Otimização de Desempenho para Sistemas Lineares Esparsos com Pré-condicionantes

SERGIO SIVONEI DE SANT'ANA FILHO GRR20242337
EDUARDO KALUF GRR20241770

3 de dezembro de 2025

1 Introdução

Este trabalho visa realizar principalmente a otimização de operações computacionais chave dentro do algoritmo do Método dos Gradientes Conjugados (CG), visando calcular de forma mais eficiente a solução de sistemas lineares da forma $Ax = b$.

Especificamente, busca-se aprimorar e avaliar o desempenho do programa computacional desenvolvido no 1º Trabalho Prático.

As operações críticas que foram alvo de otimização são:

- A Iteração Central do Método de Gradiente Conjugado utilizando o Pré-condicionador de Jacobi (PCG).
- O Cálculo do Resíduo do sistema.

A motivação para estas otimizações reside na natureza intensiva em computação dessas operações, bem como no grande tráfego de memória, especialmente ao lidar com matrizes esparsas de grande dimensão.

2 Otimizações Implementadas

Nesta seção, detalhamos as técnicas de otimização aplicadas às operações, descrevendo seu impacto teórico e prático no desempenho, bem como os desafios encontrados durante o processo de implementação e ajuste.

2.1 Otimização do Gradiente Conjugado (PCG)

A otimização da iteração do Gradiente Conjugado, que envolve múltiplas operações de produto matriz-vetor ($A \times v$), foi realizada primariamente através da alteração do formato de armazenamento da matriz do sistema.

2.1.1 Matriz CSR (Compressed Sparse Row)

O formato **CSR** (Compressed Sparse Row) foi a principal modificação estrutural implementada.

Este foi o terceiro método de armazenar a matriz que experimentamos, primeiramente tentamos guardá-las em um único vetor sem offsets e depois tentamos em diversos vetores de tamanhos diferentes. Desistimos de ambas as implementações, pois não pareciam práticas para realizar as operações e não ofertavam nenhum grande benefício aparente além da diminuição do tanto de memório utilizada

O método escolhido é um esquema de armazenamento de matrizes projetado especificamente para lidar com matrizes esparsas, ou seja, matrizes que possuem uma alta proporção de elementos nulos.

Este formato elimina a necessidade de armazenar informações redundantes (os zeros), operando sobre apenas três vetores de dimensão reduzida:

- **'Values'**: armazena apenas os valores não nulos da matriz.
- **'Col_indices'**: armazena os índices das colunas correspondentes a cada valor não nulo.
- **'Row_pointers'**: armazena os índices que indicam o início de cada nova linha nos vetores 'values' e 'col_indices'.

A principal vantagem do **CSR** é sua extrema eficiência para operações de multiplicação matriz-vetor ($A \times v$).

Dada a estrutura da iteração do Gradiente Conjugado (PCG), que exige duas ou mais dessas multiplicações por passo, o uso do **CSR** é crucial para reduzir o tempo de execução. Esta abordagem melhora a localidade de dados

e permite **acessos sequenciais** à memória, que são mais rápidos e otimizam a utilização da cache.

Vetorialização (Uso de AVX): Além disso, devido ao modo como o armazenamento **CSR** funciona, foi possível utilizar **AVX** neste método, deixando a iteração do método ainda mais rápida. A vetorização utiliza as Extensões Avançadas de Vetor (**AVX**) do processador, permitindo que uma única instrução da CPU (SIMD - Single Instruction, Multiple Data) opere simultaneamente sobre múltiplos dados. Isso foi essencial para **diminuir drasticamente** o tempo de execução das operações vetoriais, explorando a capacidade de paralelismo ao nível de hardware.

2.2 Otimização do Cálculo do Resíduo

O cálculo do resíduo $r = b - Ax$ requer primariamente uma multiplicação matriz-vetor e uma subtração de vetores. A otimização focou em garantir a máxima eficiência nessas operações, complementando a desempenho obtida com a implementação do **CSR** para multiplicação entre matriz vetor.

Foram feitas algumas pequenas alterações de otimização, porém o maior ganho de desempenho foi devido à estrutura **CSR**. Houve também a tentativa de implementar o *loop unroll & jam*, porém sem sucesso, pois os testes mostraram uma piora significativa no tempo de execução do calcúlo.

