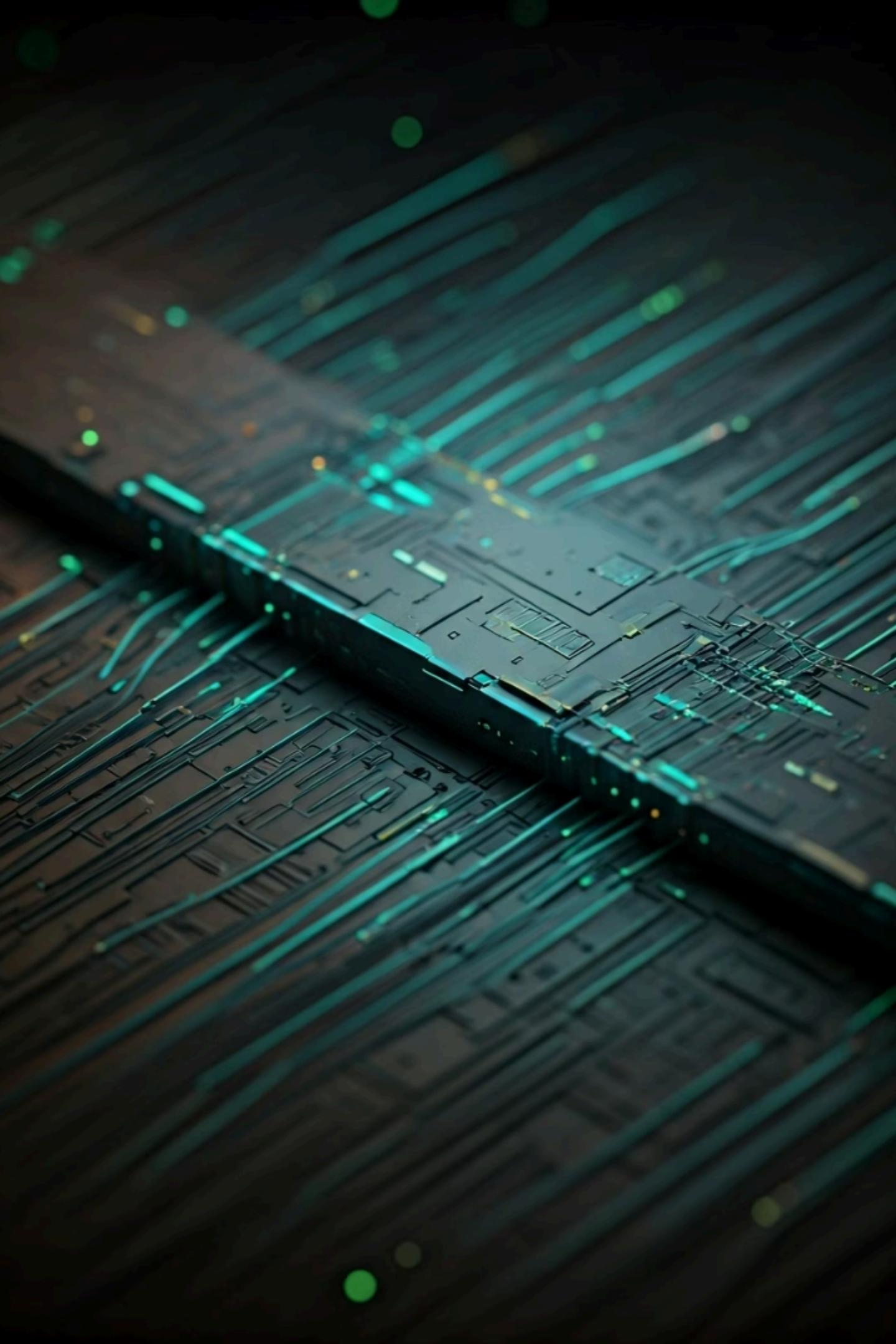


Multiplicação de Matrizes com Cálculo Paralelo usando Threads

Compontes: Guilherme Ramos e Lucas Nobre

Uma jornada no mundo da computação de alto desempenho.



O que é Multiplicação de Matrizes?

Definição Fundamental

Operação entre duas matrizes A ($m \times n$) e B ($n \times p$).

Gera uma matriz C ($m \times p$).

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \cdot B[k][j]$$



Entender essa operação é crucial para otimização.

Custo Computacional



Complexidade $O(n^3)$

Multiplicação tradicional.

Tempo Cresce Rapidamente

Aumento exponencial com o tamanho da matriz.

Solução: Paralelização

Distribuir o trabalho.

A ineficiência sequencial exige abordagens inovadoras.

Paralelismo com Threads

Distribuição de Trabalho
Múltiplas threads envolvidas.



Threads transformam o desempenho computacional.

Atribuição de Tarefas
Cada thread calcula linhas ou blocos.

