
# Relatório da Tarefa 01

*O gargalo de von Neumman e a memória cache*

20210093987 - Francisca Paula de Souza Braz
Data da entrega: 01/04/2025
Disciplina: DCA3703 - PROGRAMAÇÃO PARALELA - T01 (2025.1)

# Introdução

A eficiência de programas que manipulam grandes quantidades de dados está diretamente relacionada à forma como a memória é acessada e utilizada. Na multiplicação de matrizes, um dos desafios é otimizar o acesso à memória cache para reduzir a latência e melhorar o desempenho. Esse problema está fortemente ligado ao **gargalo de von Neumann**, que limita a taxa de transferência de dados entre o processador e a memória, tornando essencial a compreensão dos padrões de acesso para otimizar a execução de algoritmos.

Dois conceitos-chave são a localidade temporal (reuso de dados recentes) e espacial (acesso a dados próximos na memória). O armazenamento de matrizes também influencia o desempenho em C que usa o (row-major), elementos de uma linha são sequenciais, enquanto em outras linguagens (como Fortran, column-major), colunas são armazenadas sequencialmente. Essa organização determina qual padrão de acesso aproveita melhor a cache, influenciando diretamente a eficiência.

Esta atividade implementará duas versões de multiplicação matriz-vetor em C: linha a linha vs. coluna a coluna, com medição de tempo para diferentes tamanhos de matriz. O objetivo é avaliar o impacto de padrões de acesso na performance, identificando quando os tempos divergem significativamente e como isso está relacionado ao uso da cache. A análise revelará como a organização de dados na memória influencia a eficiência.

# Metodologia

A metodologia adotada para a análise do desempenho da multiplicação de matriz por vetor (MxV) foi dividida em três etapas: **implementação em C**, **medição de tempo** e **testes realizados**. O objetivo foi comparar duas abordagens distintas de varredura da matriz (por linhas e por colunas) e avaliar o impacto do acesso à memória na eficiência do processamento.

A implementação consistiu na construção de duas versões do algoritmo de multiplicação de matriz por vetor.

```c
void multiplicacaoLinha(double** matriz, double* vetor, double* resultado, int N) {
    for (int i = 0; i < N; i++) {
        resultado[i] = 0.0;
        for (int j = 0; j < N; j++) {
            resultado[i] += matriz[i][j] * vetor[j];
        }
    }
}
```

**Figura 01 :** Percorrendo a matriz por linhas

```c
// Multiplicação por colunas
void multiplicacaoColuna(double** matriz, double* vetor, double* resultado, int N) {
    for (int i = 0; i < N; i++) {
        resultado[i] = 0.0;
    }
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < N; i++) {
            resultado[i] += matriz[i][j] * vetor[j];
        }
    }
}
```

**Figura 02:** Percorrendo a matriz por colunas

A principal diferença entre essas abordagens está na ordem de acesso aos elementos da matriz. Enquanto a primeira versão segue o row-major order que favor os acessos consecutivos na memória cache, a segunda versão acessa os elementos em ordem de coluna, o que pode reduzir a eficiência do cache e aumentar o tempo de execução.

Para obter medições precisas do tempo de execução de cada abordagem, utilizou-se a função `clock_gettime()` da biblioteca `<time.h>`. Essa função permite um controle mais preciso do tempo gasto na execução do algoritmo. O tempo foi medido da seguinte forma: registrando o **tempo** da execução do algoritmo após a conclusão da execução para cada interação e para diferentes tamanhos de matrizes.

Foi realizado diversos testes com variações de tamanhos maiores que as exemplificadas, mas o tempo de processamento passou a ser muito demorado para valores acima de 10.000 e para a atividade optou-se pelos tamanhos abaixo:

```c
int tamanhos[] = {1000, 2000, 3000, 5000, 1000};
```

# Resultados

Os resultados obtidos estão apresentados na tabela abaixo:

| Tamanho da Matriz (N) | Tempo - Linha (s) | Tempo - Coluna (s) |
|---|---|---|
| 1000 | 0.004508 | 0.007280 |
| 2000 | 0.011214 | 0.022581 |
| 3000 | 0.026943 | 0.045741 |
| 5000 | 0.075300 | 0.159375 |
| 10000 | 0.298434 | 1.320533 |

