RELATÓRIO DE OTIMIZAÇÕES APLICADAS AO PROGRAMA DE AJUSTE DE CURVAS

Luan Carlos Maia Cruz
Leonardo Marin Mendes Martin

Sumário

1 TOPOLOGIA MÁQUINA DE TESTES 2

2 MODIFICAÇÕES NAS ESTRUTURAS DE DADOS UTILIZADAS 3
2.01 Em libSL.h . . . 3
2.02 Em libRange.h . . . 3

3 PRINCIPAIS ALTERAÇÕES/ OTIMIZAÇÕES 4
3.01 addRange, subtractRange, timeRange, powerRange . . . 4
3.01 allocateLinearSystem . . . 4
3.02 createLinearSystem . . . 4
3.03 classicEliminationWithPivot e retroSubstitution . . . 4
3.04 calculateResidualVector . . . 4

4 GERAÇÃO DO SISTEMA 5

5 SOLUÇÃO DO SISTEMA 7

6 RESÍDUO 8

7 CONCLUSÃO 10

Referências 10

Lista de Figuras

1 Gráfico Métrica TEMPO - Geração do Sistema. . . . 5
2 Gráfico Métrica L2 CACHE - Geração do Sistema. . . . 5
3 Gráfico Métrica L3 - Geração do Sistema. . . . 6
4 Gráfico Métrica FLOPS DP / FLOPS AVX - Geração do Sistema. . . . 6
5 Gráfico Métrica TEMPO - Solução do Sistema. . . . 7
6 Gráfico Métrica FLOPS DP / FLOPS AVX - Solução do Sistema. . . . 7
7 Gráfico Métrica TEMPO - Resíduo. . . . 8
8 Gráfico Métrica L2 CACHE - Resíduo. . . . 8
9 Gráfico Métrica L3 - Resíduo. . . . 9
10 Gráfico Métrica FLOPS DP / FLOPS AVX - Resíduo. . . . 9

1 – TOPOLOGIA MÁQUINA DE TESTES

Utilizamos a máquina H13 do Departamento de Informática para obter os resultados das métricas a serem comparadas. Abaixo, apresentamos a topologia da máquina utilizada:

CPU Information

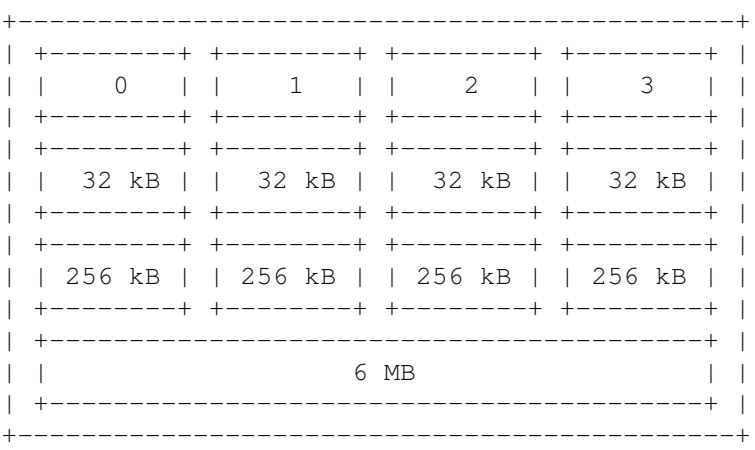
CPU name: Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
CPU type: Intel Coffeelake processor
CPU stepping: 9

Cache Topology

Level 1	Size	32 kB
	Type	Data cache
	Associativity	8
	Number of sets	64
	Cache line size	64
	Cache type	Non Inclusive
	Shared by threads	1
	Cache groups	(0) (1) (2) (3)
Level 2	Size	256 kB
	Type	Unified cache
	Associativity	4
	Number of sets	1024
	Cache line size	64
	Cache type	Non Inclusive
	Shared by threads	1
	Cache groups	(0) (1) (2) (3)
Level 3	Size	6 MB
	Type	Unified cache
	Associativity	12
	Number of sets	8192
	Cache line size	64
	Cache type	Inclusive
	Shared by threads	4
	Cache groups	(0 1 2 3)

Graphical Topology

Socket 0:



2 – MODIFICAÇÕES NAS ESTRUTURAS DE DADOS UTILIZADAS

1) Em libSL.h

```
typedef struct  /* Com Otimizacao */
{
    Range_t *cm; /* Vetor */
    Range_t *vit;
} LinearSystem_t;
```

```
typedef struct /* Sem Otimizacao */
{
    Range_t **cm; /* Vetor de ponteiros */
    Range_t *vit;
} LinearSystem_t;
```

Inicialmente, reformulamos a estrutura da matriz de coeficientes, originalmente declarada no programa sem otimização como um vetor de ponteiros do tipo $Range_t$(Intervalo), para um vetor padrão. Dessa forma a matriz de coeficientes agora é alocada de forma contínua na memória, melhorando de maneira significativa o acesso a dados.

