# Análise de Desempenho da Multiplicação de Matrizes com Paralelização OpenMP

**Ellen Bonafin, Heloisa Alves**

Centro de Ciências Exatas e Tecnológicas – Ciência da Computação
Universidade Estadual do Oeste do Paraná (UNIOESTE) – Cascavel, PR – Brasil

{ellencarine41@gmail.com, heloisaalves2001@gmail.com}

***Abstract.*** *This article demonstrates the performance of a parallelized code in the C programming language that performs matrix multiplication. We will use the OpenMP library for parallelization. We will show the analysis of different dimensions of arrays as we proportionally increase the number of threads. Our study points out that the problem is not scalable because it does not guarantee the laws of Amdahl and Gustafson.*

***Resumo.*** *Este artigo demonstra o desempenho de um código paralelizado na linguagem de programação C que realiza a multiplicação de matrizes. Utilizaremos a biblioteca OpenMP para a paralelização. Mostraremos a análise de diferentes dimensões de matrizes à medida que aumentamos proporcionalmente o número de threads. Nosso estudo aponta que o problema não é escalável pois ele não garante as leis de Amdahl e Gustafson.*

## 1. Introdução

Nos dias atuais, processadores de computadores e notebooks estão sendo fabricados com múltiplos núcleos. Assim necessitando ser compreendida a necessidade por parte dos desenvolvedores de software que avaliam sempre a melhor performance de suas aplicações, a buscarem a programação paralela.

"Programação paralela é a divisão de uma determinada aplicação em partes, de maneira que essas partes possam ser executadas simultaneamente, por vários elementos de processamento" (PUCRS– FACIN – Prof.TiagoFerreto).

Para melhor visualização e compreensão desse tema, será analisado neste artigo o desempenho de um algoritmo paralelizado de multiplicação de matrizes. Para o desenvolvimento deste trabalho utilizaremos a linguagem C, que tem como suporte a biblioteca OpenMP para o modelo de programação paralela de memória compartilhada.

O trabalho foi subdividido em: descrição do algoritmo (seção 2), contendo informações sobre a implementação paralela em C; apresentação do ambiente de execução (seção 3); apresentação os resultados obtidos (seção 4) e as considerações finais (seção 5).

## 2. Multiplicação de Matrizes

A implementação desse algoritmo segue a mesma lógica de um algoritmo sequencial, onde temos três laços de repetição para a manipulação das matrizes. O primeiro laço é responsável por iterar a matriz linha a linha, o segundo a coluna a coluna, e o terceiro percorrer a matriz linha a coluna. A única diferença é que foi atribuída as diretivas: *omp_set_num_threads()* e *#pragma omp parallel for*.

```
void MulM1M2(int **M1, int **M2, int **MR, int l, int c){
   int aux = 0;

   omp_set_num_threads(2);
   int i,j,k;
   #pragma omp parallel for private(aux,j,k) firstprivate(l,c) shared(i)
    for (i=0;i<l;i++){
         for (j=0;j<c;j++){
            aux = 0;
            for(k=0;k<c;k++) aux+=M1[i][k]*M2[k][j];

            MR[i][j] = aux;
         }
    }
}
```

Essa implementação completa está disponível em: https://github.com/Helogizzy/Matrix_Multiplication_OpenMP

Note que somente a variável I é compartilhada durante a execução, isso é necessário pois é ela que fará o controle do laço para as duas matrizes enquanto o J e K seguem privados para manter a igualdade para ambas as matrizes. Já as variáveis L e C recebem o *firstprivate,* visto que elas serão copiadas para as memórias locais da *thread*.

Durante os testes a diretiva *omp_set_num_threads()* receberá valores diferentes de *threads* para avaliar o impacto no tempo de processamento, mais detalhes desses valores serão abordados na seção 4.

## 3. Ambiente de Execução

Foi utilizado o supercomputador Santos Dumont (SDumont) por meio de uma VPN (rede privada virtual) para a execução do algoritmo. O SDumont está localizado na sede do Laboratório Nacional de Computação Científica (LNCC), em Petrópolis – RJ. Possui uma configuração híbrida de nós computacionais que são interligados por uma rede de interconexão Infiniband proporcionando alto rendimento e baixa latência tanto para comunicação entre os processos quanto para o acesso ao sistema de arquivos (SINAPAD, 2014).

Para a realização dos testes utilizamos a versão 9.3 do *gcc* (compilador C).

