**UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE**
**IDCA3703 - PROGRAMAÇÃO PARALELA**

# TAREFA 16 - COMUNICAÇÃO COLETIVA COM MPI
# **RELATÓRIO DE EXECUÇÃO**

ERNANE FERREIRA ROCHA JUNIOR

NATAL/RN, 15 DE MAIO DE 2025

**SUMÁRIO**

# 1. INTRODUÇÃO

O aumento da demanda computacional em aplicações científicas e de engenharia tem impulsionado o uso de processamento paralelo como solução para problemas de grande escala. Entre os diversos paradigmas de paralelismo, o modelo baseado em troca de mensagens, como o MPI (*Message Passing Interface*), é amplamente utilizado em ambientes de computação de alto desempenho (*HPC*), permitindo a distribuição eficiente de tarefas entre múltiplos processos.

Neste contexto, o presente trabalho tem como objetivo implementar e avaliar o desempenho de um programa paralelo em *MPI* para o cálculo do produto de uma matriz por um vetor, definido como $y = A \cdot x$, onde $A$ é uma matriz de dimensão $M \times N$ e x é um vetor de tamanho $N$. O problema foi resolvido distribuindo-se as linhas da matriz $A$ entre os processos utilizando *MPI_Scatterv*, enquanto o vetor $x$ é transmitido integralmente a todos os processos com *MPI_Bcast*. Após o cálculo local de cada processo, os resultados parciais são reunidos no processo mestre utilizando *MPI_Gatherv*.

Para avaliar o desempenho do algoritmo, foram realizados experimentos variando o número de processos *MPI* e o tamanho da matriz, com medições precisas de tempo de execução. A análise baseia-se em métricas como o tempo médio de execução e o *speedup* obtido com o aumento do paralelismo, permitindo observar a escalabilidade do algoritmo implementado em diferentes configurações computacionais. Os experimentos foram conduzidos no ambiente do *NPAD/UFRN* (Núcleo de Processamento de Alto Desempenho da Universidade Federal do Rio Grande do Norte), explorando de forma prática o potencial do paralelismo distribuído para operações matriciais.

## 2. METODOLOGIA

A metodologia adotada neste trabalho compreende três etapas principais: a implementação do algoritmo paralelo utilizando *MPI*, a configuração dos testes no ambiente *HPC* do *NPAD/UFRN* e a análise dos dados obtidos por meio de medições de desempenho. O que queremos fazer exatamente é:

$$y = A \cdot x$$

Onde:

- $A \in \ \Re^{M \times N}$
- $x \in \Re^{N}$
- $y \in \Re^{M}$

Cada elemento de $y$ é dado por:

$$y_i = \sum_{j=1}^{N} A_i j \cdot x_j, \text{ para } i = 1, 2, \ \ldots, M$$

Como temos $P$ processos MPI, e queremos dividir as $M$ linhas da matriz $A$ entre eles. Então:

- Cada processo $p \in \{0, 1, \ \ldots, P-1\}$ recebe um subconjunto de linhas da matriz:

$$A^{(p)} \in \Re^{M_p \times N}$$

com $\sum_{p=0}^{P-1} M_p = M.$

- Com o vetor $x \in \Re^{N}$ realizamos o *broadcast* para todos os processos:

$$\forall p, \ x^{(p)} = x$$

- Cada processo calcula seu bloco local de $y$, denotado por:

$$y^{(p)} = A^{(p)} \cdot x$$

Dessa forma, depois que todos os processos calcularam seus $y^{(p)}$, o processo mestre realiza o *gather* (reunião em $y$)

$$y = \begin{bmatrix} y^{(0)} \\ y^{(0)} \\ \vdots \\ y^{(P-1)} \end{bmatrix}$$

Ou seja,

$$y = \bigcup_{p=0}^{P-1} \left( A^{(p)} \cdot x \right) \text{ onde } A^{(p)} \in \Re^{M_p \times N}, \ \sum_{p} M_p = M$$

Temos uma distribuição das partes de $A$, realização de *broadcast* do valor presente em $x$, a realização

do calculo das partes individuais em $y$ (locais) e uma reunião desses valores obtidos no $y$ da master em suas respectivas localidades.

A implementação desse algoritmo foi realizada em linguagem *C*, utilizando a biblioteca *MPI* para a comunicação entre processos. O problema central consiste no cálculo do produto vetor-matriz $y = A \cdot x$, onde $A$ é uma matriz $M \times N$ e $x$ um vetor de tamanho $N$. A matriz foi dividida entre os processos *MPI* por linhas, utilizando a função *MPI_Scatterv* para distribuir de forma balanceada (mesmo em casos em que $M$ não é divisível por $P$, número de processos). O vetor $x$, comum a todos os processos, foi distribuído com *MPI_Bcast*, garantindo que todos os processos possuíssem os dados necessários para realizar o cálculo local do produto escalar. Após os cálculos locais, os vetores parciais resultantes foram reunidos no processo mestre por meio da função *MPI_Gatherv*.