2) Em libRange.h

```
/* Struct of Arrays */
typedef struct /* Com Otimizacao */
{
    Range_t *x;  /* Tornou-se um vetor */
    Range_t *y;  /* Tornou-se um vetor */
} PointsRange_t;
```

```
/* Array of Structs */
typedef struct /* Sem Otimizacao */
{
    Range_t x;
    Range_t y;
} PointsRange_t;
```

Buscando soluções para otimizar o código por meio de modificações nas estruturas de dados, aprimoramos a organização que armazena os intervalos de x e y (pontos fornecidos como entrada). Na versão não otimizada, utilizávamos um Array of Structs; no entanto, enfrentávamos um problema significativo: nem sempre x e y eram utilizados simultaneamente. Com a necessidade de flexibilizar o carregamento dos vetores, transformamos o Array of Structs em uma Struct of Arrays, possibilitando o carregamento dos vetores em memória conforme solicitado e evitando que x e y fossem carregados de forma desnecessária, ou seja, em momentos que não era necessário utilizar os 2 vetores em simultâneo.

3 – PRINCIPAIS ALTERAÇÕES/ OTIMIZAÇÕES

libRange.c

Geral

- Passamos a fazer a chamada de *nextAfter()* somente na geração dos intervalos. Esta função deixou de ser chamada ao realizar os cálculos porque afetava fortemente o desempenho do programa, principalmente dos cálculos em SIMD.

1) addRange, subtractRange, timeRange, powerRange

- Estas funções sofreram otimizações básicas. Descartamos as chamadas das funções *findLargest()* e *findSmallest()*, que eram chamadas pelas funções *addRange()*, *subtractRange()*, *timeRange()* na versão não otimizada ao realizar operações intervalares. Estas duas funções eram chamadas somente para executar a função *nextafter()* afim de determinar os limitantes do intervalo resultante.
- Em razão das entradas sempre serem positivas, a utilização das funções de multiplicação(*timeRange()*) e potenciação(*powerRange()*) criadas anteriormente acabaram sendo descartadas. Na função de multiplicação, por exemplo, é necessário calcular o intervalo resultante apenas da forma: [a,b] * [c,d] = [a*c, b*d], desconsiderando as verificações que eram feitas no Trabalho I.
- Além disso foram realizadas outras otimizações básicas e correções de detalhes específicos relacionados as operações com intervalos.

libSL.c

Geral

- Evitamos a realização das operações intervalares a partir da chamada de funções, pois dessa forma fazemos melhor uso do pipeline e permitimos operações vetoriais SIMD(*Single Instruction, Multiple Data*). Como exemplo, podemos citar a mudança dos cáculos de somas, subtrações, multiplicações e potências, que tiveram suas respectivas funções descartadas, utilizando suas formas simplificadas no código.

1) allocateLinearSystem

- A matriz de coeficientes e o vetor de termos independentes são inicializados com zero, utilizando a função *memset()*

2) createLinearSystem

- Aprimoramos o acesso à memória. Na versão não otimizada, o vetor de structs que continha os intervalos x e y era acessado repetidamente para produzir os cálculos. Na versão otimizada, fundimos loops e implementamos uma melhoria na forma em que os cálculos são realizados, focando no aproveitamento dos dados(potências) em cache, somando os termos do Sistema Linear e Vetor de Termos Independentes de forma parcial, o que reduziu drásticamente o acesso aos valores de x e y.
- Especialmente no cálculo das potências, utilizamos um vetor temporário para armazenar os valores das potências do ponto calculado em determinada iteração. Dessa forma conseguimos fazer cáculos em SIMD.

3) classicEliminationWithPivot e retroSubstitution

- Para estas funções que fazem parte da solução do sistema linear, não foram realizadas otimizações de grande impacto, devido a restrição do tamanho do sistema ser 5x5(polinômio de grau 4).
- Foram realizadas apenas otimizações básicas, evitando chamadas de funções para as operações intervalares quando possível.

4) calculateResidualVector

- Similar ao que foi implementado na função *createLinearSystem()*, utilizamos um vetor temporário para armazenar as potências do ponto calculado. Dessa forma, foi possível multiplicar os coeficientes pelas potências usando SIMD, armazenando o resultado no vetor de resíduo.

