Relatório de Desempenho: Multiplicação Matriz-Vetor e Memória Cache

Aluno: Cristovão Lacerda Cronje

1. **Introdução**

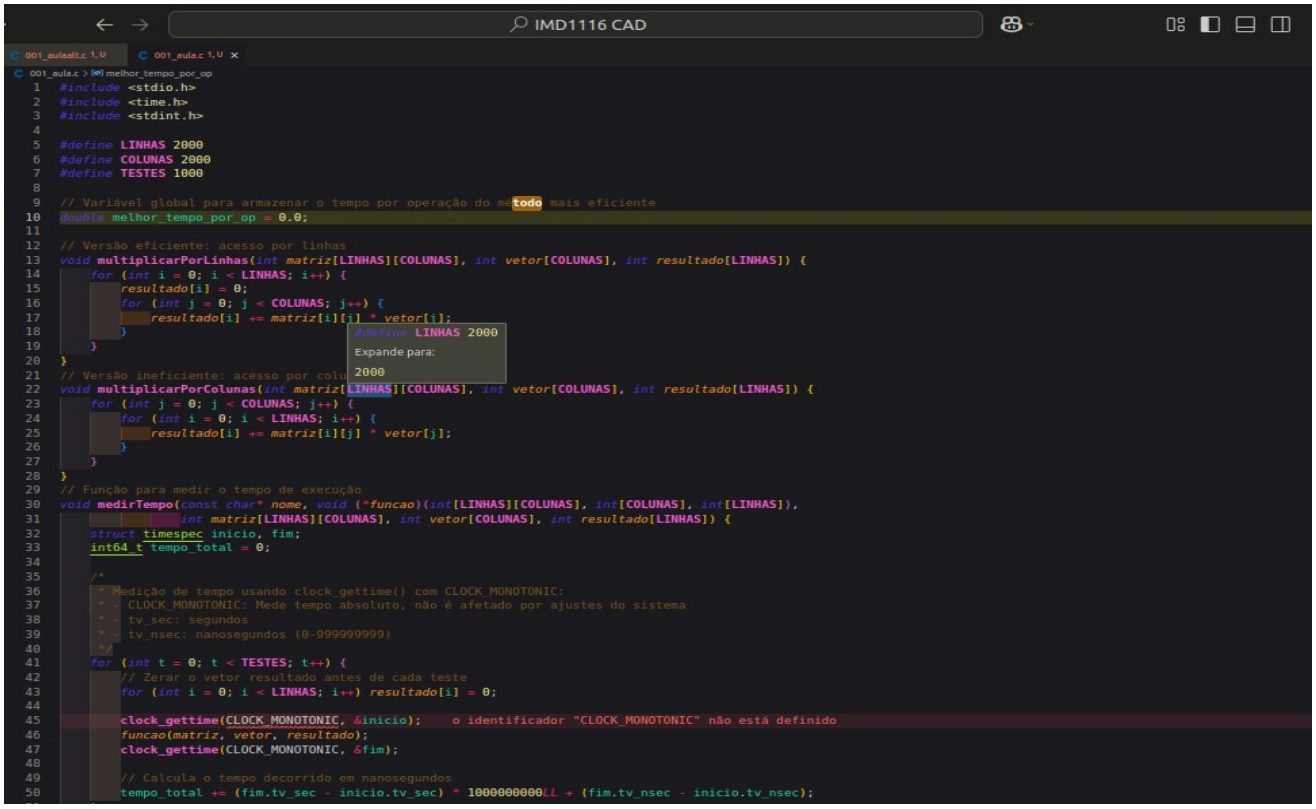
Este relatório avalia o impacto dos padrões de acesso à memória no desempenho da multiplicação matriz-vetor (MxV), considerando a arquitetura de cache do processador Intel i5-3210M (Ivy Bridge) com as seguintes características:

