# UNIVERSIDADE FEDERAL DO CARIRI

CICERO IGOR ALVES TORQUATO DOS SANTOS

## MULTIPLICAÇÃO DE MATRIZES

IMPLEMENTAÇÃO

JUAZEIRO DO NORTE

2023

# O PROBLEMA

O problema de multiplicação de matrizes envolve multiplicar duas matrizes para obter uma terceira matriz resultante. A multiplicação de matrizes é um problema complicado devido à sua natureza computacionalmente intensiva e ao grande volume de dados envolvidos. Existem algumas razões específicas pelas quais esse problema pode ser desafiador como a complexidade algorítmica e o consumo de memória.

Primeiramente, o algoritmo de multiplicação de matrizes possui uma complexidade de tempo de O(n³), onde "n" é a dimensão das matrizes. Isso significa que o tempo de execução aumenta rapidamente à medida que o tamanho das matrizes aumenta. Para matrizes muito grandes, o tempo necessário para calcular a multiplicação pode se tornar impraticável.

Além disso, a multiplicação de matrizes requer armazenamento temporário dos elementos intermediários durante o cálculo. Se as matrizes envolvidas são muito grandes, o consumo de memória pode se tornar um problema, especialmente em sistemas com restrições de memória. Gerenciar eficientemente a alocação e liberação de memória é crucial para evitar problemas de falta de memória durante a execução do programa.

Devido a esses desafios, é importante aplicar técnicas de otimização e utilizar bibliotecas como o OpenMP e o Open MPI para tirar proveito de recursos paralelos e reduzir o tempo de execução da multiplicação de matrizes.

# ALGORITMOS

## 2.1. ALGORITMO SEQUENCIAL

|  |
| --- |
| [Programacao\_Concorrente/Sequencial.c at main · IgorTorquatto/Programacao\_Concorrente · GitHub](https://github.com/IgorTorquatto/Programacao_Concorrente/blob/main/Projeto_Final/Sequencial/Sequencial.c) |

## 2.2. ALGORITMO PARALELO COM OPENMP

|  |
| --- |
| [Programacao\_Concorrente/OpenMp.c at main · IgorTorquatto/Programacao\_Concorrente · GitHub](https://github.com/IgorTorquatto/Programacao_Concorrente/blob/main/Projeto_Final/Paraleliza%C3%A7%C3%A3o/OpenMp.c) |

## 2.3. ALGORITMO PARALELO COM OPENMPI

|  |
| --- |
| [Programacao\_Concorrente/OpenMPI.c at main · IgorTorquatto/Programacao\_Concorrente · GitHub](https://github.com/IgorTorquatto/Programacao_Concorrente/blob/main/Projeto_Final/Paraleliza%C3%A7%C3%A3o/OpenMPI.c) |

# ANÁLISE DE DESEMPENHO

## 3.1. RESULTADOS

Computador do laboratório K05-s da UFCA:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **Quantidade de elementos** | **Tempo (segundos)** | **Speedup** | **Comentários** |
| Sequencial | 27.000.000.000 | 108.102 | - | O tempo de execução mostrado é a média dos valores (108.07, 106.70, 108.42, 108.52, 108.80). |
| OpenMP (4 threads) | 27.000.000.000 | 112.594 | 0.960 | O tempo de execução mostrado é a média dos valores (112.10, 112.35, 112.88, 111.77, 113.87). |
| MPI (4 processos) | 27.000.000.000 | 96.828 | 1.116 | O tempo de execução mostrado é a média dos valores (96.46, 96.54, 96.69, 97.02, 97.43). |
| MPI (5 processos) | 27.000.000.000 | 96.536 | 1.119 | O tempo de execução mostrado é a média dos valores (96.59, 96.93,96.31, 96.54, 96.31). |

Computador pessoal:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **Quantidade de elementos** | **Tempo (segundos)** | **Speedup** | **Comentários** |
| Sequencial | 27.000.000.000 | 311.5 | - | O tempo de execução mostrado é a média dos valores (315.76, 288.36, 318.98,316.66, 317.74). |
| OpenMP (2 threads) | 27.000.000.000 | 196.17 | 1.587 | O tempo de execução mostrado é a média dos valores (196.99, 191.00, 210.31, 194.59, 187.96). |
| OpenMP (4 threads) | 27.000.000.000 | 143.902 | 2.164 | O tempo de execução mostrado é a média dos valores (139.78, 146.34, 148.93, 138.74, 145.72). |