Redução de Tempo
Execução significativamente mais rápida.

Arquitetura do Projeto



`utils.c / utils.h`

Funções auxiliares.



`sequential.c`

Implementação sem threads.



`paralelo.c`

Uso de `pthread.h`.



`benchmark.py`

Script de comparação.

Estrutura modular para clareza e eficiência.

Os códigos

```
1 #ifndef UTILS_H
2 #define UTILS_H
3
4 int** alocar_matriz(int linhas, int colunas);
5 void liberar_matriz(int** matriz, int linhas);
6 void preencher_matriz(int** matriz, int linhas, int colunas);
7 void salvar_matriz(const char* nome_arquivo, int** matriz, int linhas, int colunas);
8 int** carregar_matriz(const char* nome_arquivo, int linhas, int colunas);
9
10#endif
11
12int** alocar_matriz(int linhas, int colunas) {
13    int** matriz = (int**) malloc(linhas * sizeof(int*));
14    for (int i = 0; i < linhas; i++) {
15        matriz[i] = (int*) malloc(colunas * sizeof(int));
16    }
17    return matriz;
18}
19
20void liberar_matriz(int** matriz, int linhas) {
21    for (int i = 0; i < linhas; i++) {
22        free(matriz[i]);
23    }
24    free(matriz);
25}
26
27void preencher_matriz(int** matriz, int linhas, int colunas) {
28    for (int i = 0; i < linhas; i++) {
29        for (int j = 0; j < colunas; j++) {
30            matriz[i][j] = rand() % 10;
31        }
32    }
33}
```

```
void salvar_matriz(const char* nome_arquivo, int** matriz, int linhas, int colunas) {
    FILE* f = fopen(nome_arquivo, "w");
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            fprintf(f, "%d ", matriz[i][j]);
        }
        fprintf(f, "\n");
    }
    fclose(f);
}
int** carregar_matriz(const char* nome_arquivo, int linhas, int colunas) {
    FILE* f = fopen(nome_arquivo, "r");
    int** matriz = alocar_matriz(linhas, colunas);
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            fscanf(f, "%d", &matriz[i][j]);
        }
    }
    fclose(f);
    return matriz;
}
```

```
def run_program(cmd):
    result = subprocess.run(cmd, capture_output=True, text=True)
    return result.stdout.strip()

def extrair_tempo(texto):
    match = re.search(r'Tempo .*: ([0-9.]+) segundos', texto)
    if match:
        return float(match.group(1))
    else:
        return None

def main():
    tamanho = 3000

    print(f"\n==== Benchmark Multiplicacao de Matrizes ({tamanho}x{tamanho}) ===\n")

    print("Versao Sequencial:")
    out_seq = run_program(["./sequencial", str(tamanho)])
    print(out_seq)
    tempo_seq = extrair_tempo(out_seq)

    print("\nVersao Paralela (8 threads):")
    out_par = run_program(["./paralelo", str(tamanho)])
    print(out_par)
    tempo_par = extrair_tempo(out_par)

    metodos = ['Sequencial', 'Paralelo (8 threads)']
    tempos = [tempo_seq, tempo_par]

    plt.figure(figsize=(8,5))
    plt.bar(metodos, tempos, color=['blue', 'green'])
    plt.ylabel('Tempo (segundos)')
    plt.title(f'Benchmark Multiplicação de Matrizes {tamanho}x{tamanho}')
    for i, v in enumerate(tempos):
        plt.text(i, v + max(tempos)*0.01, f"{v:.2f}s", ha='center', fontweight='bold')
    plt.show()

if __name__ == "__main__":
    main()
```

Benchmark em Python



Arquivo benchmark.py

Script principal.



Execução subprocess

Chama os programas C.



Medição de tempo

time.perf_counter().