4 – GERAÇÃO DO SISTEMA

TEMPO

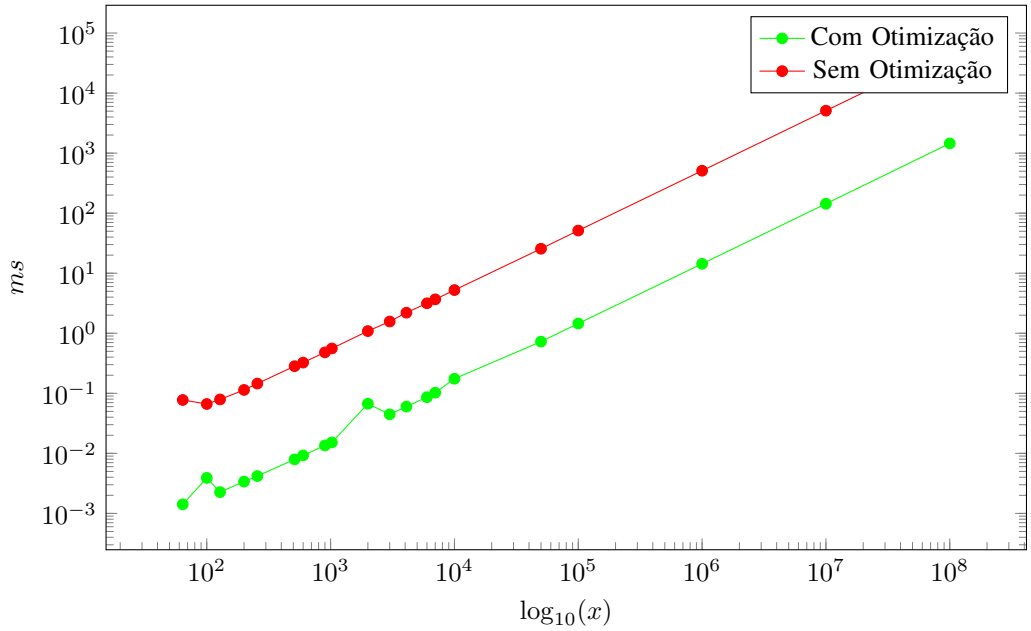


Figura 1. Gráfico Métrica TEMPO - Geração do Sistema.

Com as melhorias aplicadas, agora o sistema linear é gerado aproximadamente 100 vezes mais rápido do que na versão sem otimização. Isso se deve à mudança que fizemos na forma de acessar os valores x e y. Também estamos utilizando um vetor temporário que funciona como uma *look ahead table* armazenando os valores das potências de x, contribuindo para que operações SIMD sejam realizadas.

L2 CACHE

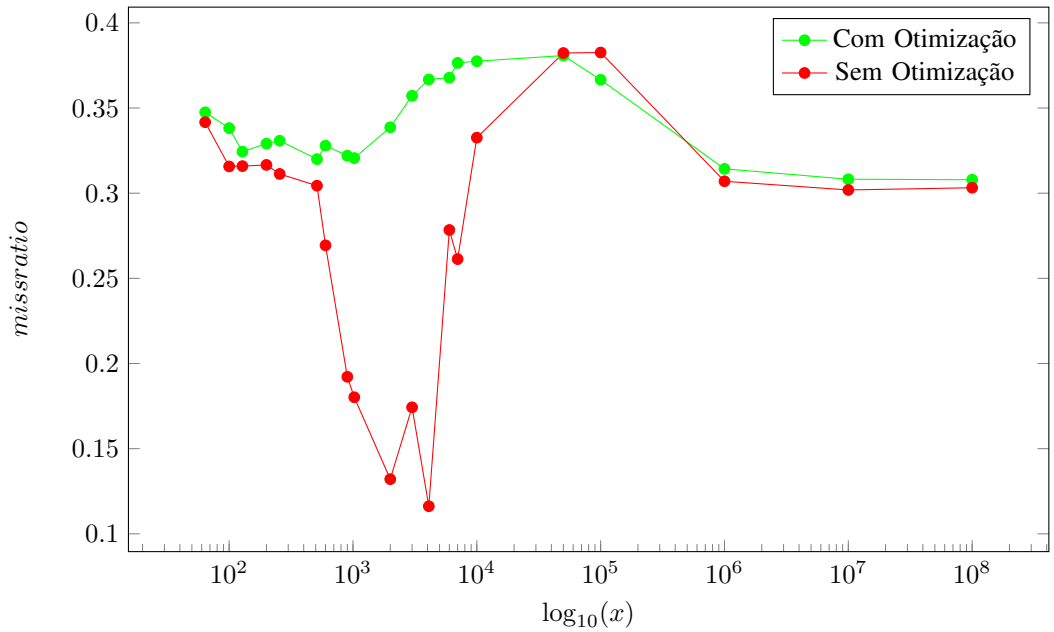


Figura 2. Gráfico Métrica L2 CACHE - Geração do Sistema.

Otimizações em determinadas partes do código muitas vezes resultam em implicações para outras, e isso é evidenciado pela ausência de melhorias significativas na métrica de *miss ratio* da cache L2, contrariando nossas expectativas iniciais. Ao alocar os pontos de forma contínua na memória, nossa intenção era aproveitar melhor a localidade espacial, visando a redução dos misses na cache. No entanto, ao analisar o gráfico, observamos uma depressão entre os pontos 10e-3 e 10e-4 na versão não otimizada. Isso sugere que, nessa versão, os dados podem ter sido completamente armazenados em cache. O que nos supreendeu é que a versão otimizada não apresentou nenhuma uma melhoria significativa na *miss ratio*. Uma possível explicação para esse fenômeno pode estar na alocação do vetor da matriz na versão otimizada, que pode não ter sido feita de

maneira alinhada à memória. Essa falta de alinhamento pode ter impedido a realização da redução na taxa de misses, tendo um desempenho sem grandes melhorias.