Outra otimização foi realizar parte do cálculo da norma dentro do laço que cálcula o vetor resíduo, assim aproveitando um valor que já está em memória sem a necessidade de outro loop que iria acessar todas as posições novamente. Todavia, nos testes realizados, não foi observado grande mudança, na maioria apenas pequenas variações de tempo.

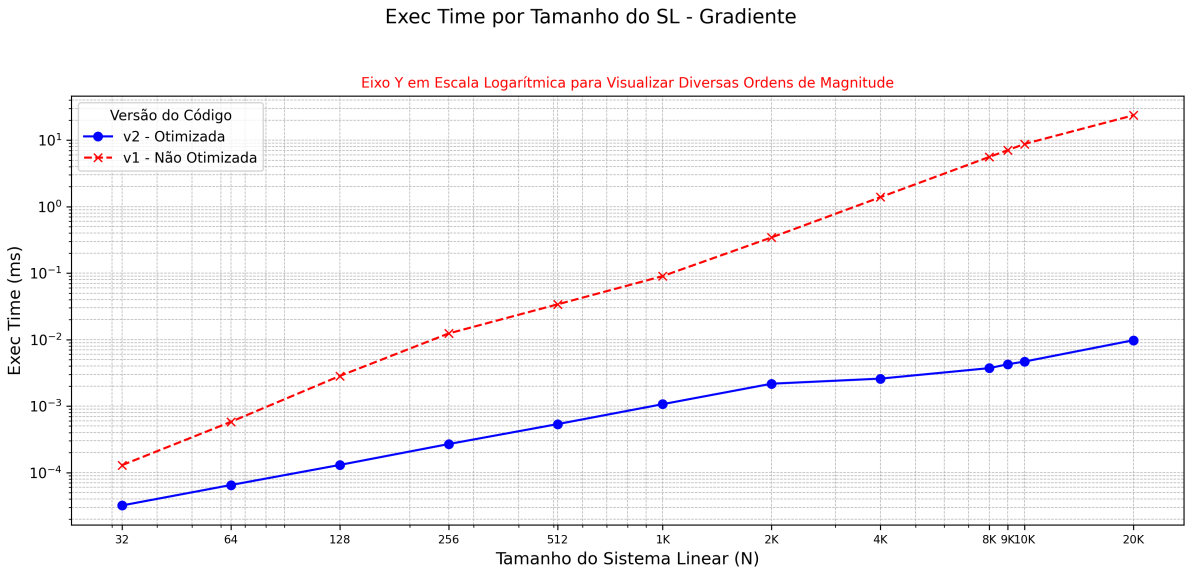
2.3 Otimização Adicional: Matrizes Simétricas Positivas

Além das duas operações fundamentais também otimizamos como a matriz simétrica positiva é calculada. O método **CSR** é ruim quando se é necessário percorrer pelas colunas, sendo assim, a multiplicação de duas matrizes do jeito convencional ficaria mais lento que o esperado, a fim de contornar isso, fizemos a multiplicação entre a matriz principal e a transposta pelas linhas, ou seja, ao invés de multiplicar linhas por colunas, multiplicamos linhas por linhas. Além disso, o método apenas itera quando existe algum resultado possível, multiplicações que irão dar zero no final são ignorados, pois não são necessárias.