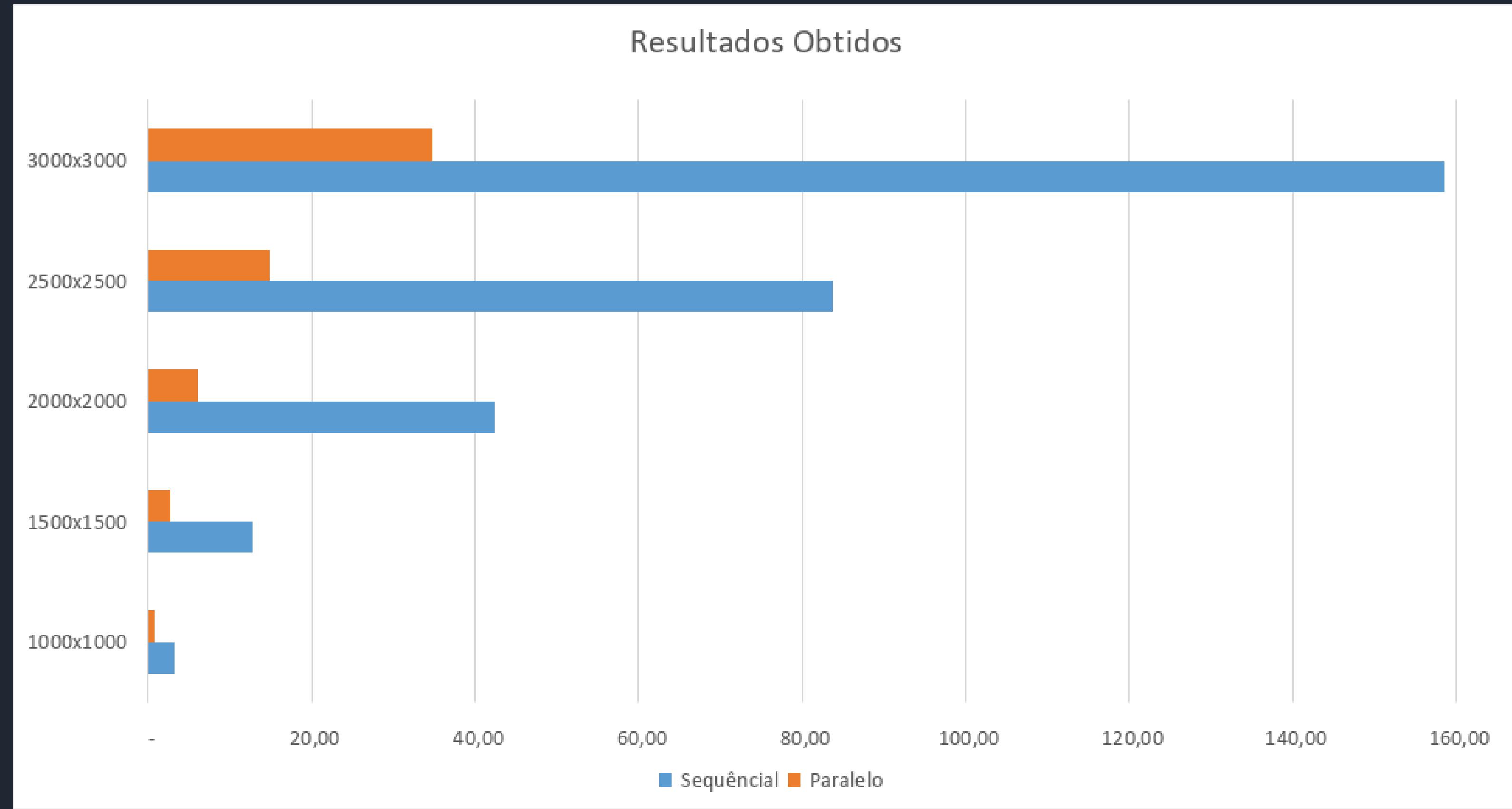


Exemplo de uso

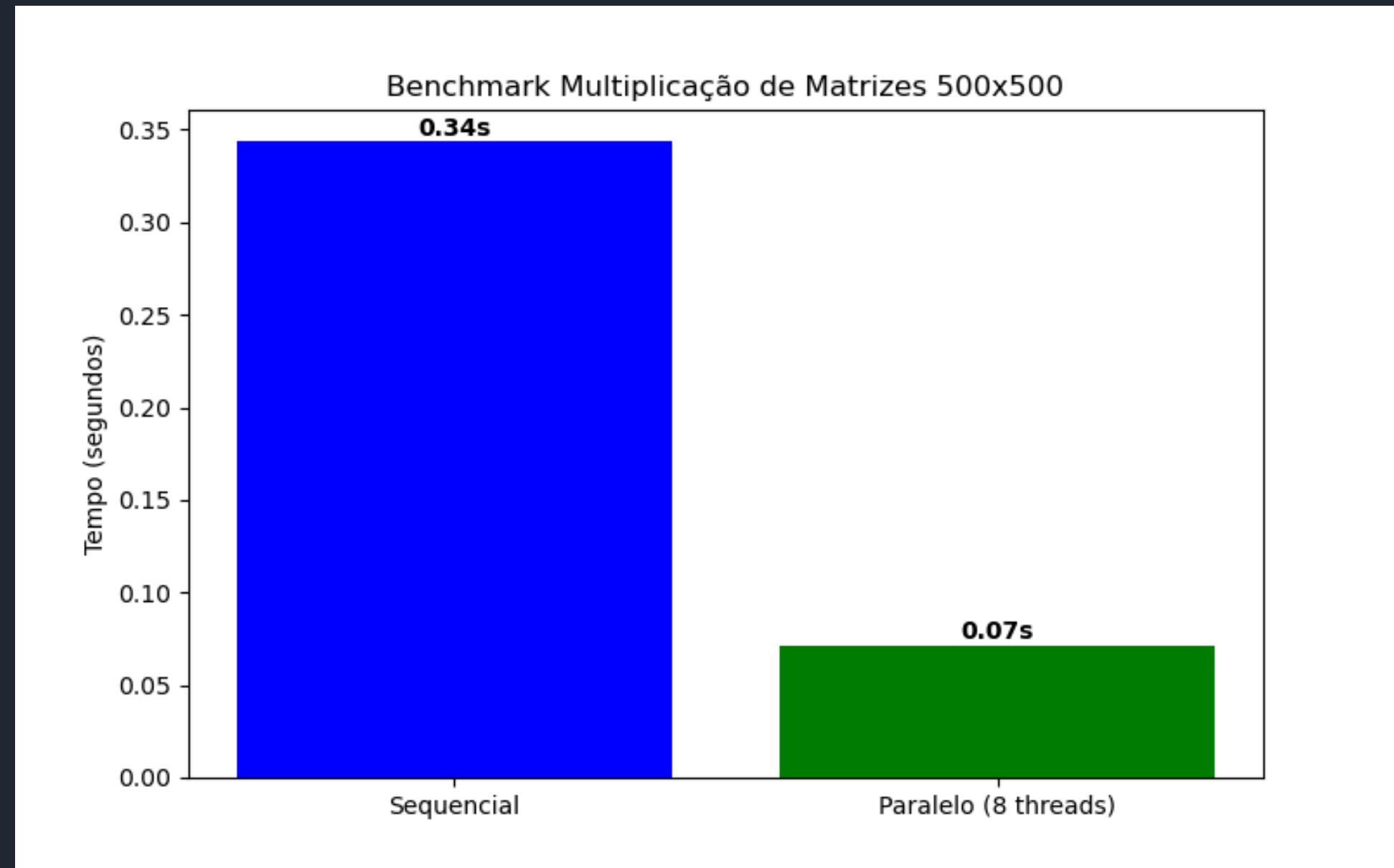
python benchmark.py

Python simplifica a análise de desempenho.