L3

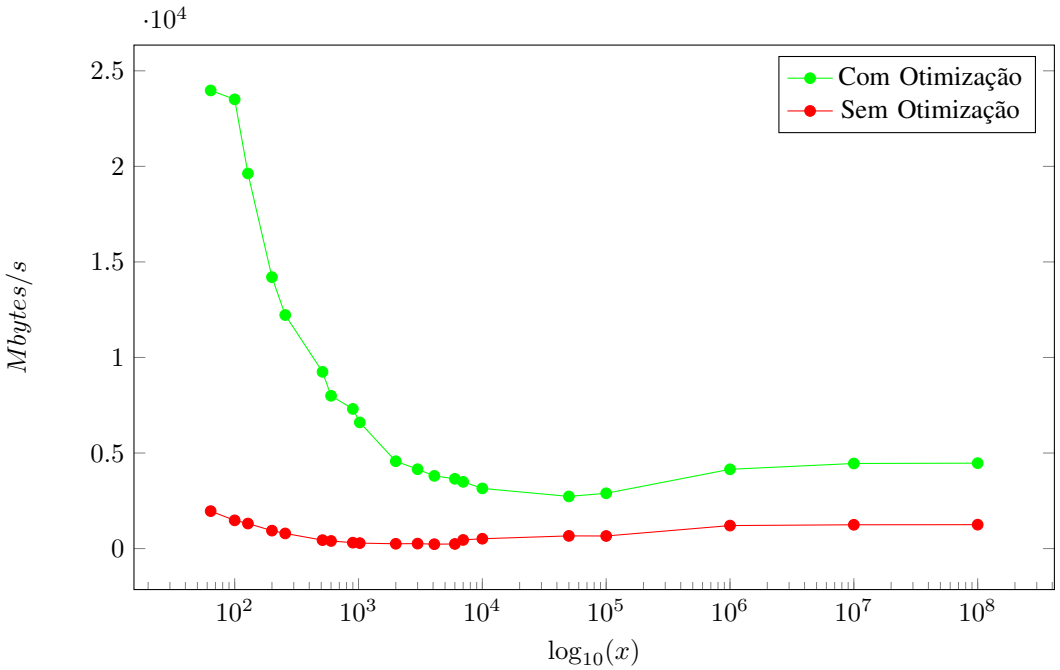


Figura 3. Gráfico Métrica L3 - Geração do Sistema.

A mudança na forma de acessar os valores x e y também impactou na largura de banda, o fato de as posições estarem alocadas de forma contínua na memória facilicita o acesso e faz um melhor aproveitamento da localidade espacial, pelo fato dos endereços de memória estarem próximos

FLOPS DP / FLOPS AVX

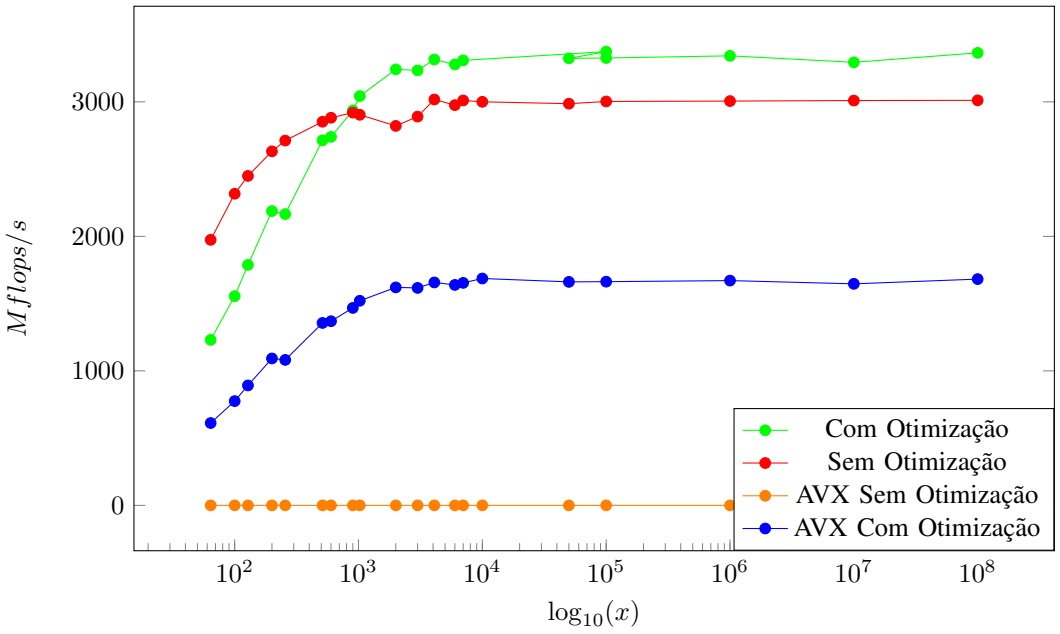


Figura 4. Gráfico Métrica FLOPS DP / FLOPS AVX - Geração do Sistema.