3 Análise dos Gráficos

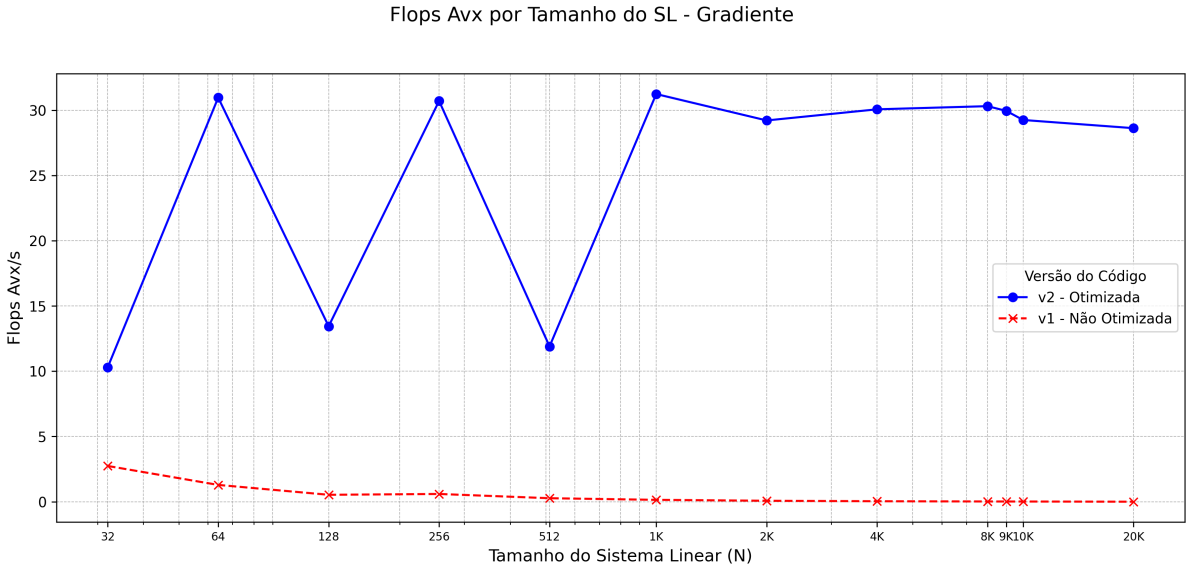
A seguir, seguem os gráficos com as métricas extraídas pelo programa **LIKWID**, conforme solicitado, para diversos valores de N.

3.1 Método dos Gradientes Conjugados (CG)



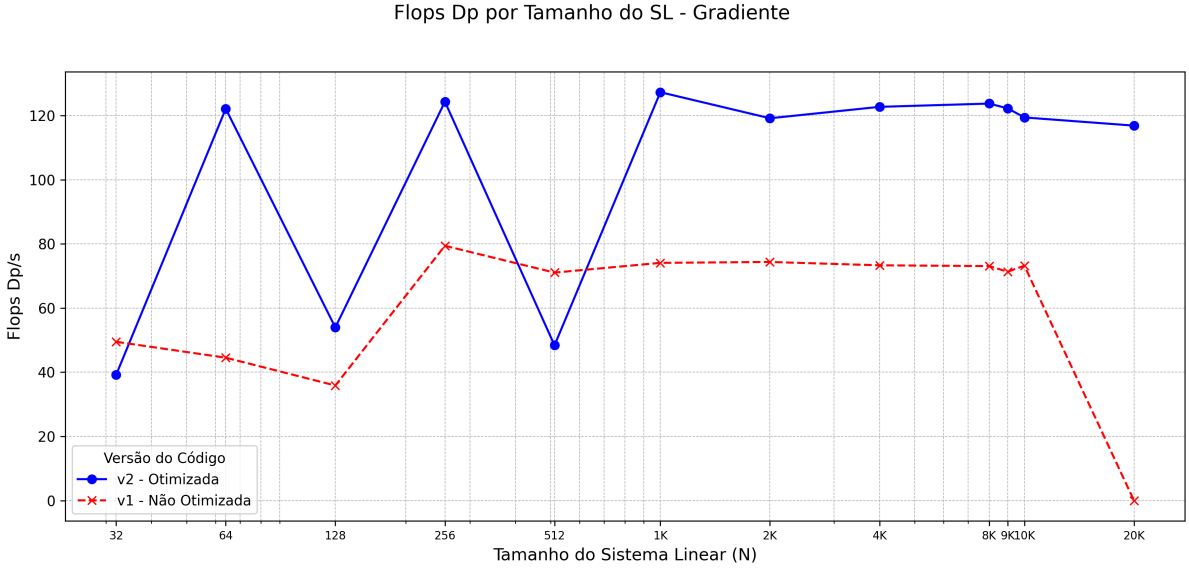
É possível observar a disparidade no tempo de execução entre o programa antigo e o novo. Torna-se evidente que os métodos de otimização utilizados obtiveram sucesso ao reduzir drasticamente o tempo de execução para a iteração do gradiente, especialmente para grandes valores de N, que é o foco principal do nosso interesse.

Essa redução no tempo é justificada pela melhora nas métricas que analisaremos a seguir.