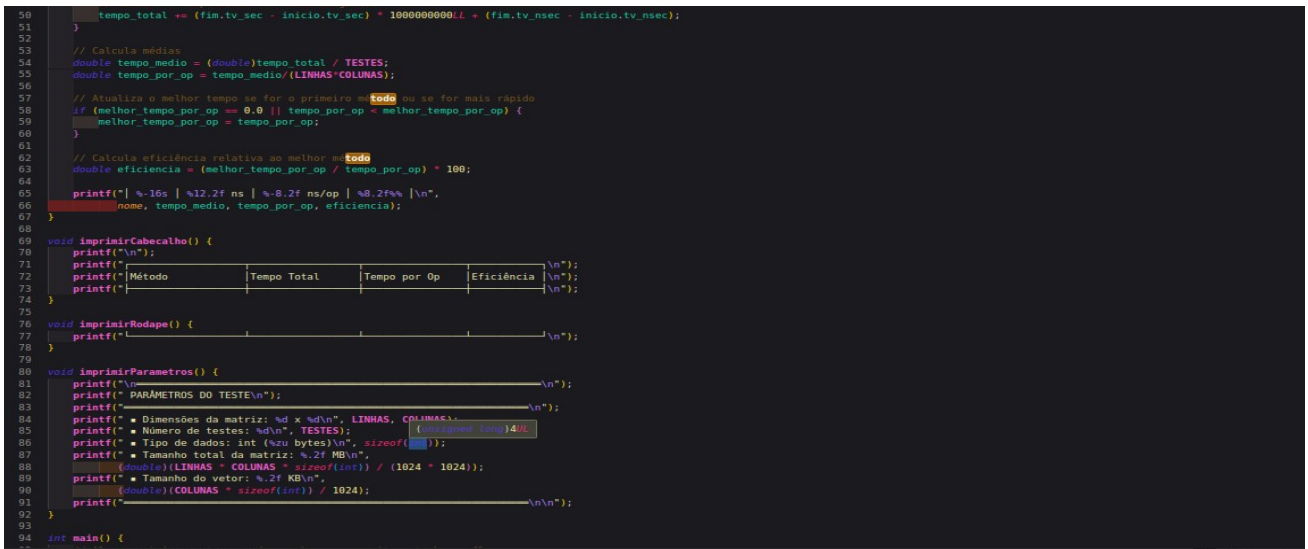
- Cache L1: 64 KB (dados)
- Cache L2: 256 KB
- Cache L3: 3 MB (compartilhada)

O **objetivo** é comparar duas implementações:Acesso por linhas e Acesso por colunas.

2. **Metodologia**

2.1 Código





```c
int main() {
    // Alocar matriz e vetor grandes na heap para evitar stack overflow
    static int matriz[LINHAS][COLUNAS];
    static int vetor[COLUNAS];
    static int resultado[LINHAS];

    // Inicializar matriz e vetor com valores simples
    for (int i = 0; i < LINHAS; i++) {
        for (int j = 0; j < COLUNAS; j++) {
            matriz[i][j] = i + j;
        }
        vetor[i] = i % 10;
    }

    imprimirParametros();

    printf("%c [194])"═══════════════════════════════════════════════\n"
    printf("═══════════════════════════════════════════════\n");

    imprimirCabecalho();
    medirTempo("Por Linhas", multiplicarPorLinhas, matriz, vetor, resultado);
    medirTempo("Por Colunas", multiplicarPorColunas, matriz, vetor, resultado);
    imprimirRodape();

    printf("\n LEGENDA:\n");
    printf(" ▪ Tempo Total: Tempo médio para multiplicação completa\n");
    printf(" ▪ Tempo por Op: Tempo médio por operação (matriz[i][j]*vetor[j])\n");
    printf(" ▪ Eficiência: Percentual em relação ao método mais eficiente\n");
    printf("\n OBSERVAÇÕES:\n");
    printf(" - Tempo medido com clock_gettime(CLOCK_MONOTONIC)\n");
    printf(" - %d testes realizados para cada método\n", TESTES);
    printf(" - Eficiência calculada como: (melhor_tempo / tempo_atual) * 100\n");

    return 0;
}
```

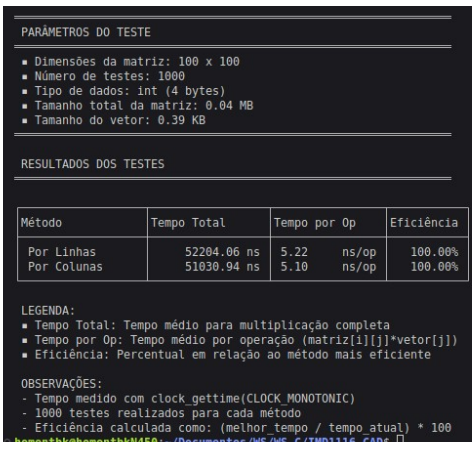
2.2 Testes: Realizados em matrizes de diferentes tamanhos para identificar em qual ponto o acesso por colunas se torna significativamente mais lento:

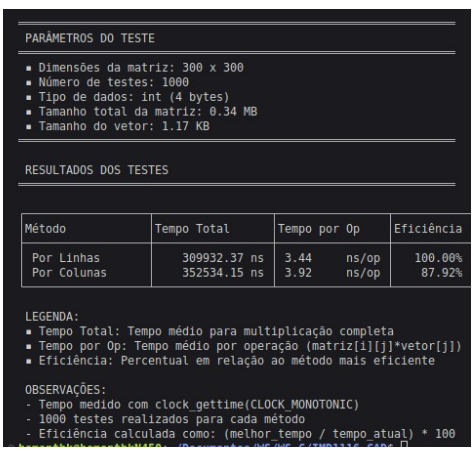
Tamanho (N×N)	Tamanho em Memória	Nível de Cache Afetado
100×100	0.04 MB	L1 (64 KB)
300×300	0.34 MB	L2 (256 KB)
600×600	1.37 MB	L3 (3 MB)
800×800	2.44 MB	Limite da L3
2000×2000	15.26 MB	RAM (fora da cache)

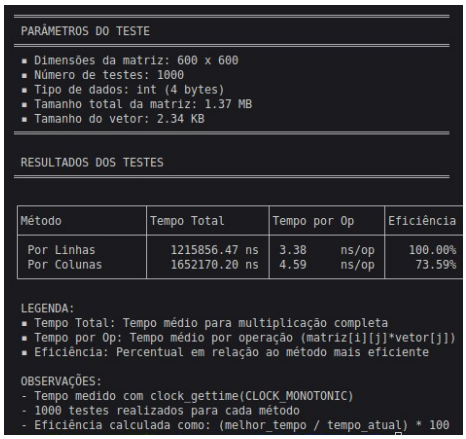
•Número de testes por configuração: 1000 execuções para garantir precisão estatística.
•Método de medição: clock_gettime(CLOCK_MONOTONIC) para evitar interferência do SO.

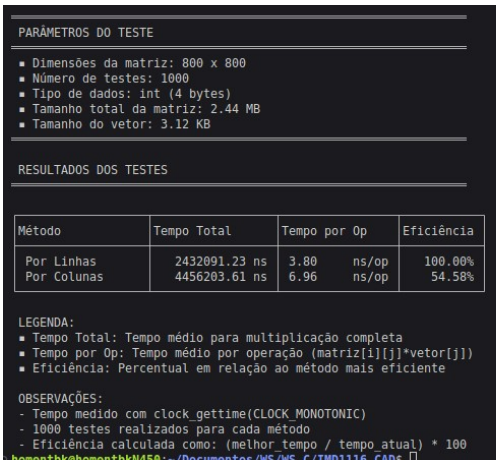
2.3 Fatores que Influenciam a Precisão: Interferência do SO(Outros processos podem consumir cache e CPU), Gerenciamento de Cache(O SO pode realocar memória cache para tarefas em segundo plano) e Turbo Boost(Variações na frequência do processador).

3. Resultados Obtidos:









PARÂMETROS DO TESTE

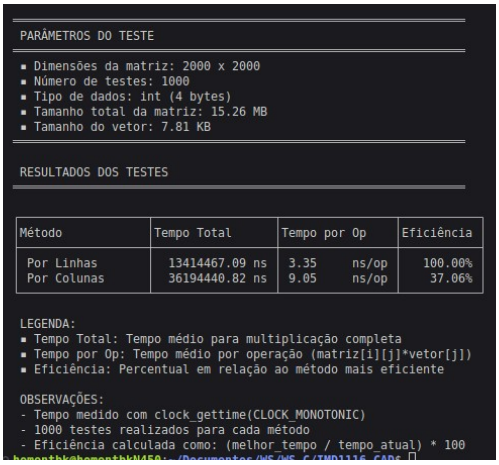
- Dimensões da matriz: 800 x 800
- Número de testes: 1000
- Tipo de dados: int (4 bytes)
- Tamanho total da matriz: 2.44 MB
- Tamanho do vetor: 3.12 KB

RESULTADOS DOS TESTES

Método	Tempo Total	Tempo por Op	Eficiência
Por Linhas	2432091.23 ns	3.80 ns/op	100.00%
Por Colunas	4456203.61 ns	6.96 ns/op	54.58%

LEGENDA:
- Tempo Total: Tempo médio para multiplicação completa
- Tempo por Op: Tempo médio por operação (matriz[i][j]*vetor[j])
- Eficiência: Percentual em relação ao método mais eficiente