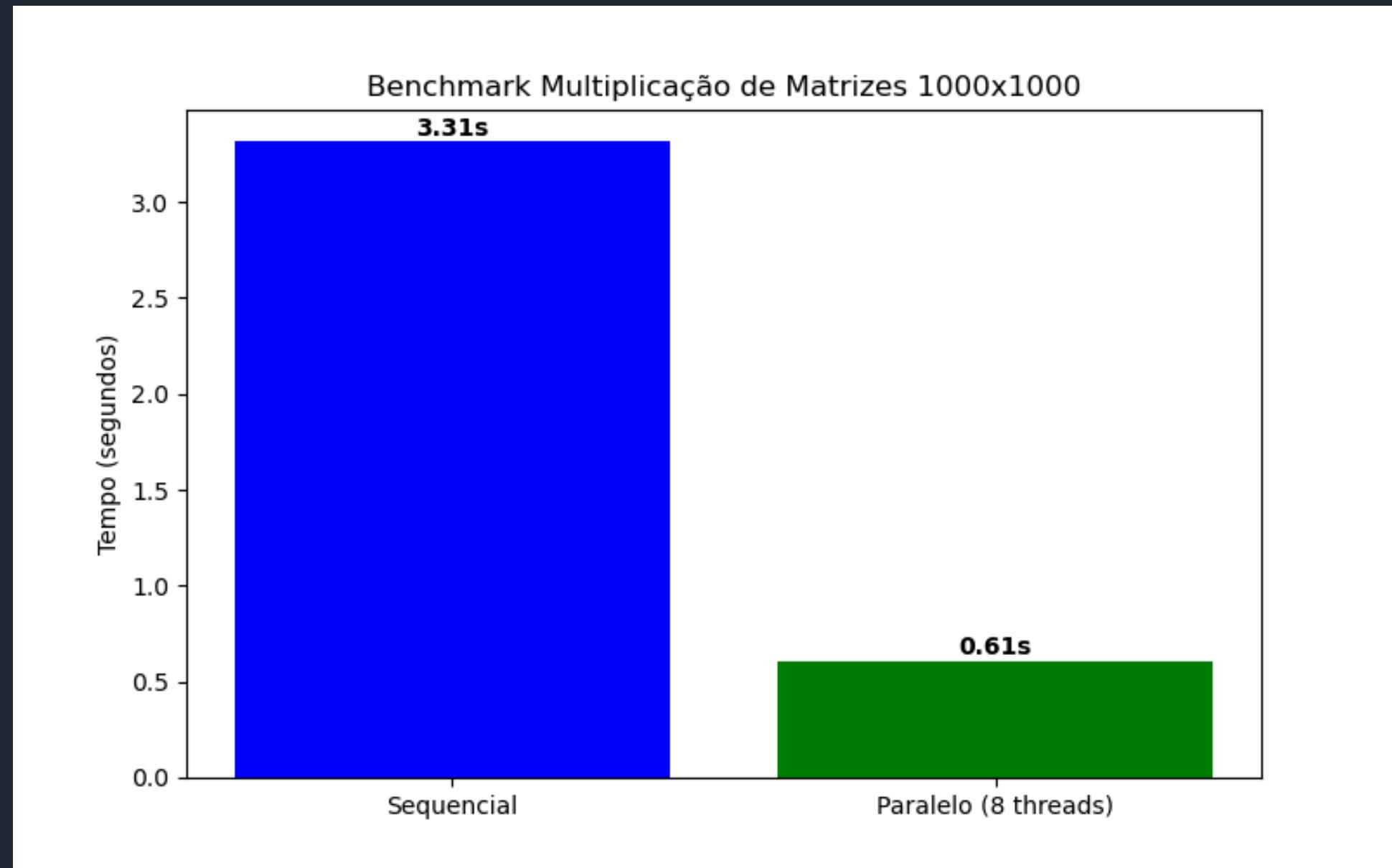
Resultados Obtidos



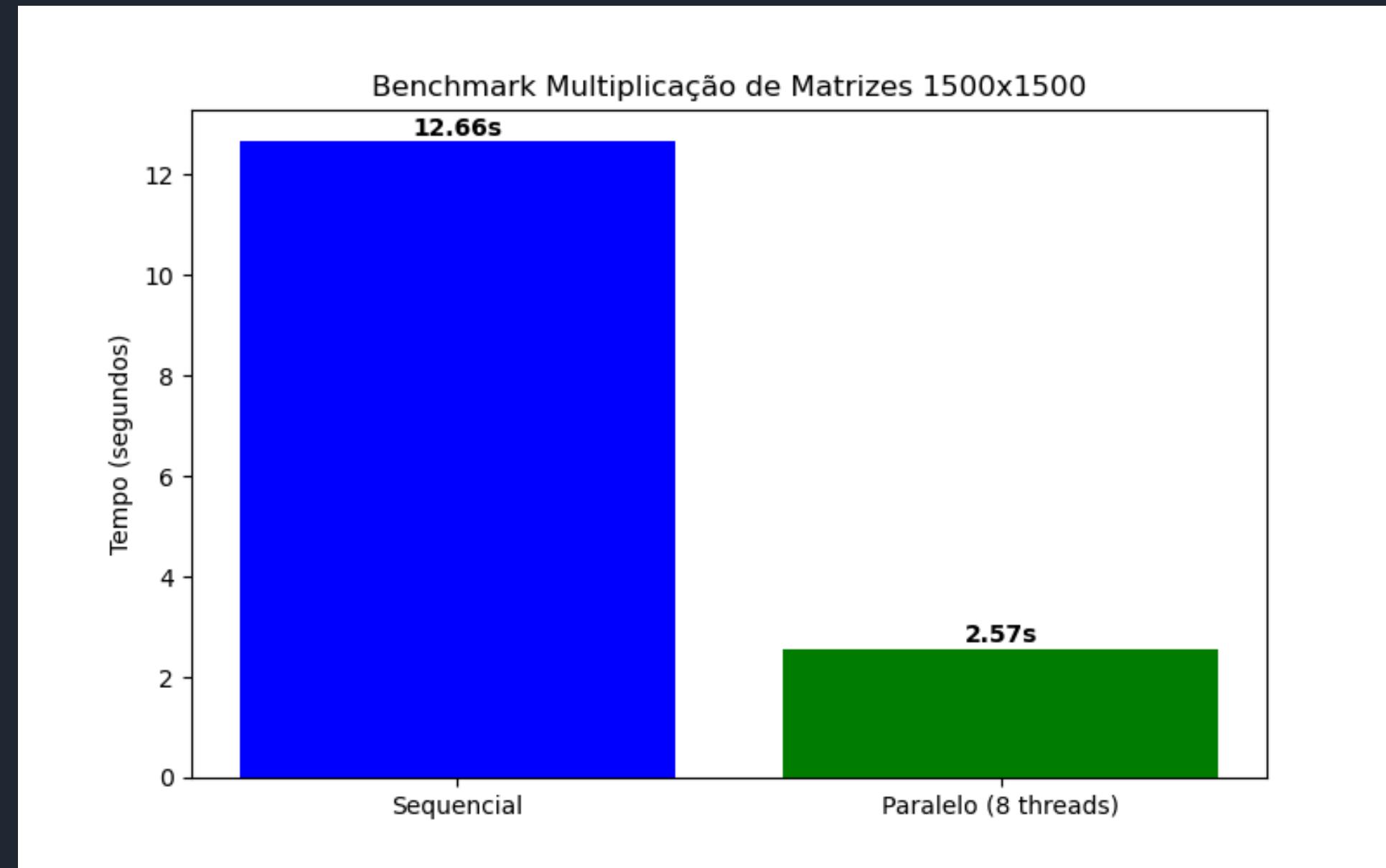
Resultados Obtidos



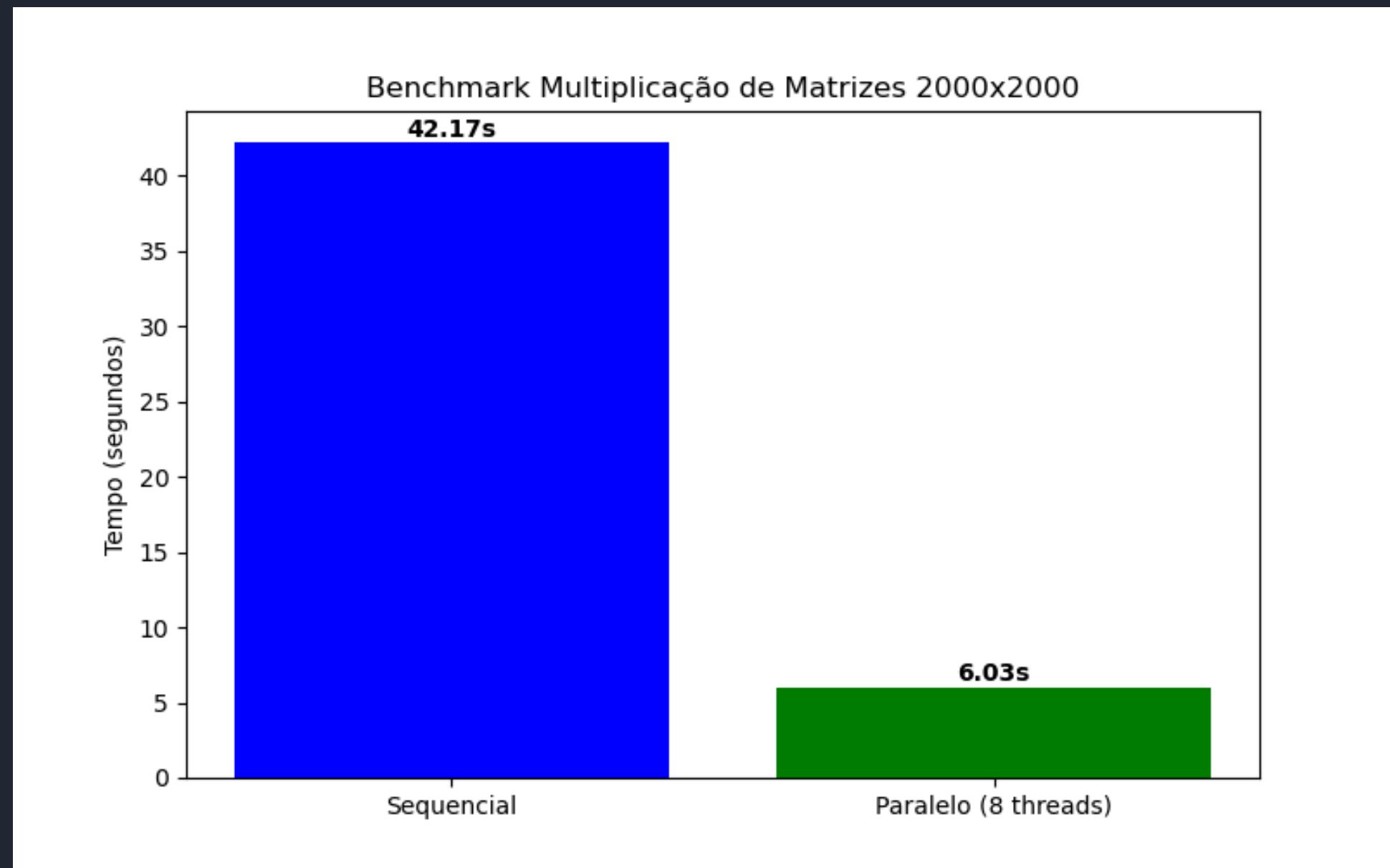
Resultados Obtidos



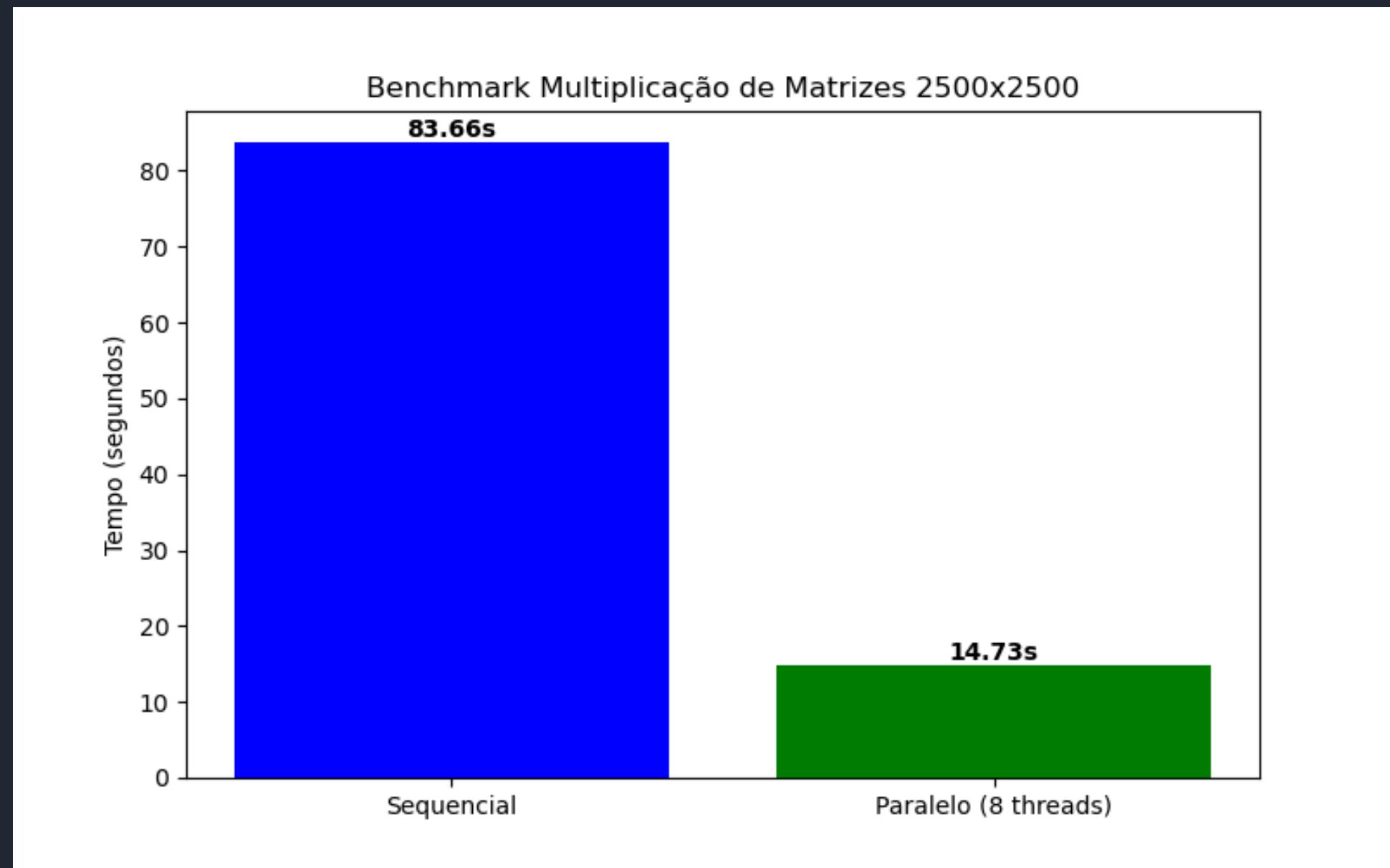