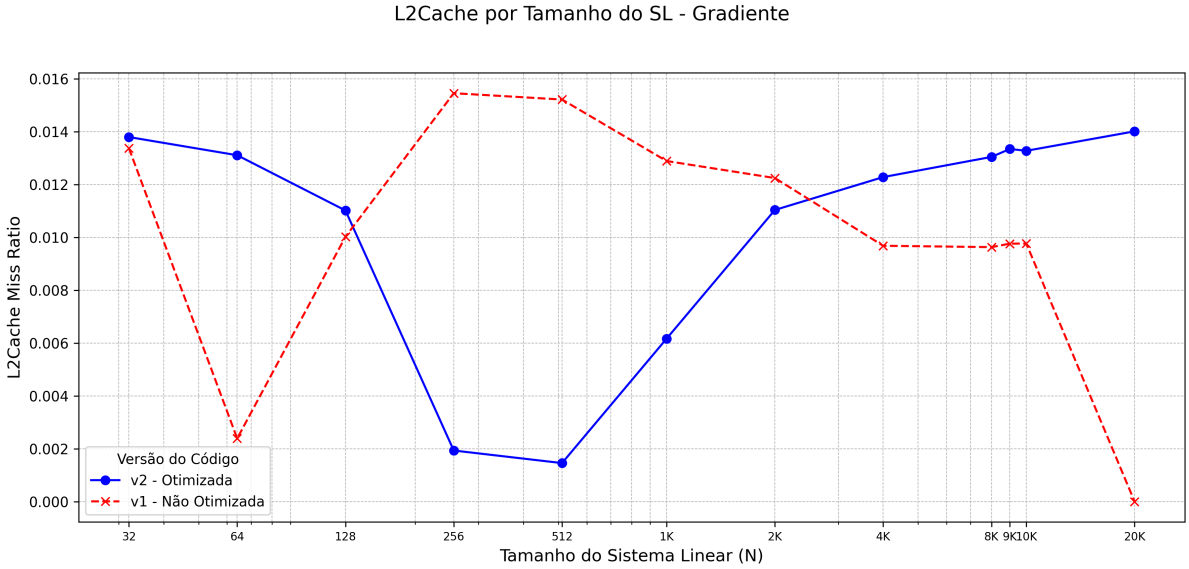
Ao analisarmos os Flops **AVX** podemos concluir que houve um uso das Extensões Avançadas de Vetor (**AVX**) em maior escala, coisa que otimiza o programa ao fazer uso de instruções SIMD.

Certas inconstancias são devido a DETALHAR AQUI



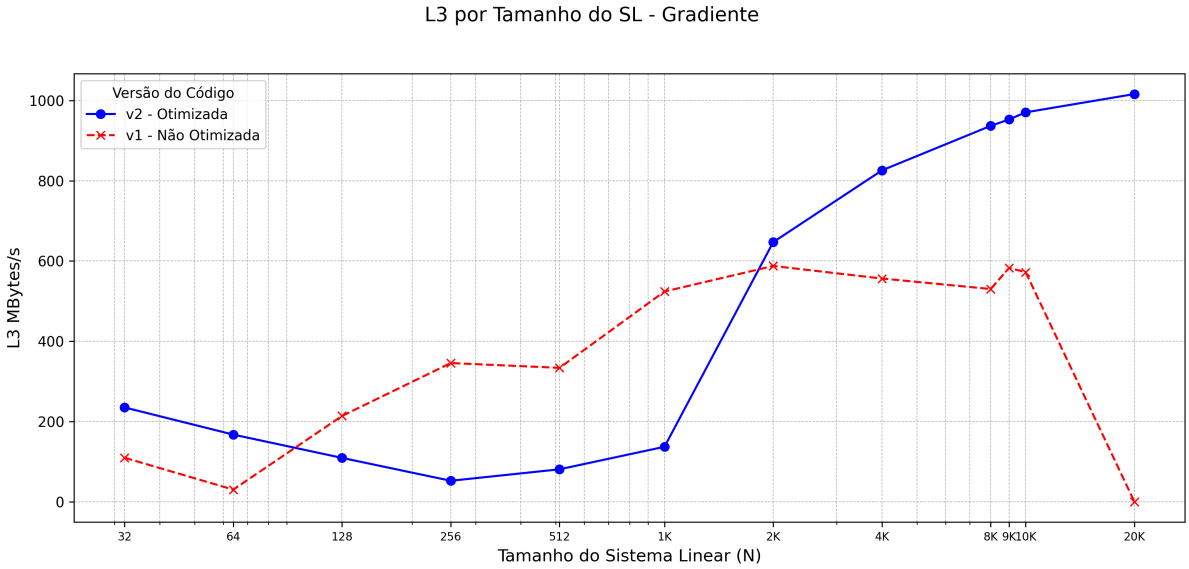
Houve também um aumento no número de Flops Double precision por segundo, coisa que impacta diretamente na velocidade do programa, já que

se há um maior numero de operações ocorrendo num mesmo intervalo de tempo, quer dizer que a máquina terminará os cálculos antes se comparado à primeira versão.



