UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
IDCA3703 - PROGRAMAÇÃO PARALELA

COMPARAÇÃO ENTRE MULTIPLICAÇÃO DE MATRIZ POR VETOR COM ACESSO POR LINHA E POR COLUNA, E O IMPACTO NO TEMPO DE EXECUÇÃO E CACHE

RELATÓRIO DE EXECUÇÃO

ERNANE FERREIRA ROCHA JUNIOR

NATAL/RN, 26 DE MARÇO DE 2025

SUMÁRIO

1. INTRODUÇÃO

Este estudo tem como objetivo comparar duas abordagens de multiplicação de matriz por vetor (MxV), uma com acesso por linha externa (row-major) e outra com acesso por coluna externa (column-major). Para isso, são realizados testes com diferentes tamanhos de matriz e medidos os tempos de execução de cada abordagem. Além disso, é discutido o impacto dessas abordagens no uso de memória cache (L1 e L2) do processador, e o efeito da alocação dinâmica (heap) versus estática (stack).

2. METODOLOGIA

O programa desenvolvido implementa a multiplicação de uma matriz de tamanho SIZE x SIZE por um vetor de tamanho SIZE, utilizando duas abordagens de acesso à matriz:

- **Acesso por linhas (*row-major*)**: A iteração é realizada de forma que a linha externa é a iteração mais externa.

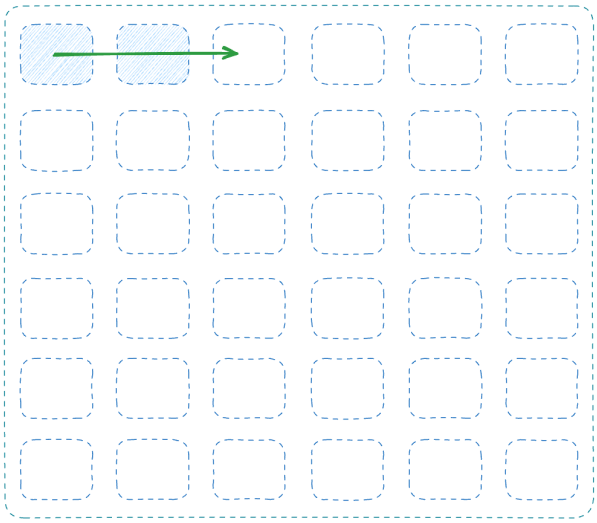

Figure 1: Representação da iteração no acesso por linhas (*row-major*):

- **Acesso por colunas (*column-major*)**: A iteração é realizada de forma que a coluna externa é a iteração mais externa.

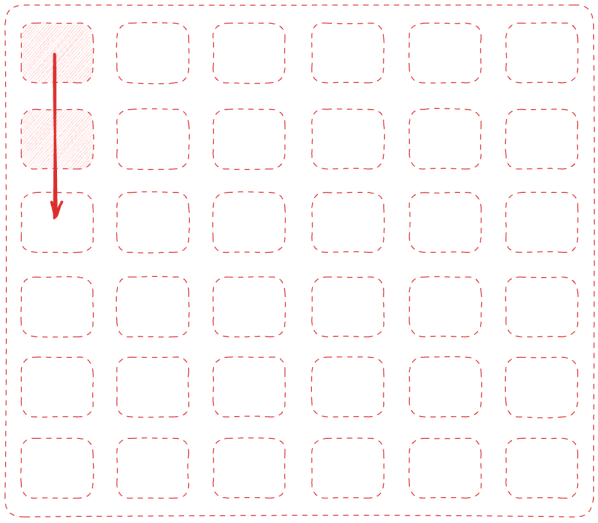

Figure 2: Representação da iteração no acesso por colunas (*column-major*):

O tempo de execução de cada abordagem é medido usando a função *gettimeofday()* para obter a precisão necessária. Além disso, os testes são realizados com diferentes tamanhos de matriz, variando de SIZE = 100 a *SIZE = 20000*, tanto com alocação estática (na stack) quanto dinâmica (na heap).

3. RESULTADOS

Abaixo estão as tabelas com os tempos de execução para os diferentes tamanhos de matriz e as comparações entre row-major e *column-major*, além das diferenças de alocação (*stack vs. heap*).