## 4. Análise de Desempenho

Adotamos as multiplicações de matrizes das seguintes dimensões: 1000x1000, 2000x2000, 4000x4000 e 8000x8000. A fim de avaliar o impacto no tempo de processamento do algoritmo paralelo, cada umas das dimensões receberam testes de 1, 2, 4, 8, 16 e 24 *threads*.

Os resultados são apresentados em segundos por meio da função *omp_get_wtime*. A seguinte função "retorna um valor de precisão dupla igual ao número de segundos desde o valor inicial do relógio de tempo real do sistema operacional".

```
tempo = omp_get_wtime();
MulM1M2(M1,M2,MR,l1,c2);
tempo = omp_get_wtime() - tempo;
printf("%f\n", tempo);
```

Essa implementação completa está disponível em: https://github.com/Helogizzy/Matrix_Multiplication_OpenMP

A avaliação do tempo de processamento pode ser observada na Tabela 1.

| Tabela de desempenho | | | | | | |
|---|---|---|---|---|---|---|
| Execução | 1 thread | 2 threads | 4 threads | 8 threads | 16 threads | 24 threads |
| 1000x1000 | 8.230.014 | 3.508.659 | 1.844.862 | 1.033.859 | 0.512569 | 0.356034 |
| 2000x2000 | 68.256.250 | 30.809.361 | 16.074.947 | 8.755.856 | 4.479.211 | 3.237.920 |
| 4000x4000 | 610.812.710 | 404.313.384 | 215.906.496 | 111.230.914 | 56.200.294 | 39.045.072 |
| 8000x8000 | 4.282.710.638 | 2.490.922.513 | 1.430.829.003 | 933.368.178 | 539.315.257 | 353.538.302 |

**Tabela 1 ( Número de Threads x Dimensão da Matriz).**

Considerando os tempos de processamento da Tabela 1 podemos perceber que dentre todos os tamanhos de matrizes, os menores tempos obtidos foram com 24 *threads*.

Para uma análise mais profunda do desempenho, foi calculado os tempos de *speedup* e eficiência, eles podem ser vistos na Tabela 2. Onde, a letra S representa o tempo de *speedup* e a letra E representa a eficiência.

| | Desempenho | 1 Thread | 2 Threads | 4 Threads | 8 Threads | 16 Threads | 24 Threads |
|---|---|---|---|---|---|---|---|
| 1000x1000 | S | 1 | 2,34 | 4,46 | 7,96 | 16,05 | 23,11 |
| | E | 1 | 1,17 | 1,11 | 0,995 | 1,0031 | 0,9629 |
| 2000x2000 | S | 1 | 2,21 | 4,24 | 7,79 | 15,23 | 21,08 |
| | E | 1 | 1,1 | 1,06 | 0,973 | 0,9518 | 0,8783 |
| 4000x4000 | S | 1 | 1,51 | 2,82 | 5,4 | 10,86 | 15,64 |
| | E | 1 | 0,75 | 0,705 | 1,35 | 0,6787 | 0,6516 |
| 8000x8000 | S | 1 | 1,71 | 2,99 | 4,58 | 7,94 | 12,11 |
| | E | 1 | 0,85 | 0,7475 | 0,5725 | 0,4962 | 0,5045 |

**Tabela 2 ( *Speedup* e eficiência das matrizes).**

Ao observar os resultados do *speedup*, notamos que as matrizes com dimensões de 8000x8000 e 4000x4000 se saíram melhor, com ênfase na de 8000x8000 que teve um *speedup* bem siginificativo a partir do uso de 8 *threads* como demonstra o Gráfico 1.