As otimizações mais significativas foram aplicadas durante a geração do sistema linear. Optamos por evitar o uso de operações intervalares, realizando as operações diretamente. Isso resultou em um aumento na contagem de FLOPS. Por outro lado, conseguimos vetorizar e elevar a contagem de *Flops Avx*, graças à fusão de loops e à implementação de um vetor temporário que funciona como uma *look ahead table*, armazenando as potências de x. Destacando que obtemos uma taxa de vetorização de aproximadamente 40%

5 – SOLUÇÃO DO SISTEMA

TEMPO

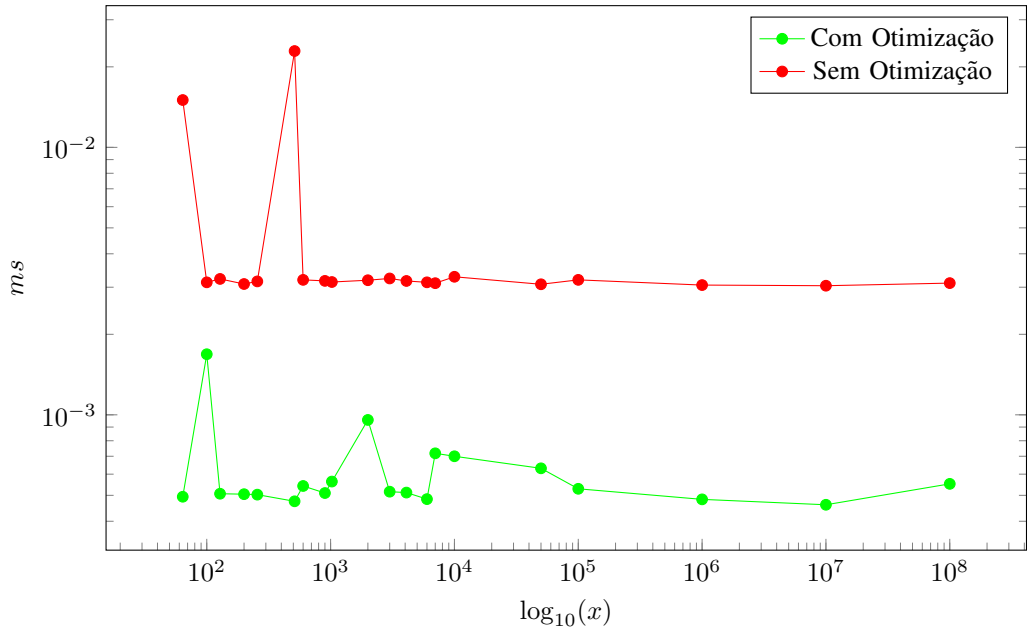


Figura 5. Gráfico Métrica TEMPO - Solução do Sistema.

Com as otimizações, o tempo da solução do sistema está aproximadamente 10 vezes mais rápido. A diminuição no tempo de execução é resultado da alteração na forma de acesso aos pontos x e y, juntamente com a decisão de evitar o uso das funções de operação intervalar. Vale a pena destacar que não foi aplicada nenhuma otimização que poderia impactar de forma siginificativa a solução do sistema linear, dessa forma o resultado obtido na métrica do tempo é consequência de otimizações mais básicas