**Tabela 01: Registro dos resultados**

Os tempos de execução mostram que, à medida que o tamanho da matriz aumenta, a diferença entre os tempos das duas abordagens se torna mais significativa, como podemos ver no gráfico abaixo:

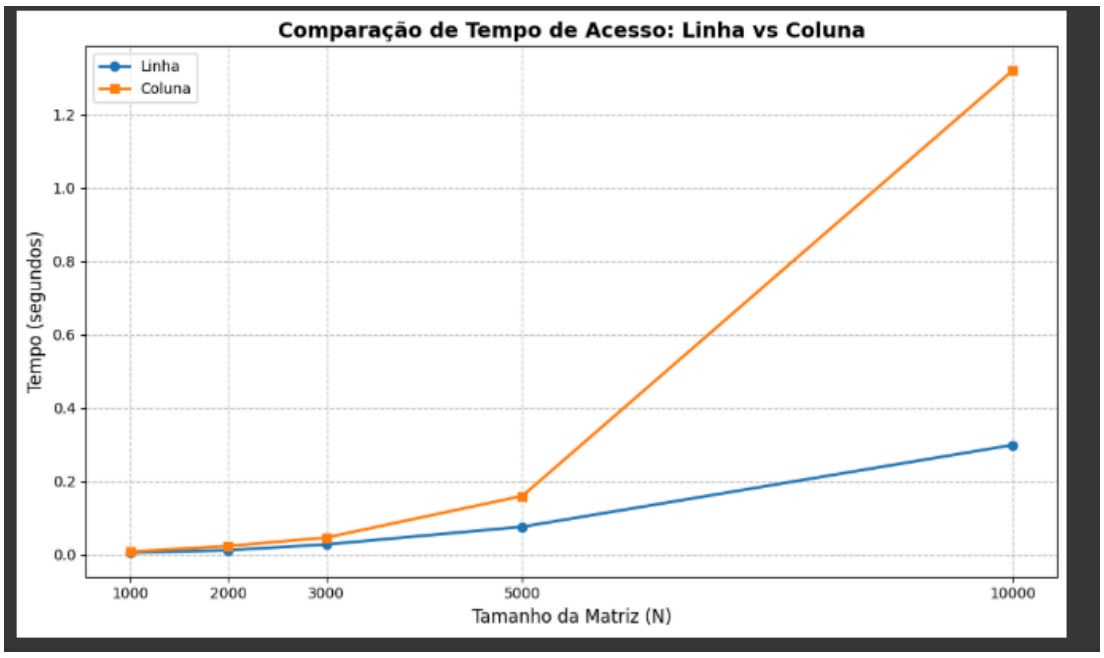


**Figura 03: Gráfico de comparação dos tempos**

Para N = 1000, o acesso por colunas já é 61,5% mais lento que o acesso por linhas, evidenciando um impacto significativo mesmo em matrizes menores. Essa diferença se

intensifica conforme o tamanho da matriz aumenta, para N = 3000, o acesso por colunas leva 69,8% mais tempo que o por linhas, confirmando a tendência de degradação de desempenho. No caso de N = 10000, a diferença atinge 342,4%, ou seja, o tempo de acesso por colunas é 3,4 vezes maior que o por linhas, um salto crítico que revela a natureza não linear do problema.

## Conclusão

A discrepância observada entre os tempos de execução está intrinsecamente relacionada ao padrão de acesso à memória cache. Em C, onde matrizes são armazenadas em row-major order, o acesso sequencial por linhas maximiza a localidade espacial onde os elementos adjacentes na memória são carregados em blocos (cache lines), minimizando cache misses e aproveitando a hierarquia de memória de forma eficiente. Cada operação de carga traz dados relevantes para iterações subsequentes, reduzindo a latência de acesso.

Por outro lado, o acesso por colunas rompe essa localidade, pois os elementos de uma coluna estão dispersos na memória (separados pelo tamanho da linha). Isso força a CPU a realizar acessos não contíguos, gerando cache misses frequentes e desperdiçando a capacidade da cache. A localidade temporal, que depende da reutilização de dados recentes, não é o fator crítico aqui, já que ambas as abordagens seguem padrões de acesso sem reuso imediato.

Com isso podemos concluir que a eficiência de algoritmos matriciais depende criticamente de alinhar o padrão de acesso à organização da memória.