Analisando o gráfico de miss ratio da cache L2 é possível inferir que: Para sistemas de tamanhos condizentes com a Cache L2 (256-512) temos um desempenho expressivo da versão otimizada. Todavia quando o tamanho aumenta muito temos uma piora por não ser mais possível aproveitar a localidade na cache, o que chega deixar o miss ratio da versão otimizada levemente pior que da versão original.

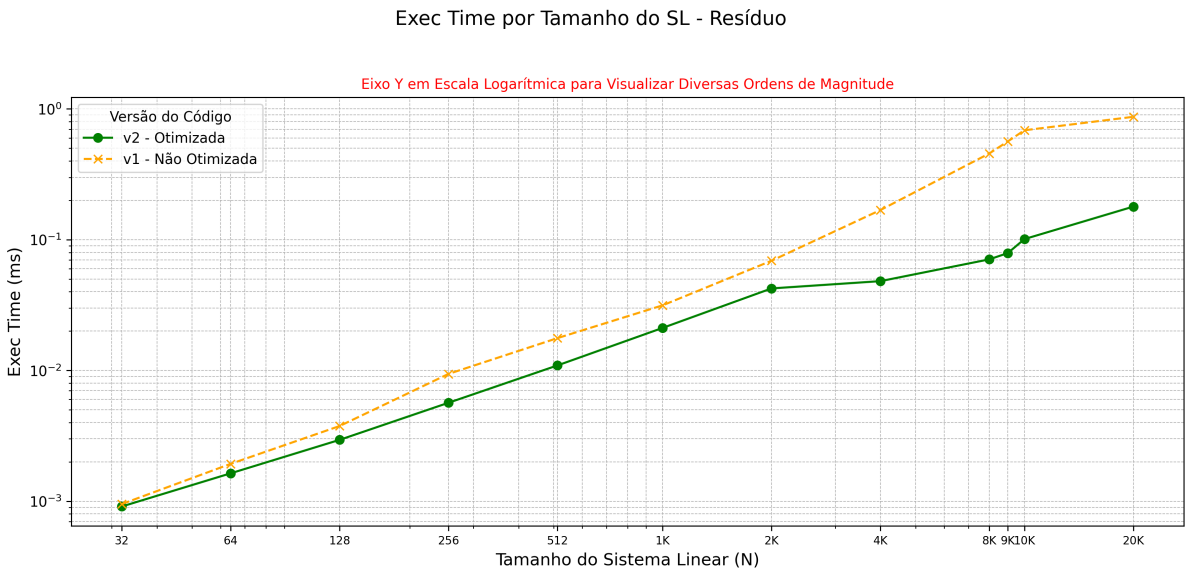
Além disso, o pode-se afirmar que a primeira versão do trabalho tem um desempenho mais consistente, apesar de ruim. Coisa que não se confere na nova versão.



Aqui podemos observar um uso muito melhor da comunicação entre processador e memória devido a largura de banda aumentar notavelmente mais na versão otimizada, o que indica que mais dados são passados por segundo, otimizando o acesso a memória.

ADD UMA SUBCONCLUSÃO??