Tamanho da Matriz (*SIZE*)	**Execução (*Row-Major*)**	**Execução (*Column-Major*)**	**Observações**
100	0.000048 segundos	0.000050 segundos	Negligível diferença
250	0.000146 segundos	0.000360 segundos	Diferença começando a aparecer
500	0.001095 segundos	0.000722 segundos	Maior tempo no acesso por linha
750	0.001583 segundos	0.001794 segundos	Diferença crescente
1000	0.003358 segundos	0.004163 segundos	Diferença mais notável
1444	0.006741 segundos	0.007982 segundos	Maior diferença de tempo
1445	Segmentation Fault	Segmentation Fault	Limitação de stack

Table 1: Execuções com Alocação Estática (Stack)

Tamanho da Matriz (*SIZE*)	**Execução (*Row-Major*)**	**Execução (*Column-Major*)**	**Observações**
100	0.000064 segundos	0.000064 segundos	Sem diferenças
250	0.000249 segundos	0.000255 segundos	Diferença mínima
500	0.001280 segundos	0.001281 segundos	Diferença muito pequena
750	0.002514 segundos	0.002581 segundos	Diferença crescente
1000	0.003866 segundos	0.004006 segundos	Diferença mais clara
10000	0.265163 segundos	0.731580 segundos	Maior diferença de tempo
20000	1.503544 segundos	4.290163 segundos	Diferença considerável de tempo

Table 2: Execuções com Alocação Dinâmica (Heap)

3.1. USO DA CACHE L1/L2

O desempenho das duas abordagens (*row-major vs*. *column-major*) está diretamente relacionado ao uso eficiente da memória cache. O processador *M2* possui cache *L1* de *128 KB* para dados e cache *L2* de *16 MB* para o desempenho máximo. A eficiência no uso da memória cache depende fortemente do padrão de acesso aos dados.

No acesso por linha (*Row-Major*), os elementos de uma linha são contíguos na memória, o que significa que a *CPU* pode manter os dados em cache de nível 1 (*L1*) ao percorrer a linha inteira. Isso aproveita bem o cache *L1*, especialmente para matrizes pequenas a médias. Já no acesso por coluna (*Column-Major*), os elementos de uma coluna não são contíguos na memória, o que resulta em mais *misses* (erros) de cache, já que a *CPU* precisa acessar diferentes linhas consecutivamente. Esse padrão é menos eficiente no uso da cache.

3.2. IMPACTO DO CACHE L1 E L2

A *Cache L1* é rápida, mas muito pequeno. Para matrizes grandes, especialmente acima de *SIZE=500*, a quantidade de dados que precisa ser carregada excede o tamanho do cache *L1*, fazendo com que a *CPU* tenha que buscar dados do cache *L2* ou até da memória principal. O cache *L2* é compartilhado entre os núcleos de alto desempenho e tem maior capacidade, o que ajuda a manter os dados mais "quentes" por mais tempo. No entanto, o acesso desordenado à memória, como no caso do acesso por coluna, não favorece a reutilização dos dados cacheados, resultando em maior tempo de execução.

3.3. DIFERENÇA DE TEMPO DE EXECUÇÃO

Para tamanhos menores de matriz, os tempos de execução para *row-major* e *column-major* são muito próximos, pois o cache *L1* é suficiente para armazenar as linhas ou colunas da matriz inteira. A diferença entre as abordagens é quase irrelevante. A partir de um certo ponto, a matriz não cabe mais no cache *L1*, e a diferença no tempo de execução se torna mais pronunciada. O *row-major* ainda se beneficia do cache *L1*, enquanto o column-major sofre com mais falhas de cache. Para matrizes muito grandes, como *SIZE=20000*, o *row-major* mostra tempos significativamente mais baixos. O *column-major* sofre uma penalização muito maior, já que os dados não são acessados de forma contígua e o cache *L2* é sobrecarregado, aumentando o tempo de execução.

3.4. LIMITAÇÃO DE STACK

A alocação de memória na stack é limitada pelo tamanho máximo da pilha do sistema, o que restringe o valor de *SIZE* a um máximo de 1444. Qualquer valor superior a isso, como *SIZE=1445*, resulta em um erro de *segmentation fault*, pois o tamanho da matriz excede a capacidade da stack.

3.5. SISTEMA

O desempenho do código de multiplicação de matriz por vetor e os tempos de execução observados estão diretamente relacionados ao sistema utilizado. Isso ocorre devido à interação entre o código e o hardware do computador, especialmente no que diz respeito ao cache da *CPU*, que é

essencial para operações de memória. Quando o código acessa a memória de forma ineficiente, o cache não é utilizado de maneira ideal, resultando em aumento nos tempos de execução.