FLOPS DP / FLOPS AVX

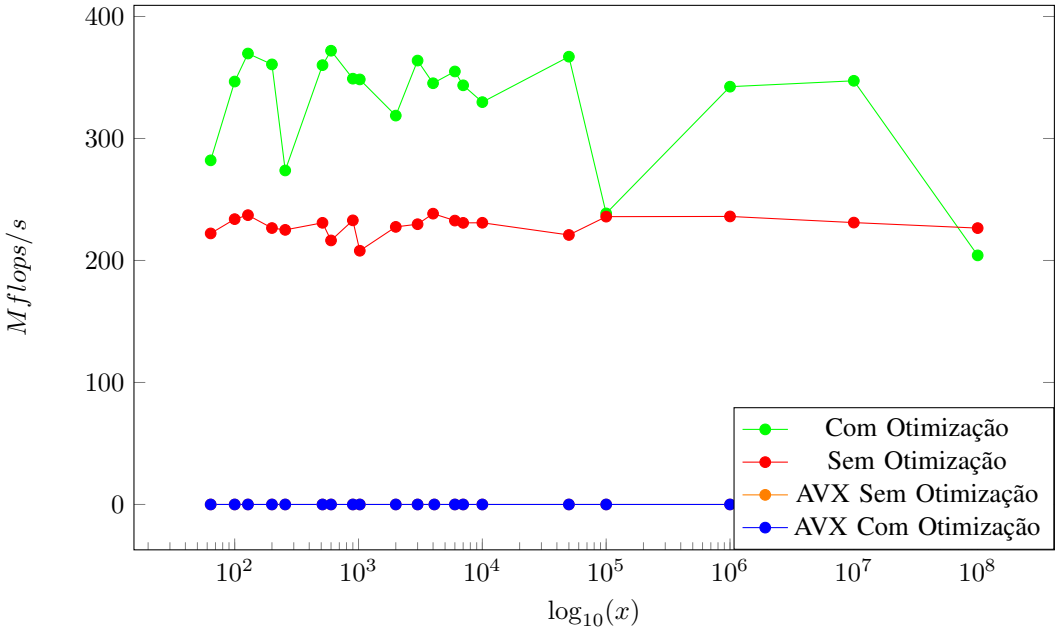


Figura 6. Gráfico Métrica FLOPS DP / FLOPS AVX - Solução do Sistema.

Na solução do sistema, optamos por não utilizar as funções intervalares para aprimorar o tempo de execução. Como resultado dessa decisão, realizamos as operações diretamente dentro das funções relacionadas à solução do sistema (Retrosubstituição e Eliminação Gaussiana). Essa escolha teve como consequência um aumento no número de *Flops* (operações de ponto flutuante por segundo). Embora o uso das funções intervalares levasse a uma contagem menor de *Flops*, priorizamos a melhora no tempo de execução.

6 – RESÍDUO

TEMPO

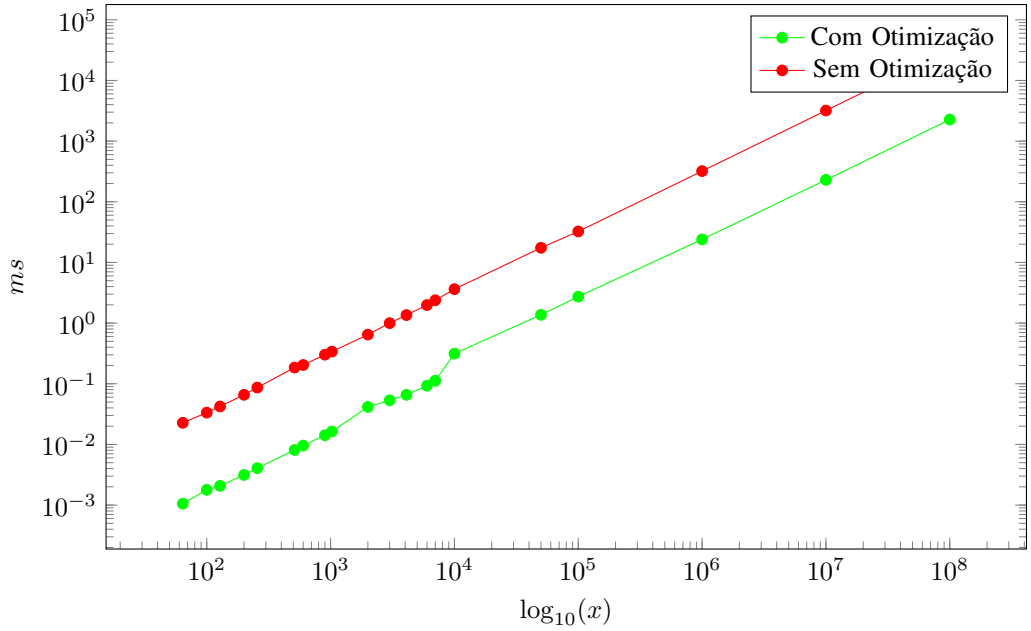


Figura 7. Gráfico Métrica TEMPO - Resíduo.

O aprimoramento no tempo do resíduo do sistema é resultado das melhorias mencionadas anteriormente: a modificação na forma de acesso aos pontos x e y e a decisão de evitar o uso de funções intervalares. Além disso, aplicamos a mesma estratégia utilizada na geração do sistema linear. Isso envolveu o uso de um vetor temporário que funciona como uma *look ahead table* para armazenar as potências, o que possibilita a realização de operações SIMD.