Os elementos da matriz $A$ e do vetor $x$ foram preenchidos com valores aleatórios uniformes entre $0$ e $1$, utilizando a função *rand_r* para garantir independência e reprodutibilidade entre os processos. A medição do tempo de execução foi realizada com *MPI_Wtime*, com início após a distribuição dos dados e término após o recolhimento dos resultados no processo mestre. O resultado do vetor $y$ não foi impresso para evitar impacto na performance e no tamanho dos logs.

Os testes foram executados no ambiente de computação de alto desempenho do *NPAD/UFRN*, utilizando um *script SLURM* para submissão de *jobs*. Foram utilizados até 32 processos *MPI*, e testados tamanhos de matrizes quadradas de $512 \times 512$ até $4096 \times 4096$. Para cada combinação de número de processos e tamanho de matriz, o experimento foi repetido três vezes, permitindo o cálculo de tempo médio de execução e desvio padrão. Os resultados foram registrados automaticamente no *log* de saída, posteriormente extraídos e organizados em uma planilha no formato *CSV* para análise estatística.

Por fim, as métricas de desempenho foram visualizadas por meio de gráficos gerados com *Python* (bibliotecas pandas e *matplotlib*). Foram construídos gráficos de tempo de execução em função do número de processos e do tamanho da matriz, bem como gráficos de *speedup*, permitindo avaliar a escalabilidade do algoritmo.

## 3. RESULTADOS

A avaliação do desempenho da implementação *MPI* para o produto $y = A \cdot x$ foi conduzida por meio de uma série de experimentos variando o número de processos (2, 4, 8, 16 e 32) e o tamanho da matriz quadrada ($512 \times 512$ até $4096 \times 4096$). Cada configuração foi executada três vezes consecutivas, e o tempo de execução foi medido utilizando *MPI_Wtime*. A média dos tempos foi utilizada como referência para análise.

Os resultados obtidos mostraram que, como esperado, o tempo de execução tende a aumentar proporcionalmente ao tamanho da matriz, e a diminuir com o aumento do número de processos. Esse comportamento é ilustrado no gráfico abaixo, que mostra uma tendência de crescimento quase linear para cada número fixo de processos. Esse crescimento é mais acentuado para configurações com menor grau de paralelismo, o que evidencia a limitação do processamento sequencial ou parcialmente paralelo quando lidando com grandes volumes de dados.