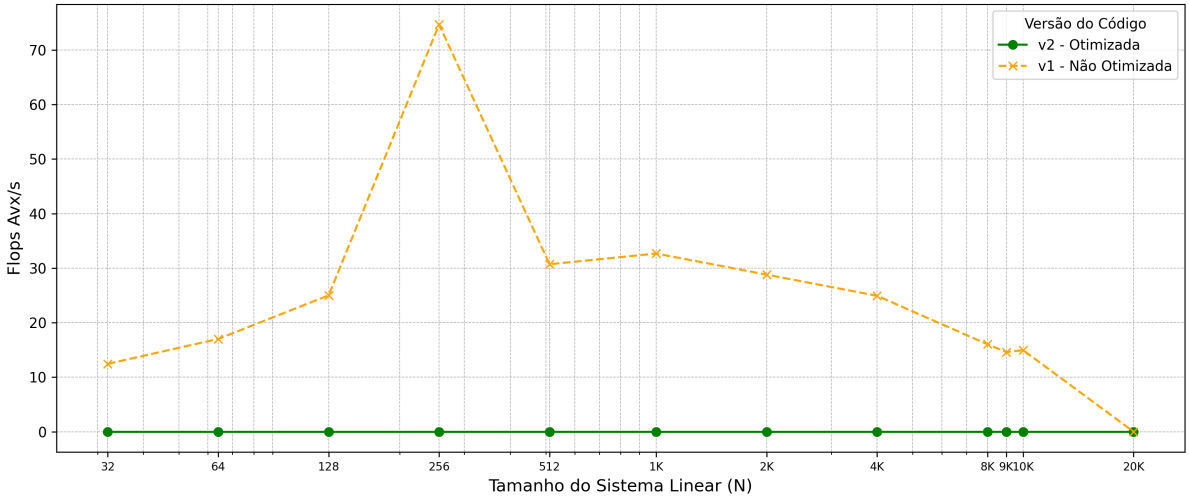
3.2 Cálculo do Resíduo



No gráfico acima percebelos uma melhora relativa entre a primeira versão (não otimizada) e a segunda (otimizada), que se mostra mais significativa em tamanhos maiores, por exemplo para N na casa dos milhares.

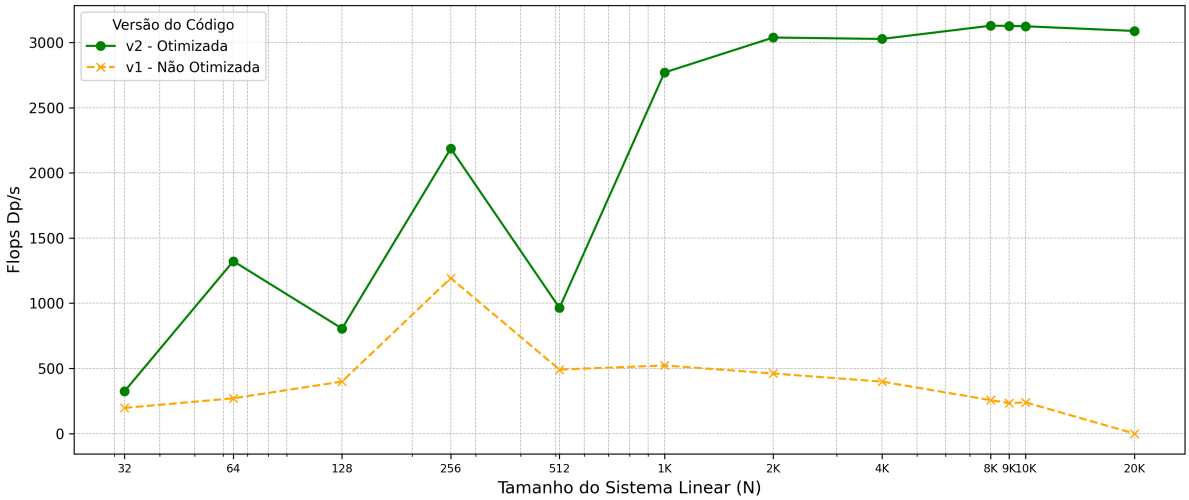
Tais melhoras podem ser justificadas pelos gráficos a seguir, bem como pelo uso do modelo **CSR**.

