Universidade Federal do Rio Grande do Norte - UFRN
Departamento de Engenharia da Computação e Automação - DCA
DCA3703 - Programação Paralela

Tarefa 01: Memória Cache

Discente: Quelita Míriam
Docente: Samuel Xavier de Souza

I. INTRODUÇÃO

Uma limitação conhecida como "gargalo de *von Neumann*", é parte do modelo de arquitetura de computadores proposto por John von Neumann em 1945, que define que a CPU e a memória compartilham um mesmo barramento para a troca de dados e instruções, se referindo ao fato de que a velocidade da CPU é significativamente maior do que a capacidade de transferência de dados da memória principal, logo, a CPU frequentemente aguarda por dados, reduzindo a eficiência do processamento.

A memória cache é um componente essencial para mitigar o gargalo de von Neumann. Ela consiste em uma memória de acesso rápido localizada entre a CPU e a memória RAM. Seu objetivo é armazenar temporariamente dados frequentemente acessados, reduzindo a latência de acesso à memória principal. O funcionamento da memória cache se baseia no princípio da localidade, que pode ser dividido em localidade temporal e localidade espacial. A localidade temporal ocorre quando um dado ou instrução foi acessado recentemente e há uma alta probabilidade de ser acessado novamente em um futuro próximo, como em loops de repetição que utilizam a mesma variável diversas vezes. Já a localidade espacial refere-se ao fato de que, ao acessar um dado na memória, há uma tendência de que outros dados armazenados próximos a ele também sejam acessados em sequência, como acontece em estruturas de dados contíguas, como vetores e matrizes.

Para explorar o impacto da memória cache, foi realizada uma experiência comparando duas versões de um algoritmo de multiplicação de matriz por vetor (MxV) em C. Na primeira, o acesso à matriz ocorre por linhas (linha externa, coluna interna – *row-major*), enquanto na segunda, o acesso ocorre por colunas (coluna externa, linha interna – *column-major).*

Ademais, a tarefa impacta em discutir sobre a diferença entre tempo de execução e tempo real. Entende-se por tempo de execução a quantidade de tempo que um programa leva para rodar, contando só quando a CPU realmente trabalha nele, sem considerar pausas ou esperas. Já o tempo real é sobre sistemas que precisam responder rápido e dentro de um prazo fixo, como freios de um carro. O primeiro depende da velocidade do computador, enquanto o segundo precisa de regras que garantam respostas rápidas.

II. METODOLOGIA

Para a presente tarefa o desenvolvimento do código[1] seguiu-se entre as duas abordagens em análise, além de abordar essa estrutura em matriz estática e alocada dinamicamente:

[1] **QUELITA2.** Programação Paralela. Repositório GitHub, 2025. Disponível em: <https://github.com/quelita2/programacao-paralela/tree/main/topico01/src>. Acesso em: 01 abr. 2025.

- Acesso por linhas (row-major): a matriz é armazenada na memória em ordem de linhas (*row-major)*, o que significa que os elementos de uma linha são armazenados de forma contígua na memória antes de passar para a próxima linha.
- Acesso por colunas (column-major): a matriz é armazenada na memória em ordem de colunas (column-major), o que significa o acesso à matriz em saltos na memória, reduzindo a eficiência do cache e aumentando o tempo de execução devido ao maior número de falhas de cache.

A tarefa relatada realiza medição entre as versões com a função "*gettimeofday()*", usada para obter o tempo real (ou wall-clock time) com uma precisão em microssegundos.

III. RESULTADOS E DISCUSSÕES

O código desenvolvido foi implementado para uma matriz alocada dinamicamente e uma estática. Abaixo, encontram-se os resultados obtidos.

Para a matriz estática:

Tamanho	Row-major	Column-major
1000	0.003475 s	0.005320 s
2000	0.018033 s	0.034001 s
5000	0.084274 s	0.312534 s
10000	0.338950 s	1.631389 s

Para a matriz alocada dinamicamente:

Tamanho	Row-major	Column-major
1000	0.003426 s	0.009768 s
2000	0.015898 s	0.036153 s
5000	0.089753 s	0.280027 s
10000	0.698242 s	3.577281 s

Na versão dinâmica, o acesso por linhas é sempre mais rápido que o acesso por colunas porque a matriz é armazenada em sequência na memória, otimizando a localidade espacial e reduzindo falhas de cache. Conforme o tamanho cresce, a penalidade do acesso por colunas aumenta drasticamente devido ao acesso disperso à memória, resultando em tempos significativamente maiores. Já na versão estática, os tempos iniciais são mais próximos porque a matriz está totalmente alocada na *stack*, onde o acesso é mais eficiente. No entanto, à medida que o tamanho cresce, a penalidade do acesso por colunas também se torna evidente, embora os tempos sejam um pouco menores que na versão

dinâmica para os maiores tamanhos devido à organização da memória stack ser mais eficiente que a *heap* em termos de acessos curtos.

IV. CONCLUSÃO

Por meio deste relatório foi possível observar que, a partir de um determinado tamanho, o tempo de execução da matriz por colunas tornou-se significativamente maior do que a versão por linhas. Isso ocorre porque a organização padrão de memória em C é baseada no armazenamento por linhas, tornando o acesso por colunas menos eficiente. Quando os acessos à memória seguem um padrão não contínuo, há um aumento de uso de cache, obrigando a CPU a buscar dados na memória principal, o que causa maior latência.

Sendo assim, o acesso sequencial por linhas é mais eficiente do que o acesso por colunas, pois reduz a quantidade de leituras desnecessárias da memória principal. Esse experimento exemplifica como padrões de acesso à memória podem impactar significativamente o desempenho das aplicações computacionais, onde a eficiência do código pode ser otimizada considerando a localidade da memória cache.

V. REFERÊNCIAS BIBLIOGRÁFICAS

CANALTECH. O que é o gargalo de von Neumann? Disponível em: <https://canaltech.com.br/hardware/o-que-e-o-gargalo-de-von-neumann/>. Acesso em: 01 abril 2025.