L2 CACHE

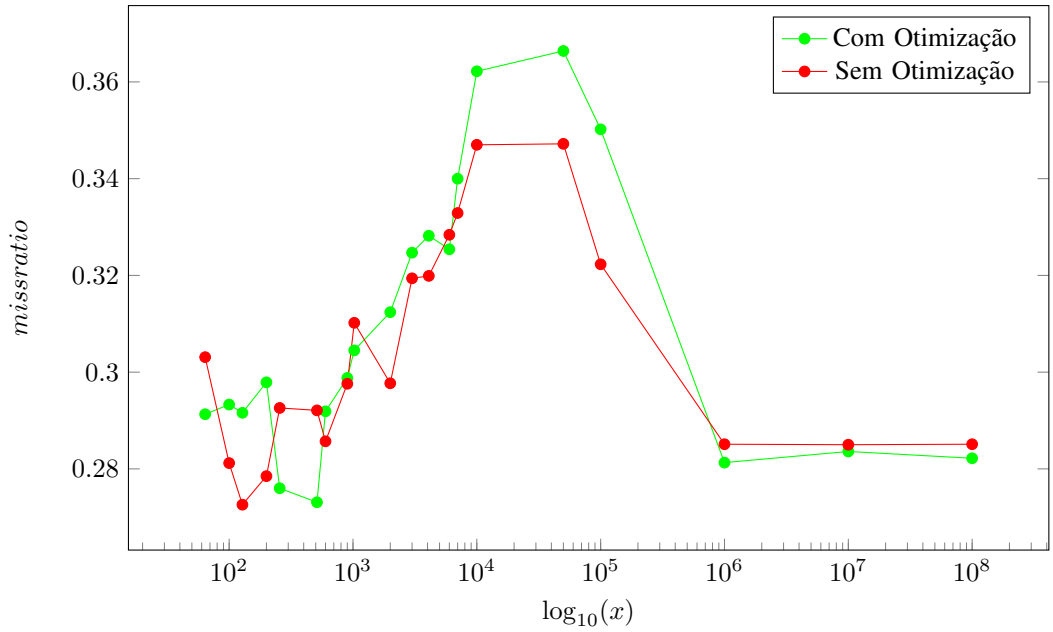


Figura 8. Gráfico Métrica L2 CACHE - Resíduo.

Como evidenciado no gráfico, observamos que a miss ratio da cache L2 não apresentou melhorias significativas. Inicialmente, esperávamos uma redução na taxa de misses ao alocar os dados como um vetor contínuo, pois isso geralmente proporciona melhor aproveitamento da localidade espacial, uma vez que os endereços estão próximos. Entretanto, é possível que a falta de melhoria na *miss ratio* esteja relacionada à alocação da matriz, que pode não ter sido alocada alinhada na memória. Essa alocação inadequada pode ter levado a um aumento na *miss ratio* em determinados pontos, contrariando as expectativas iniciais.

L3

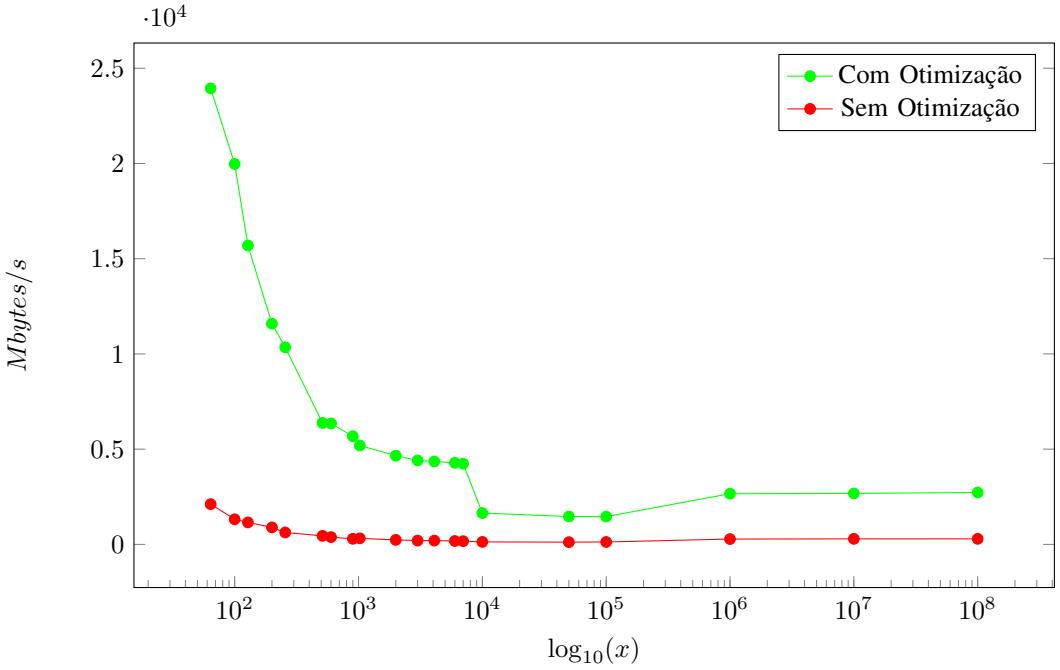


Figura 9. Gráfico Métrica L3 - Resíduo.