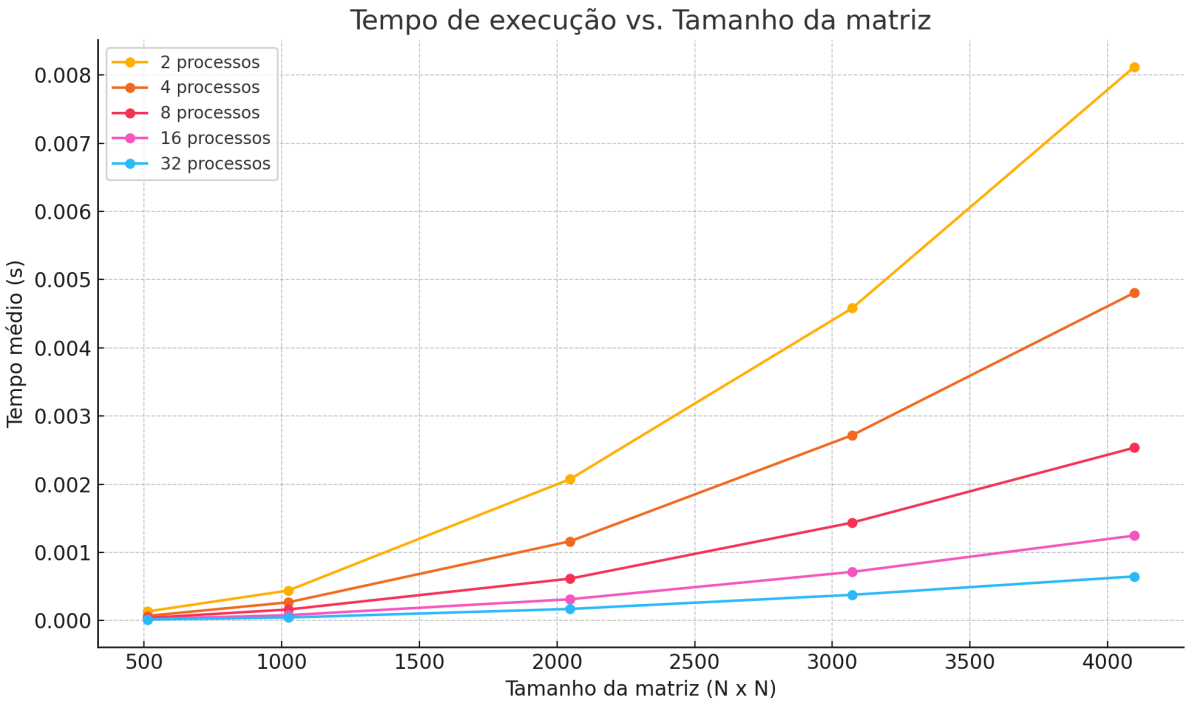


**Figure 1:** Tempo de execução vs. Tamanho da matriz

No gráfico abaixo, observa-se uma queda significativa no tempo de execução à medida que mais processos são utilizados, especialmente nas matrizes maiores. Esse resultado reforça a eficácia da distribuição das linhas da matriz entre os processos *MPI*. Contudo, nas matrizes menores, como $512 \times 512$ e $1024 \times 1024$, o ganho com o aumento de processos se torna marginal ou até negligenciável a partir de certo ponto, devido ao custo de comunicação entre processos superar o ganho computacional — um comportamento típico em aplicações com baixa carga de trabalho por processo.

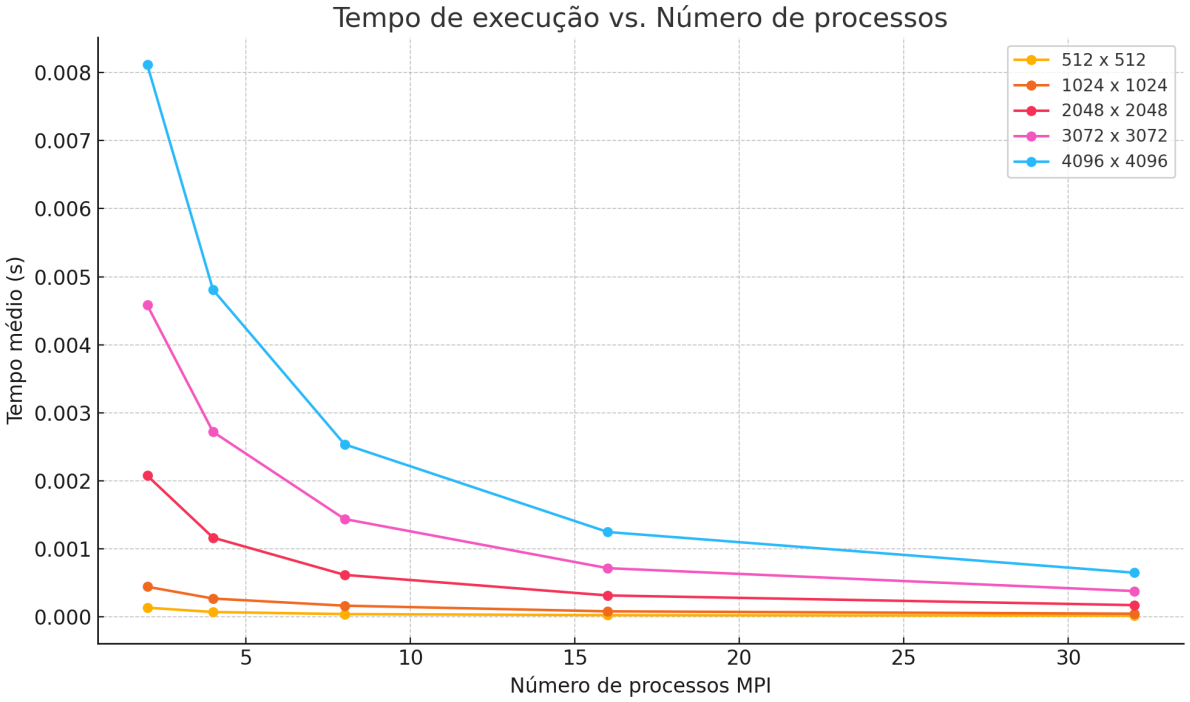


**Figure 2:** Tempo de Execução vs. Número de Processos

O gráfico mais revelador, ilustrado abaixo, apresenta uma visão clara da escalabilidade da aplicação. O *speedup* é definido como a razão entre o tempo de execução com 2 processos e o tempo com $P$ processos, e representa o ganho de desempenho relativo ao aumento do paralelismo. Ou seja:

$$Speedup(p) = \frac{T_{base}}{T(p)}$$

Onde,

- $T_{base}$: tempo de referência (nesse caso, 2 processos);
- $T(p)$: tempo com $p$ processos.