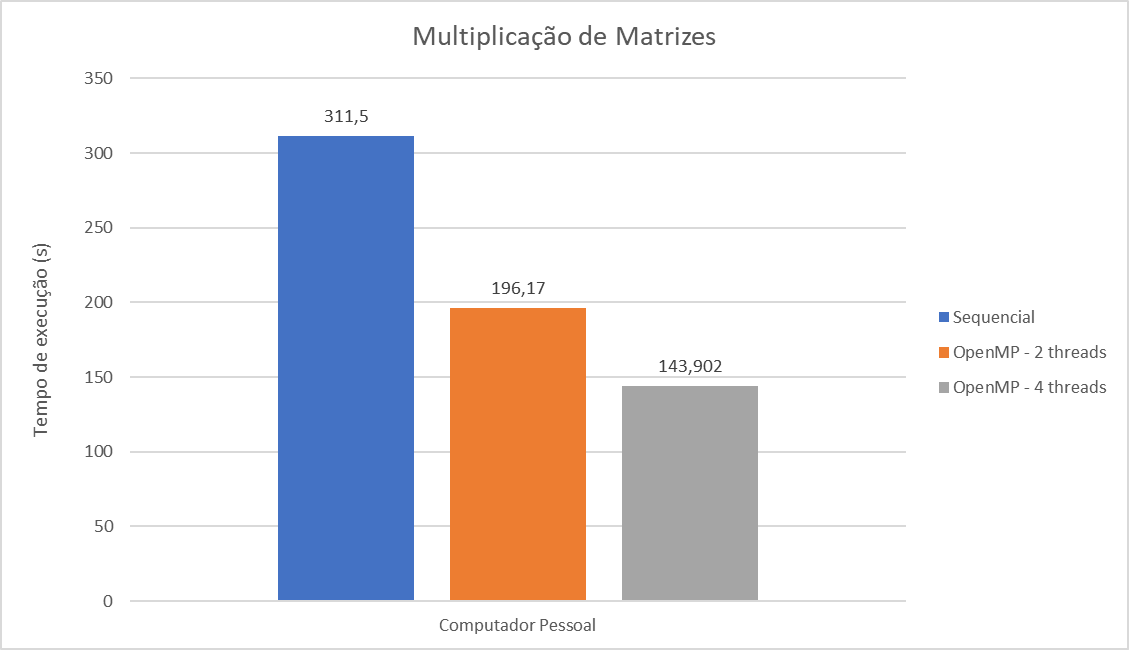
Em primeiro plano, para obter uma maior diversidade nos resultados, a análise de desempenho foi feita em dois computadores: o do laboratório K-05s e o pessoal. Nesse sentido, os recursos computacionais de ambos os computadores estão especificados nesse documento. Além disso, os algoritmos utilizados foram os mesmos nos dois casos, mas somente o computador do laboratório executou o algoritmo que utiliza a biblioteca OpenMPI devido a limitações no computador pessoal para executar processos.