Flops Avx por Tamanho do SL - Resíduo



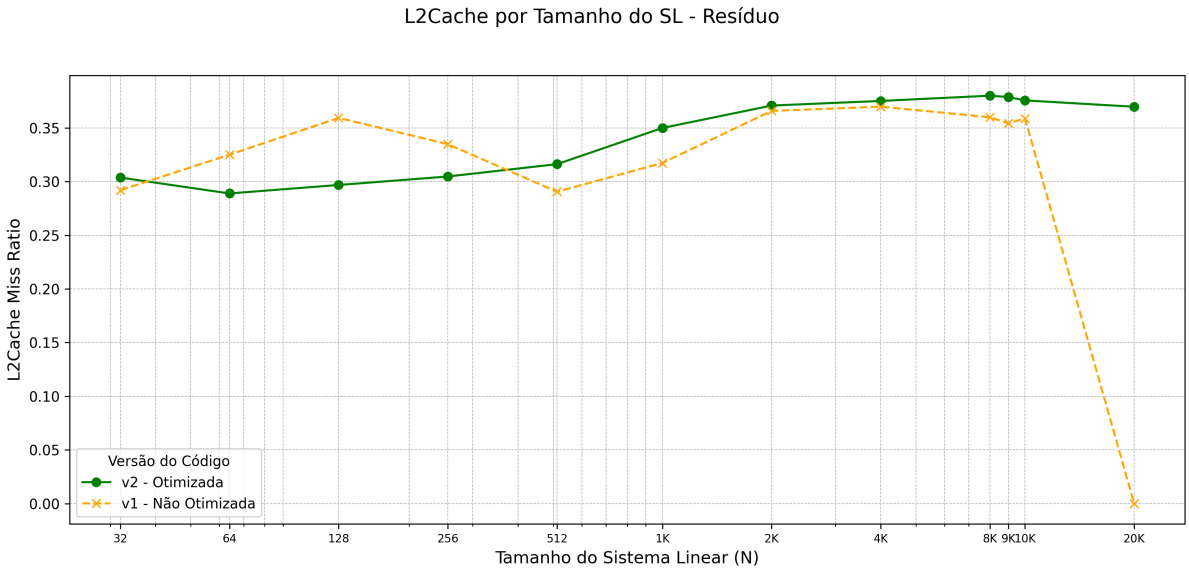
No que diz respeito ao uso de Flops **AVX** no cálculo do resíduo temos que na nova versão não houve o uso da extensão em nenhum momento, coisa que após analisarmos o código do programa e as métricas, consideramos ser devido ao uso do **CSR**, que acaba por minar o uso de instruções SIMD ao não ser possível desenrolar o laço e/ou identificar pontos de possível uso.

Flops Dp por Tamanho do SL - Resíduo

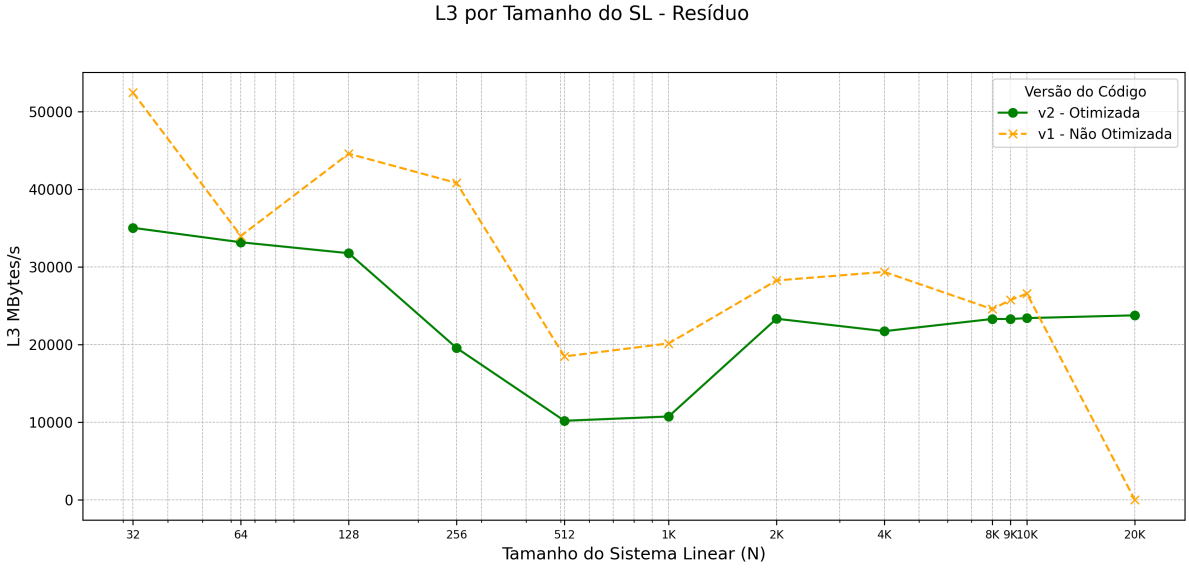


Ao contrário do que foi visto no último grafico, podemos perceber aqui

uma impressiva melhora, ao passo que realiza muitos mais cálculos em uma mesma quantidade de tempo, acelerando o programa.