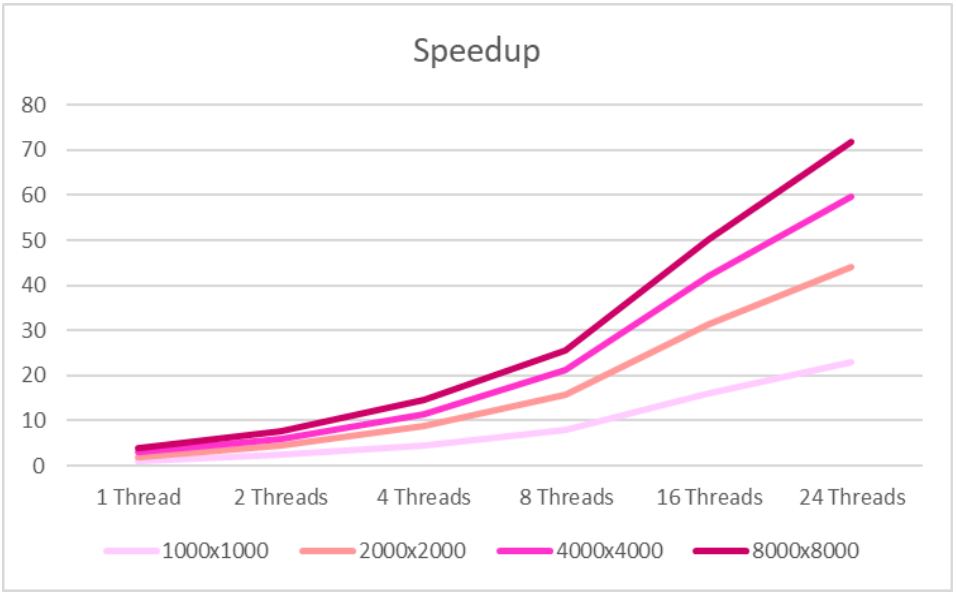


**Gráfico 1 (*Speedup* das matrizes).**

Agora, analisando os resultados **d**a eficiência temos o mesmo resultado, matrizes de dimensões 4000x4000 e 8000x8000 se sobresaíram novamente. Porém, em destaque para a matriz de 8000x8000 utilizando 16 e 24 *threads* observado no Gráfico 2.

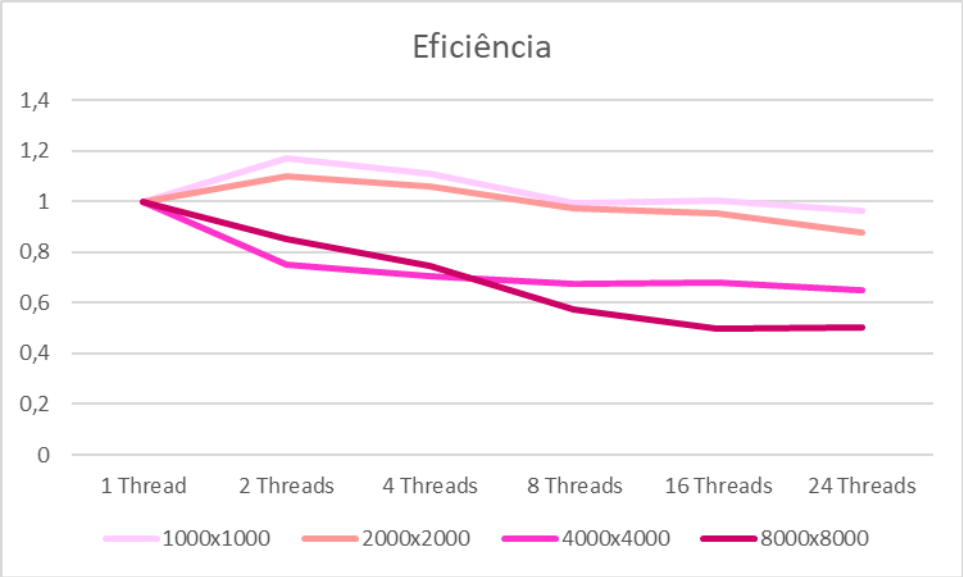


**Gráfico 2 (Eficiência das matrizes).**

## 5. Conclusão

Através dos resultados obtidos nos testes de desempenho podemos concluir que a aplicação não é escalável. Por mais que o speedup tenha se destacado e sido positivo, e o tempo de execução tenha caído pela metade, a eficiência não conseguiu acompanhar esse ritmo. Conseguimos observar que só vemos eficiência a partir da dimensão 4000x4000 e para ser escalável necessitaria demonstrar a capacidade de manter a eficiência conforme o número de processos aumenta, indo contra as leis de escalabilidade de Amdahl e Gustafson.

Os autores reconhecem o Laboratório Nacional de Computação Científica (LNCC/MCTI, Brasil) por fornecer recursos de HPC do supercomputados SDumont, que contribuíram para os resultados da pesquisa relatados neste artigo. *URL:* *http://sdumont.lncc.br*

## 6. Referências

BRASIL. Ministério da ciência, tecnologia e inovações. Projeto Sistema de Computação Petaflópica do SINAPAD. Configuração do SDumont. Laboratório Nacional de Computação Científica, 2014. Disponível em: < https://sdumont.lncc.br/machine.php?pg=machine >. Acesso em: 29/07/2022.

IBM. XL Fortran for AIX 15.1.0. 2021. Omp_get_wtime(). Disponível em: < https://www.ibm.com/docs/en/xl-fortran-aix/15.1.0?topic=openmp-omp-get-wtime >. Acesso em: 31/07/2022.