Resultados Obtidos



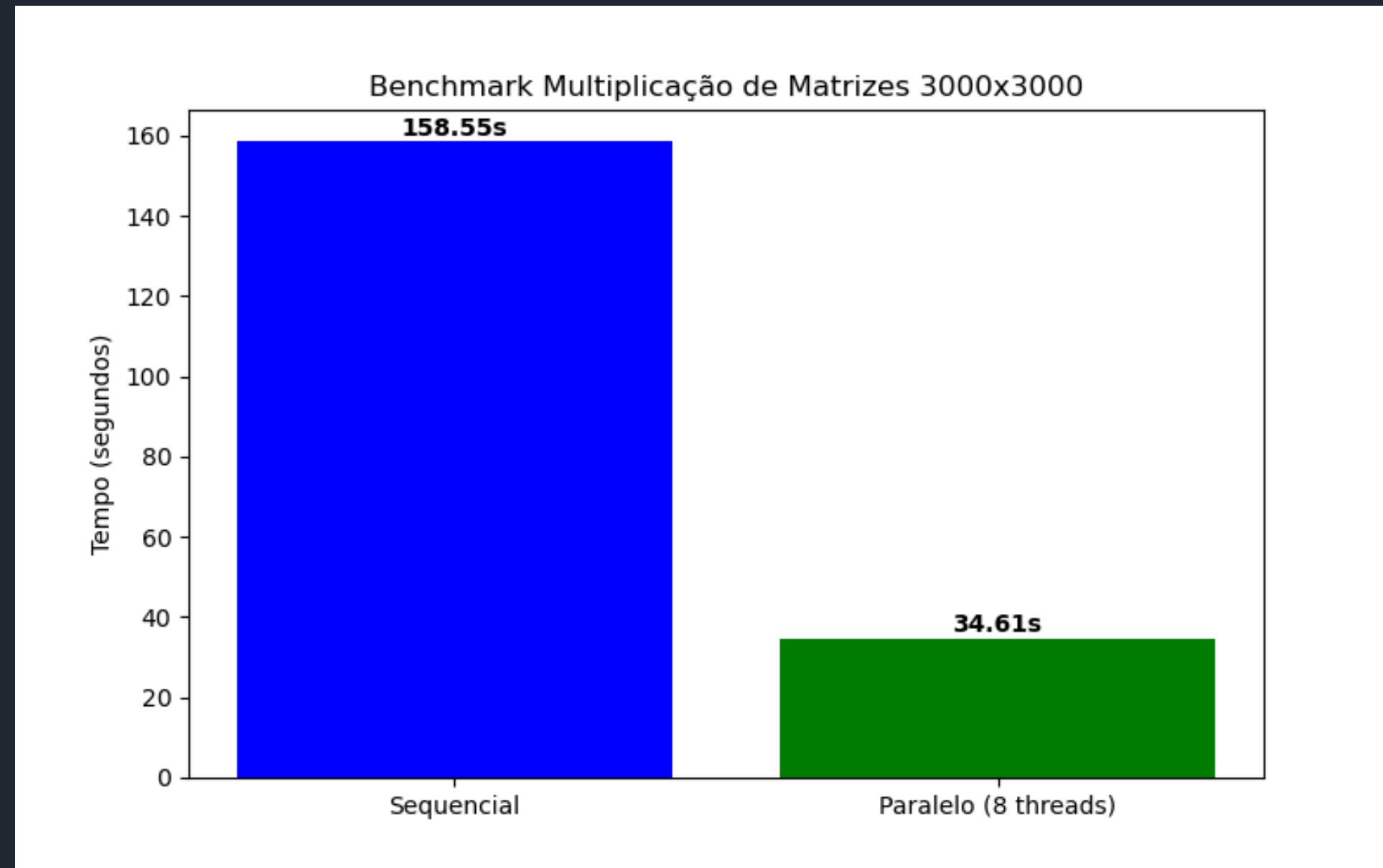
Resultados Obtidos



Resultados Obtidos



Resultados Obtidos



Considerações Técnicas



Sem Deadlock

Acesso a linhas distintas.