Pelas mesmas razões comentadas sobre esta métrica na Geração do Sistema Linear, a mudança na forma de acessar os pontos x e y foi o que possibilitou melhorar a largura de banda no Resíduo do sistema. Neste cenário, o vetor x que armazena os pontos das abcissas, era acessado uma única vez, antes um vetor de ponteiros passou a ser alocado de forma contínua

FLOPS DP / FLOPS AVX

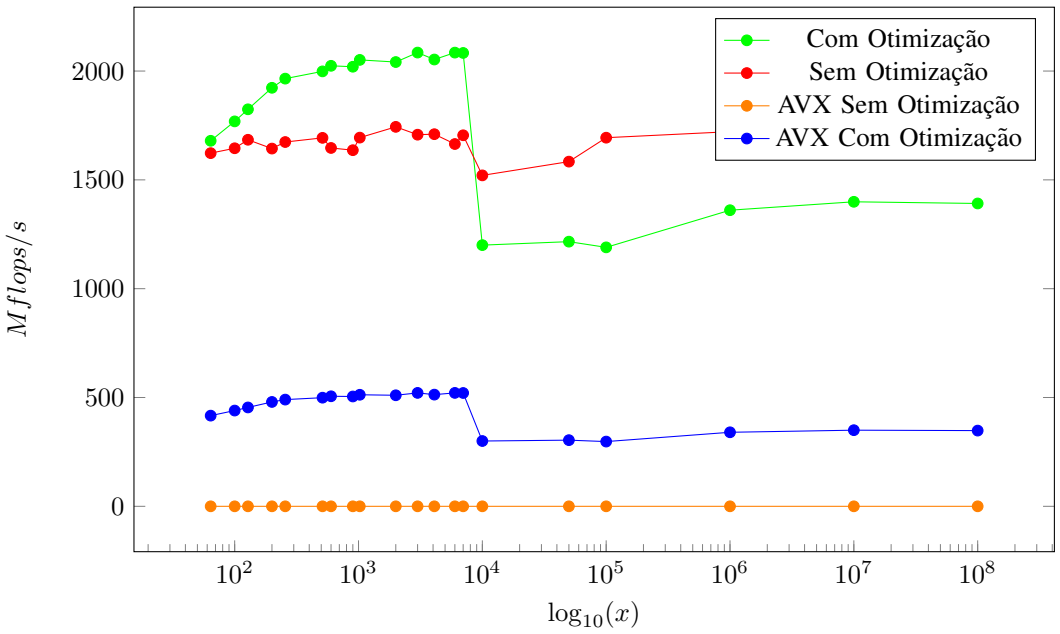


Figura 10. Gráfico Métrica FLOPS DP / FLOPS AVX - Resíduo.

Similar a geração do sistema linear, aplicamos também no resíduo um vetor temporário que funciona como uma *look ahead table* e evitamos o uso das funções de operação intervalar, isso gerou um aumento na contagem de *Flops*, mas possibilitou o uso de operações *SIMD*. Destacando que obtemos uma taxa de vetorização de aproximadamente 11%

7 – CONCLUSÃO

Com o objetivo de aprimorar o Trabalho I, dedicamos nossa atenção à identificação de pontos passíveis de otimização no desempenho do programa. Nesse contexto, analisamos melhorias nas estruturas de dados, no acesso à memória, na redução das chamadas de funções e na simplificação dos cálculos, levando em consideração que as entradas agora são restritas a valores positivos.

Em nossa abordagem de otimização, dedicamo-nos de forma prioritária ao aprimoramento do acesso à memória e à redução do tempo de execução, considerando esses como os pontos centrais para impulsionar a eficiência do programa.

Assim, modificamos a estrutura de dados do Trabalho I, migrando de um Array de Structs para uma Struct de Arrays. Agora, ao invés de armazenar pares (x, y) em um único vetor, optamos por manter os pontos x e seus respectivos y's em vetores separados. Essa escolha não apenas evita a transferência desnecessária de valores da memória, mas também abre a possibilidade de trazer mais valores úteis de uma só vez, otimizando o tempo de execução e aprimorando a vetorização do código.

Por outro lado, com o intuito de diminuir o tempo dedicado às etapas cruciais do ajuste polinomial, optamos por descartar o uso da maioria das funções responsáveis pelas operações intervalares. Em vez disso, simulamos essas operações de maneira simplificada, sem aumentar a extensão do código. Em decorrência dessa simplificação, foi possível até mesmo considerar a eliminação de grande parte das funções da biblioteca intervalar libRange.

Por fim, os resultados obtidos foram superiores as nossas expectativas iniciais, de modo que o tempo dos pontos principais do ajuste polinomial ficaram de 10-100x mais rápidos e também por conseguirmos fazer o uso de operações SIMD.

Referências

[1] DELGADO, Armando. DERENIEVICZ, Guilherme, *Engenharia de Perfomance*, Departamento de Informática - UFPR, 2023, 34.

[2] DELGADO, Armando. DERENIEVICZ, Guilherme, *Otimização Básica Código Serial*, Departamento de Informática - UFPR, 2023, 24.

[3] DELGADO, Armando. DERENIEVICZ, Guilherme, *Otimização de Acesso a Dados*, Departamento de Informática - UFPR, 2023, 37.