

**Performance evaluation of a single core and multi core**

CDP Project

**Turma 4 – Grupo 14**

Gaspar Manuel Ferrás Faria (up202108797)

Miguel Filipe Rodrigues dos Santos (up202008450)

Rafael de Conceição Cerqueira (up201910200)

**Licenciatura em Engenharia Informática e Computação**

**Índice**

[1.Introdução 3](#_Toc161098914)

[2.Multiplicação de Matrizes (Algoritmos) 4](#_Toc161098915)

[2.1 Multiplicação Simples de Matrizes 4](#_Toc161098916)

[2.3Algoritmo de Multiplicação de Matrizes por Bloco 5](#_Toc161098919)

[3.Linguagens de programação testadas 6](#_Toc161098920)

# 1.Introdução

Este projeto foca se na análise do desempenho de algoritmos de multiplicação de matrizes em diferentes ambientes de processamento, desde sistemas de núcleo único até implementações multi-core. O nosso objetivo é compreender como a hierarquia de memória afeta o desempenho do processador ao lidar com volumes significativos de dados. Para atingir este objetivo, iremos utilizar a API de Desempenho (PAPI) para recolher dados relevantes durante a execução dos programas. Através de uma análise abrangente, esperamos obter insights valiosos sobre a otimização de algoritmos para diferentes arquiteturas de processamento e compreender melhor as complexidades do desempenho computacional em ambientes de núcleo único e multi-core.

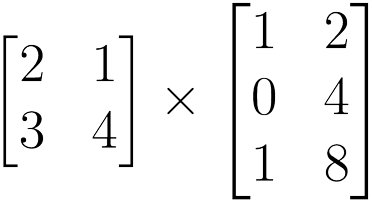
# 2.Multiplicação de Matrizes (Algoritmos)

Em busca de otimizar programas sequenciais, muitas vezes negligenciamos a importância da manipulação eficiente da memória. Neste projeto, exploramos a capacidade de aprimoramento de algoritmos sequenciais, propondo três abordagens distintas para medir o desempenho de um único núcleo diante de enormes volumes de dados, cada uma se diferenciando pela maneira como gerir alocação de memória. Esses algoritmos abrangem: Multiplicação Simples de Matrizes (já fornecido), Multiplicação de Matriz por Linha, Multiplicação de Matriz por Bloco.

Na tentativa de desenvolver soluções mais eficientes, encaramos o desafio de implementar códigos tanto em C/C++ quanto em uma linguagem adicional, optando pelo Java devido à sua afinidade com C/C++ em termos de abstração e sintaxe. Essa escolha não apenas facilitou a tradução de código entre as linguagens, mas também nos permitiu explorar as nuances de uma linguagem reconhecida por sua natureza híbrida entre interpretada e compilada, além de sua orientação totalmente voltada para objetos. Tais características têm potencial para influenciar significativamente o acesso à memória e, consequentemente, o tempo de execução dos algoritmos. Para a Multiplicação de Matriz por Bloco, porém, concentramo-nos exclusivamente na implementação em C/C++.

## 2.1 Multiplicação Simples de Matrizes

Neste método, cada valor presente na matriz A é multiplicado individualmente por cada elemento da linha correspondente da matriz B. Os resultados dessas multiplicações são então inseridos na linha apropriada da matriz resultante, denominada C.



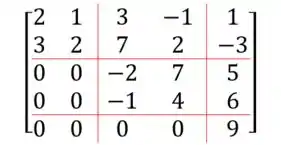
## 2.2 Multiplicação de Matrizes Linha a Linha

No algoritmo de multiplicação de matrizes linha-a-linha, cada elemento presente na matriz A é multiplicado individualmente por cada elemento da linha correspondente na matriz B. Os resultados dessas multiplicações são então inseridos na linha apropriada da matriz resultante C.

## Como resolver multiplicação de matrizes de maneira simples | O Baricentro da Mente

## 2.3Algoritmo de Multiplicação de Matrizes por Bloco

No caso deste método, desenvolvemos um algoritmo aprimorado que divide as matrizes originais em blocos de tamanho predefinido pelo utilizador. Em seguida, é realizada a multiplicação dos blocos correspondentes das duas matrizes para calcular a matriz resultante. Esta multiplicação entre os blocos é executada utilizando o algoritmo de multiplicação linha por linha.



# 3.Linguagens de programação testadas

Para avaliar o desempenho dos algoritmos nas versões em C/C++, utilizamos a Performance API (PAPI), que oferece acesso a diversas métricas da CPU e aos níveis de memória cache usados pelo processo. Além do tempo de execução do algoritmo, também consideramos o número de Operações de Ponto Flutuante (FLOP) e o número de falhas de cache, tanto para L1 quanto para L2. Falhas de cache podem resultar em um aumento significativo do tempo de processamento, por isso, é crucial monitorar sua variação para avaliar a eficiência da implementação. Esses valores não são estimados, são exatos e diretamente obtidos da API.

No presente estudo, a principal linguagem de programação utilizada foi o C++, escolhida devido à sua compatibilidade com a ferramenta PAPI. No entanto, para uma avaliação comparativa em diferentes ambientes, também decidimos utilizar Java. Uma característica importante do C++ é que ele armazena arrays bidimensionais em ordem de linha principal (row-major order), o que significa que os elementos de uma matriz são armazenados em linhas contínuas na memória. Em contrapartida, em uma ordem de coluna principal (column-major order), os elementos das colunas são armazenados consecutivamente na memória. Em ambas as linguagens, implementamos versões dos algoritmos de multiplicação, tanto por coluna quanto por linha, e analisamos o tempo de execução de cada uma. Para garantir resultados mais precisos, realizamos 10 execuções de cada algoritmo e calculamos a média dos resultados obtidos.

Os algoritmos implementados serão referidos como onMult, onMultLine e onMultBlock, correspondendo, respetivamente, aos algoritmos de multiplicação normal de matrizes, multiplicação linha-a-linha e multiplicação bloco-a-bloco.