grande escala. A transição da matriz 2D para um vetor unidimensional, otimizando o processo computacional através da cópia da diagonal em vez do cálculo individual, desempenhou um papel crucial nesse aprimoramento. Esta simplificação eficiente reflete a importância de considerar a estrutura da matriz ao projetar algoritmos para cenários de grande escala, resultando em ganhos significativos no tempo de execução. Outra coisa que pode ter contribuido seria que as funções de arredondamento de pontos flutuantes (que são utilizadas em TODAS as operações aritméticas) se tornaram *inline*, o que aumenta a performance de funções curtas que são utilizadas demasiadamente no código

4.2 CÁLCULO DO RESÍDUO

4.2.1 MFLOP

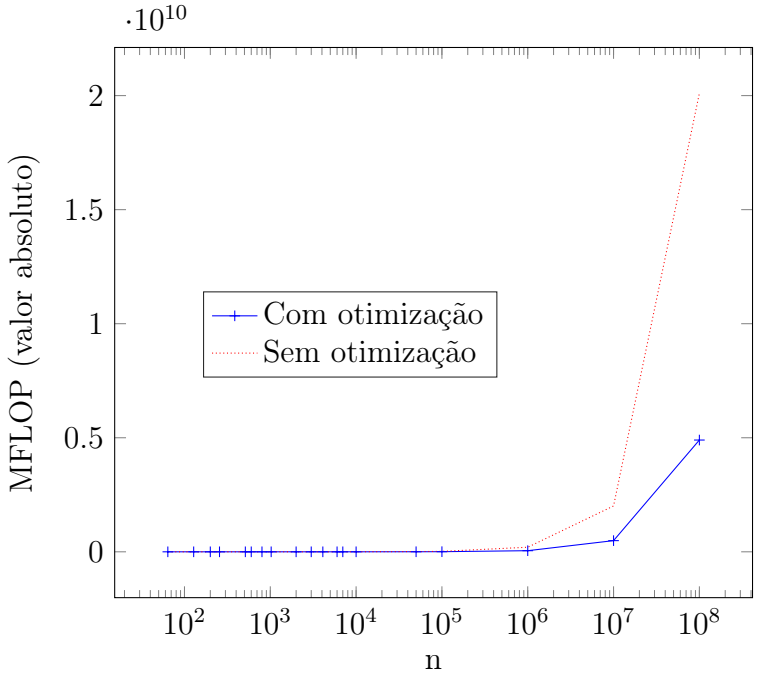


Figura 5 – Gráfico com a relação de MFLOP por tamanho da entrada no cálculo do resíduo.

No cálculo do resíduo como evidenciado na figura 5, a versão otimizada, ao incorporar o reaproveitamento de cálculos anteriores para potências e substituir exponenciações por multiplicação, resultou em uma considerável melhoria nos MFLOPs atigindo o mesmo resultado com a mesma precisão.

4.2.2 TAXA DE CACHE MISS NA L2

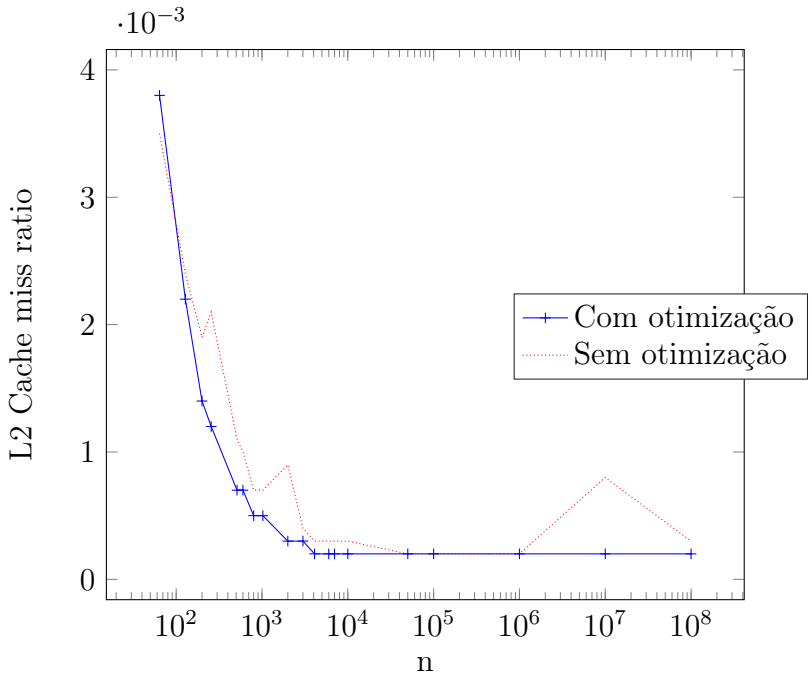


Figura 6 – Gráfico com a relação de l2 cache miss ratio por tamanho da entrada no cálculo do resíduo.