OBSERVAÇÕES:
- Tempo medido com clock_gettime(CLOCK_MONOTONIC)
- 1000 testes realizados para cada método
- Eficiência calculada como: (melhor_tempo / tempo_atual) * 100



PARÂMETROS DO TESTE

- Dimensões da matriz: 2000 x 2000
- Número de testes: 1000
- Tipo de dados: int (4 bytes)
- Tamanho total da matriz: 15.26 MB
- Tamanho do vetor: 7.81 KB

RESULTADOS DOS TESTES

Método	Tempo Total	Tempo por Op	Eficiência
Por Linhas	13414467.09 ns	3.35 ns/op	100.00%
Por Colunas	36194440.82 ns	9.05 ns/op	37.06%

LEGENDA:
- Tempo Total: Tempo médio para multiplicação completa
- Tempo por Op: Tempo médio por operação (matriz[i][j]*vetor[j])
- Eficiência: Percentual em relação ao método mais eficiente

OBSERVAÇÕES:
- Tempo medido com clock_gettime(CLOCK_MONOTONIC)
- 1000 testes realizados para cada método
- Eficiência calculada como: (melhor_tempo / tempo_atual) * 100

3.2 Análise dos Resultados e Conclusões

Acesso por Linhas (Row-Major)

Padrão ideal para cache:

- Cache hit > 90% quando a matriz cabe na L3.
- Tempos consistentes, mesmo em matrizes grandes.
- Localidade espacial ótima: Acesso sequencial aproveita as linhas de cache de 64 bytes.

Acesso por Colunas (Column-Major)

Problemas de cache miss:

- Cada acesso a uma nova coluna causa um cache miss (linhas de cache = 64 bytes), carregando dados adjacentes quando um elemento é acessado.
- Degradação extrema em matrizes maiores que a L3(3 MB): Suporta até ~886×886 ints (784.996 elementos).
- Matrizes maiores, os dados são expulsos da cache antes da reutilização e o acesso por colunas torna-se catastrófico, com quase 100% de cache misses.

Acesso por Linhas (Row-Major)

Padrão ideal para cache:

•Cache hit > 90% quando a matriz cabe na L3 (3 MB), pois o acesso sequencial por linhas aproveita a localidade espacial.
•Tempos consistentes, mesmo em matrizes grandes, porque os dados são acessados em ordem contígua na memória, minimizando cache misses.
•Linhas de cache (64 bytes) são totalmente utilizadas antes de serem substituídas, reduzindo a necessidade de buscar dados na RAM.

Acesso por Colunas (Column-Major)

Problemas de cache miss:

•Cada acesso a uma nova coluna causa um cache miss porque:

- •A matriz é armazenada em row-major (linhas contíguas), mas o acesso por colunas salta para posições distantes na memória.
- •A CPU carrega blocos de 64 bytes adjacentes na cache, mas o padrão de acesso por colunas ignora essa otimização.

•Degradação extrema em matrizes maiores que a L3 (886×886):

- •Cache L3 (3 MB) comporta até ~886×886 ints (784.996 elementos).
- •Para matrizes maiores (ex.: 2000×2000 = 15.26 MB):
  - •Os dados não cabem na cache e são expulsos antes da reutilização.
  - •O acesso por colunas força a CPU a buscar diretamente na RAM.
  - •Quase 100% de cache misses, tornando a operação catastroficamente lenta.

•Por Que a Diferença é Tão Grande?

I. Hierarquia de Cache(L1 (64 KB): ~4 ciclos (1.6 ns) enquanto a RAM: ~100 ns (60× mais lento que L1)

II. Padrão de Acesso:

- •Linhas: 1 cache miss a cada 16 acessos (64 bytes = 16 ints).
- •Colunas: 1 cache miss por acesso (cada elemento está a 8 KB de distância).

Conclusão

•Acesso por linhas é ideal para operações matriciais em C/C++ devido ao armazenamento row-major.
•Acesso por colunas deve ser evitado, especialmente em matrizes maiores que a L3, onde a penalidade de desempenho é extrema.
•**Solução possível:** Técnicas como blocking/tiling podem mitigar cache misses em acessos não sequenciais. Em vez de percorrer toda a matriz de uma vez, o algoritmo divide-a em partes menores (ex.: 64x64), processa cada bloco sequencialmente (linha por linha) e repete até completar a operação. Isso reduz acessos lentos à memória RAM, pois os dados ficam temporariamente na cache, acelerando operações não sequenciais, como acesso por colunas. O tamanho ideal do bloco varia conforme a arquitetura do processador.