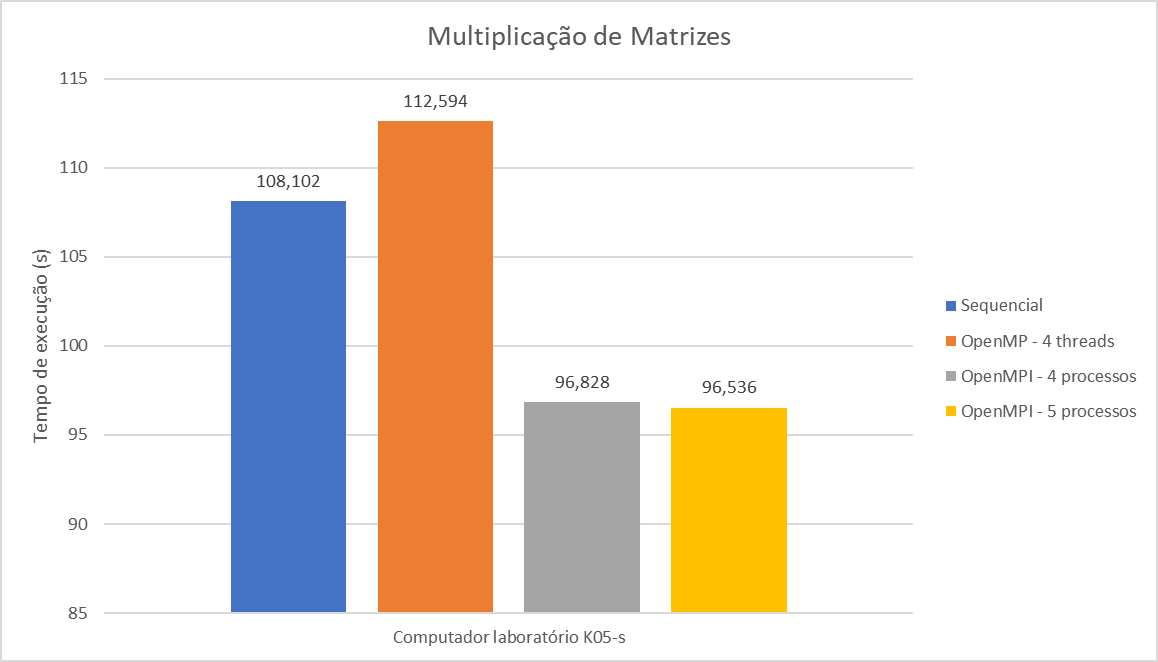
Dessa forma, por ser o mesmo código, podemos concluir que o problema de multiplicação de matrizes vale a pena ser paralelizado pois, com os resultados do computador pessoal, percebemos que o tempo de execução diminuiu consideravelmente ao utilizarmos threads, com o OpenMp. Ainda nesse contexto, com os resultados do computador do laboratório, percebemos que mesmo que não haja uma diferença tão grande nos tempos de execução do algoritmo sequencial para o algoritmo que utiliza threads, o algoritmo que utiliza processos, com o OpenMPI, se sai melhor nesse quesito.

Com isso, destacam-se as execuções no computador pessoal o qual o algoritmo com 2 threads alcança 0.79 de eficiência e o algoritmo com 4 threads alcança 0.54 de eficiência.

## 3.2. GRÁFICOS

Multiplicação de matrizes para N= 3000 (30003  = 27000000000).





# HARDWARE E SOFTWARE

Computador laboratório K05-s da UFCA:

|  |  |
| --- | --- |
| **HARDWARE/SOFTWARE** | **MODELO** |
| CPU | Intel (R) Core (TM) i5-8500T 2.10GHz |
| RAM | 8GB |
| SISTEMA OPERACIONAL | Debian GNU/Linux 11 |
| LINGUAGEM DE PROGRAMAÇÃO | C |
| COMPILADOR | GCC 10.2.1 |
| MPI | 4.0.2 |

Computador pessoal:

|  |  |
| --- | --- |
| **HARDWARE/SOFTWARE** | **MODELO** |
| CPU | AMD Athlon 3000G |
| RAM | 8GB DDR4 |
| SISTEMA OPERACIONAL | Windows 10 |
| LINGUAGEM DE PROGRAMAÇÃO | C |
| COMPILADOR | gcc (MinGW.org GCC-6.3.0-1) 6.3.0 |
| MPI | Microsoft MPI (MS-MPI) v10.1.2 |

# IMPLEMENTAÇÃO

[Programacao\_Concorrente/Projeto\_Final at main · IgorTorquatto/Programacao\_Concorrente (github.com)](https://github.com/IgorTorquatto/Programacao_Concorrente/tree/main/Projeto_Final)

# DIFICULDADES ENCONTRADAS

As maiores dificuldades encontradas foram gerenciar a alocação e liberação de memória porque quando se trabalha com matrizes grandes, como no problema, a alocação e liberação adequadas de memória se tornam importantes. Dessa forma, foi necessário garantir que havia memória suficiente para que as matrizes fossem alocadas e depois do término do programa desalocadas.

Nesse sentido, para que o código não apresentasse erro de segmentação foi utilizado o conceito de ponteiro para matriz para alocar memória para as matrizes A, B e C. Isso permitiu a alocação correta de memória para matrizes bidimensionais de tamanho N x N. Além disso, foi adicionado o código para liberar a memória alocada usando a função free no final do programa. Essas modificações na alocação de memória corrigiram o erro de segmentação que estava ocorrendo nas primeiras execuções.

# REFERÊNCIAS

STEIN, Benhur; NAVAUX, Philippe Olivier Alexandre. Programação Paralela com OpenMP.

BORIN, Edson. Matrizes - Algoritmos e Programação de Computadores. Instituto de Computação (IC/Unicamp)