No gráfico 8, pode-se perceber que tivemos uma melhor em quase todos os pontos do gráfico. A melhora é explicada pelo uso de constantes nos parâmetros da função. Já que, dessa forma, a função sabe que o valor não precisa ser re-lido ou alterado com o tempo. Dessa forma, conseguimos entender a melhora apresentada no gráfico. Também, no ponto crítico da função acessa-se somente um vetor, assim, não disputam linhas de cache.

4.2.3 LARGURA DE BANDA DE MEMÓRIA

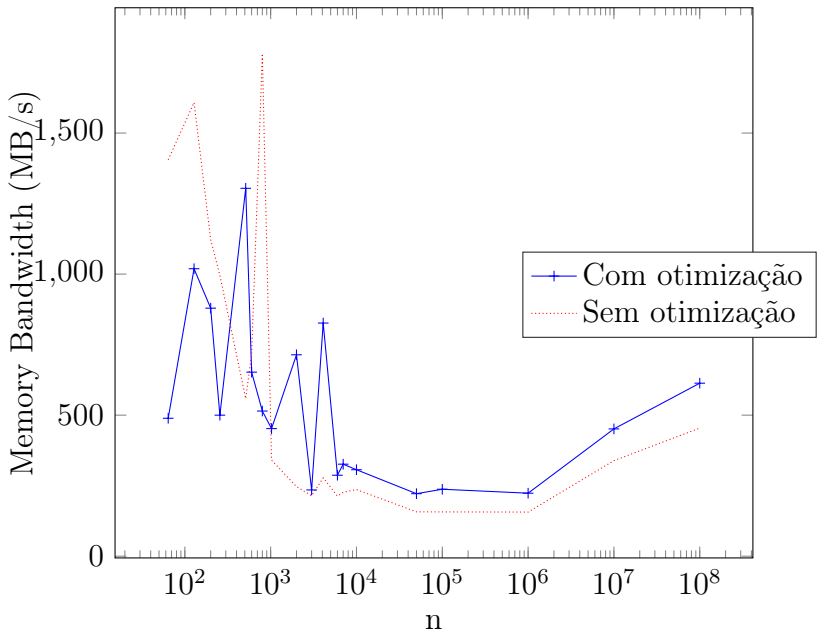


Figura 7 – Gráfico com a relação da largura de banda em MB/s por tamanho da entrada no cálculo do resíduo.

A leitura e o armazenamento de dados foram melhoradas em comparação a versão não-otimizada. Algumas melhorias podem ser citadas para justificar tal progresso. Primeiramente, deixamos de utilizar uma função de potenciação no cálculo do resíduo, o que diminui o tempo de execução e consequentemente aumenta a velocidade em que os dados transitam na memória. Outra melhoria que pode ser citada é que, no loop mais interno, deixamos de acessar outros vetores além de b. Isso diminui a disputa de dados da cache, o que melhora o cache miss ratio e consequentemente a velocidade que os dados são encontrados na cache. Essas melhorias e as justificativas representam o gráfico mostrado.

4.2.4 TEMPO DE EXECUÇÃO

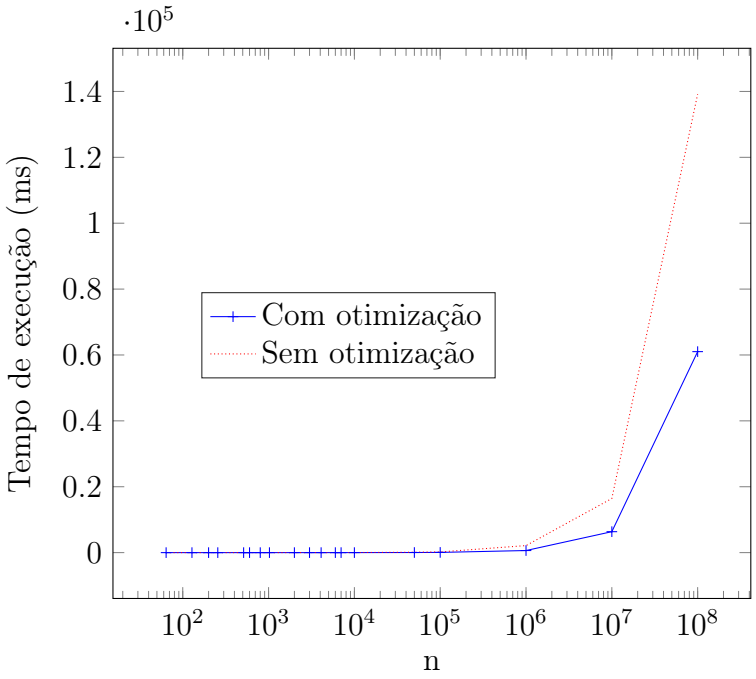


Figura 8 – Gráfico com a relação do tempo de execução em ms por tamanho da entrada no cálculo do resíduo.