No exemplo em questão, o sistema utilizado é um *MacBook Air* com chip *Apple M2*, que possui uma arquitetura de cache sofisticada. O cache *L1* (nível 1) e *L2* (nível 2) têm características específicas que impactam o desempenho, principalmente ao lidar com matrizes grandes.

```
$ nproc
> 8

$ sysctl -a | grep cache
# ...
> hw.perflevel1.l1icachesize: 131072
> hw.perflevel1.l1dcachesize: 65536
> hw.perflevel1.l2cachesize: 4194304
> hw.perflevel0.l1icachesize: 196608
> hw.perflevel0.l1dcachesize: 131072
> hw.perflevel0.l2cachesize: 16777216
# ...

$ lscpu | grep -i cache # Para linux
# ...
> L1d cache: ...
> L1i cache: ...
> L2 cache:  ...
> L3 cache:  ...
# ...
```

Ou seja,

- Cache *L1* (nível 1):
  - Instruções (*perflevel0* - alto desempenho): 192 *KB* (196608 *bytes*)
  - Dados (*perflevel0* - alto desempenho): 128 *KB* (131072 *bytes*)
  - Instruções (*perflevel1* - eficiência): 128 *KB* (131072 *bytes*)
  - Dados (*perflevel1* - eficiência): 64 *KB* (65536 *bytes*)
- Cache *L2* (nível 2, compartilhado):
  - *Perflevel0* (alto desempenho): 16 *MB* (16777216 *bytes*)
  - *Perflevel1* (eficiência): 4 *MB* (4194304 *bytes*)

Essas especificações indicam que o *M2 Air* tem um cache *L1* de *192 KB* para instruções e *128 KB* para dados por núcleo, com 8 núcleos de alto desempenho e eficiência. O cache *L2* compartilhado é de *16 MB* para os núcleos de alto desempenho e *4 MB* para os núcleos de eficiência. Com isso, o comportamento de tempo de execução e o uso de memória cache são altamente dependentes do sistema.

No caso do *MacBook Air M2*, a capacidade limitada de cache *L1* para dados e a presença de cache *L2* afetam significativamente o desempenho, especialmente com matrizes grandes. O código executado em memória *heap* pode acomodar matrizes muito maiores sem risco de falhas de segmentação, mas o desempenho depende tanto do tamanho do cache disponível quanto do padrão de acesso à memória utilizado, esse nos dois casos. A opção de acesso por coluna acaba sendo mais lenta nesse cenário, pois resulta em mais falhas de cache.

Esse comportamento de tempo de execução e cache ocorre em outros sistemas também, como em Linux. A interação do código com o cache e as limitações do sistema sempre terão impacto no desempenho independente das otimizações do sistema operacional para gerenciamento de cache ser ou não mais personalizada para o hardware utilizado.

4. CONCLUSÃO

A escolha entre os padrões de acesso por linha e por coluna exerce um impacto significativo no desempenho do programa, com o acesso por linha se destacando por sua maior eficiência, especialmente em matrizes grandes, devido ao melhor aproveitamento do cache *L1*. Para matrizes de tamanho pequeno ou médio, no entanto, essa diferença se torna praticamente irrelevante. O acesso por coluna, embora possa ser vantajoso em casos específicos, apresenta um desempenho inferior, principalmente devido à maior ocorrência de falhas de cache, tornando-se menos eficiente à medida que o tamanho da matriz aumenta. A alocação dinâmica na heap oferece uma solução interessante para trabalhar com matrizes de grandes dimensões, superando as limitações da stack, que pode resultar em erros de segmentação quando utilizada para alocar matrizes maiores que determinados tamanhos. O impacto no tempo de execução, relacionado ao uso do cache pelo processador, independe dessa escolha, mas a alocação dinâmica possibilita o manuseio de matrizes e vetores de tamanhos maiores, o que, dependendo do algoritmo escolhido, pode influenciar diretamente o desempenho. Dessa forma, a decisão entre stack e heap, juntamente com o padrão de acesso, deve ser cuidadosamente considerada em função do tamanho dos dados, da capacidade do sistema e dos requisitos de desempenho da aplicação.

5. ANEXOS

- Repositório no Github com o programa desenvolvido: https://github.com/ErnaneJ/parallel-programming-dca3703