Neste ponto é possivel notar que não houve melhora no Miss Ratio da cache L2, até mesmo representando uma pequena piora em parte dos casos.



A L3 bandwitch apresentou uma piora, pois menos dados estão sendo carregados por segundo, mesmo que seja uma pequena diferença em boa

parte dos casos.

4 Resultados e Análise de Desempenho

Os resultados demonstram que a combinação do armazenamento **CSR** com a vetorização **AVX** produziu uma redução significativa no tempo de execução total, especialmente em sistemas de alta dimensão e esparsidade.

A Tabela 1 resume o ganho de desempenho (speedup) alcançado para as operações críticas:

Tabela 1: Comparação de Tempo de Execução: Original vs. Otimizado (Tempo para N=20.000 em milisegundos)

Operação	**Tempo Original**	**Tempo Otimizado**	**Speedup**
Iteração do PCG	23.60	0.0097	2432.98x
Cálculo do Resíduo	0.87	0.17	5.11x

Vale ressaltar que no teste de N = 20.000 do 1º trabalho, o **LIKWID** não gerou nenhum dado correspondente a execução, portanto apenas a medida de tempo deve ser considerada, já que todas as outras são iguais a zero.

5 Estimativa de Desempenho

Baseado nas modificações implementadas na segunda versão e considerando uma máquina com 8GB de memória RAM disponível para seu programa e Sistemas com 7 (sete) diagonais, pode-se responder às perguntas dadas no enunciado do trabalho.

1. Qual o valor máximo estimado da ordem N do sistema linear que você seria capaz de resolver?

   Após breves discussões, concluímos que conseguiriamos resolver até um sistema de ordem 12. Apesar de demanadar muito tempo e esforço, parece uma métrica realizável em algumas horas.

2. Qual o ganho estimado de tempo para uma iteração do método de Gradiente Conjugado com Pré-condicionador de Jacobi (em função de N)?

   Estimamos que o ganho de desempenho seria algo dentre o intervalo de 8 a 16 vezes mais rápido. Pois, acreditavamos que isso já seria um ganho notável de desempenho.

3. Qual a estimativa para a quantidade de operações em ponto flutuante executadas em cada iteração do método de Gradiente Conjugado com Pré-condicionador de Jacobi (em função de N)?

   Estimamos que teriamos em torno de 100 Flops Dp por segundo no método de Gradiente Conjulgado, coisa que já seria melhor que no primeiro trabalho.

4. Compare suas estimativas acima com os resultados obtidos nos testes.

   Bom, apenas as perguntas '2.' e '3.' podem ser comparadas, visto que a primeira não trata sobre o programa/otimizações em si.

   Sobre o ganho estimado (2.); percebemos que a diferença foi muito maior, chegando a casos da segunda versão apresentar um tempo de execução 2458 vezes menor que a versão 1.

   Já sobre a quantidade de operações de ponto flutuante (3.); obtivemos 120 Flops Dp em média nos testes, bem como tivemos um expressivo aumento em Flops AVX.

6 Conclusão

Por fim, nota-se que as operações críticas do método dos Gradientes Conjugados foram otimizadas com sucesso. Utilizando métodos estruturais (como a matriz **CSR** para matrizes esparsas) e técnicas de paralelismo ao nível de instrução (como a **vetorização AVX**), alcançamos uma melhoria substancial no desempenho do programa. Conseguimos confirmar a execução otimizada pelo **AVX** em pelo menos uma das operações e, como resultado, **diminuímos drasticamente** o tempo de execução total do algoritmo, tornando o Método dos Gradientes Conjugados mais eficiente para a resolução de sistemas lineares de grande escala.

Referências

[1] CENAPAD-SP. *Engineering and Scientific Subroutine Library Guide and Reference: Subroutine Reference SDRSBC*, 2025.

[2] Ed F. D'Azevedo. Vectorized Sparse Matrix Multiply for Compressed Row Storage Format, 2025.

[3] Lei Mao. CSR Sparse Matrix Multiplication, 2019.

[4] Netlib. Parallel Sparse Matrix-Vector Multiplication, 2025.

[5] Wikipedia. Sparse matrix: Compressed sparse row (CSR, CRS or Yale format), 2025.