Memória Controlada

Uso otimizado.

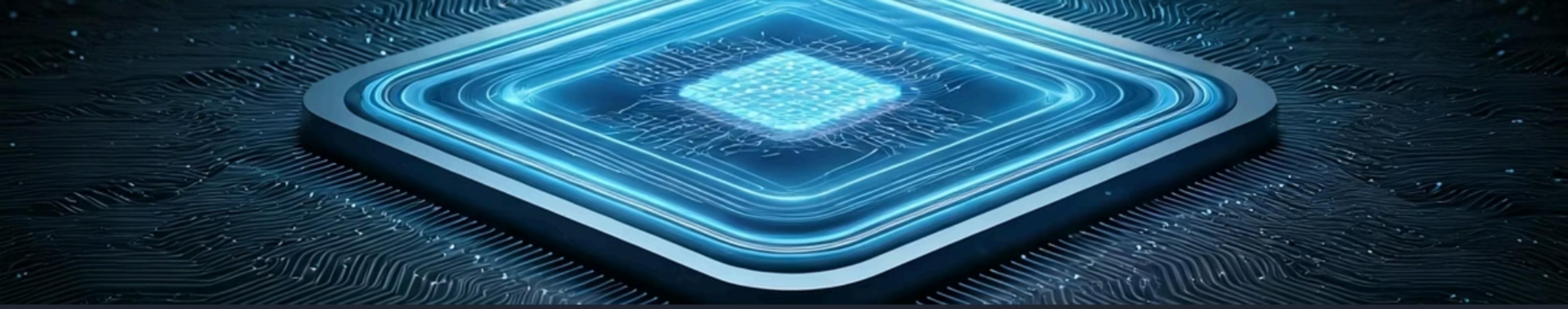


Cuidado com Grandes Matrizes

RAM e tempo.

Pontos cruciais para robustez e escalabilidade.





Conclusões

Paralelização Essencial

Redução drástica no tempo de execução.

A computação paralela é a chave para o futuro do processamento de dados.

Ideal para Matrizes Grandes
Acima de 1000x1000.

Python Simplifica
Benchmark e visualização.