Como já dito antes, a não utilização de uma função computacionalmente cara (potenciação) diminui o tempo de execução necessário no cálculo do resíduo. A diminuição de disputa de cache também pode influenciar uma diminuição do tempo de execução. Vimos no gráfico que a mudança é mais perceptível em entradas maiores, o que faz sentido, já que a duração do loop é proporcional ao tamanho da entrada. Quanto maior o tamanho, mais rápido vai ser comparado com a versão não otimizada.

4.3 RESOLUÇÃO DO SISTEMA

4.3.1 FLOPS

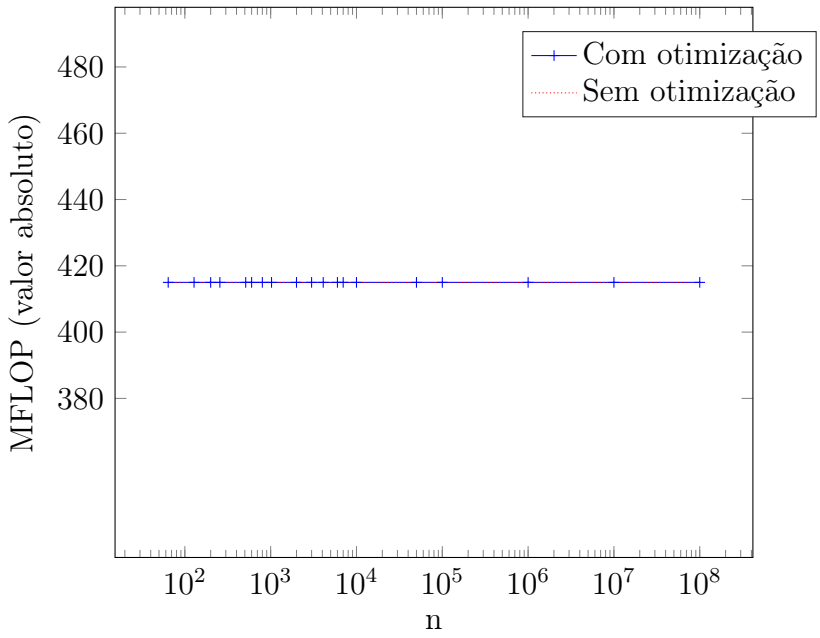


Figura 9 – Gráfico com a relação de MFLOP por tamanho da entrada na resolução do sistema.

No gráfico (Figura 9) dos FLOPS do algoritmo de solução da matriz, observamos uma estabilidade notável, mantendo-se em 415, independentemente do tamanho da entrada, isto porque nenhuma otimização no aspecto aritmético foi realizada.

4.3.2 TEMPO DE EXECUÇÃO

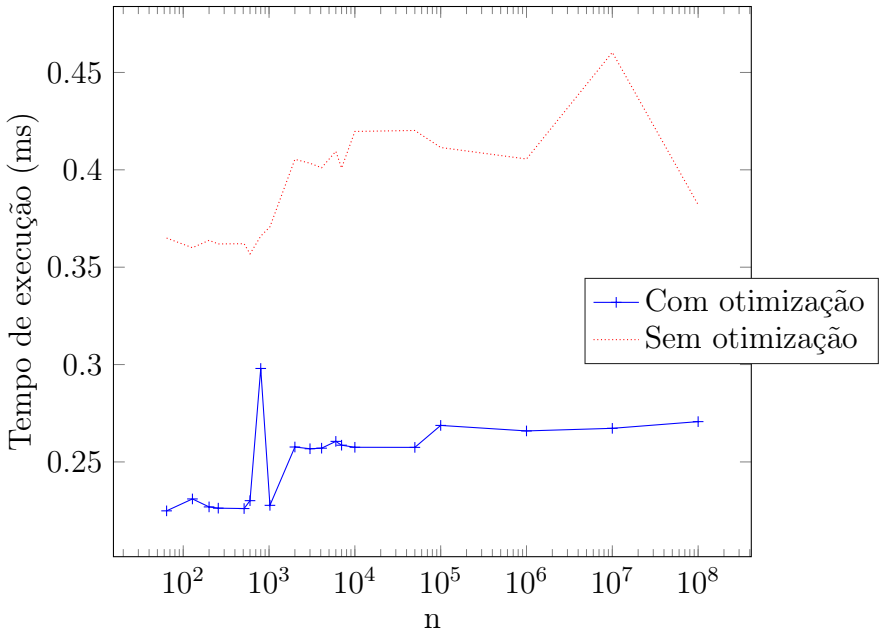


Figura 10 – Gráfico com a relação do tempo de execução em ms por tamanho da entrada na resolução do sistema.

No que se refere ao tempo de execução no gráfico 10, observamos uma melhoria significativa devido à linearização do acesso a dados pela transformação da matriz 2D em 1D e ao uso de memcpy para alterar as linhas da matriz. Essas otimizações se refletiram em uma eficiência considerável, simplificando o acesso aos dados e contribuindo para uma execução mais eficaz do algoritmo.