Em configurações com matrizes maiores, como $3072 \times 3072$ e $4096 \times 4096$, o *speedup* se aproxima de uma curva quase linear, atingindo valores acima de 12 quando se utilizam 32 processos — o que evidencia excelente escalabilidade para esses casos. Já para matrizes menores, o *speedup* cresce mais lentamente, revelando que o uso intensivo de paralelismo só é vantajoso quando há carga computacional suficiente para justificar a sobrecarga da comunicação *MPI*.

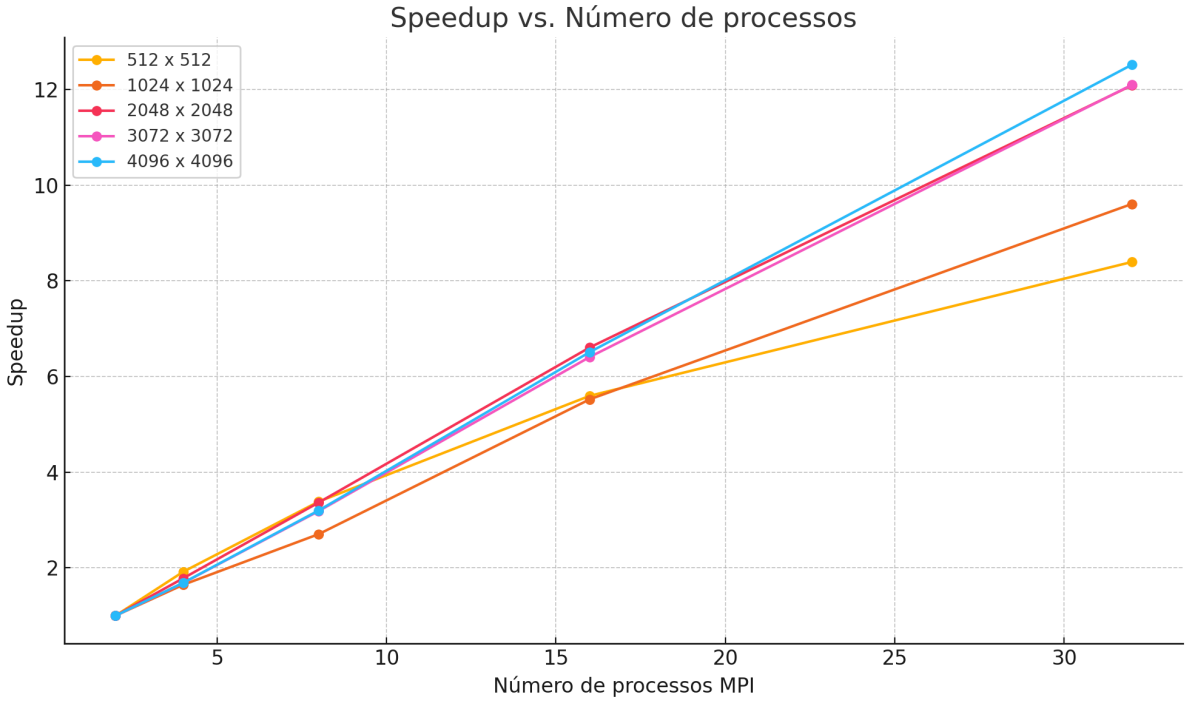


**Figure 3:** Speedup vs. Número de processos

Com base nos dados empíricos, pode-se afirmar que a implementação atinge seu melhor desempenho com o uso de 32 processos e matrizes de tamanho acima de $2048 \times 2048$. Nessas configurações, o tempo de execução é reduzido drasticamente em relação à configuração com apenas 2 processos, validando a eficiência da divisão de carga via *MPI_Scatterv*, a difusão do vetor $x$ com *MPI_Bcast* e a coleta dos resultados com *MPI_Gatherv*.

# 4. CONCLUSÃO

O experimento realizado demonstrou com clareza a efetividade do paralelismo via *MPI* para o cálculo do produto matriz-vetor $y = A \cdot x$ em ambientes de computação de alto desempenho. A implementação aproveitou adequadamente os recursos do modelo de troca de mensagens, utilizando *MPI_Scatterv* para distribuir as linhas da matriz $A$, *MPI_Bcast* para replicar o vetor $x$ entre os processos e *MPI_Gatherv* para reunir os resultados no processo mestre.

A análise dos tempos de execução evidenciou que o desempenho do programa melhora substancialmente com o aumento do número de processos, sobretudo para matrizes de grande porte. Os gráficos mostraram que, em cenários com matrizes maiores, a escalabilidade se aproxima do ideal, com *speedup* quase linear. Por outro lado, para matrizes pequenas, os ganhos de desempenho são limitados devido à sobrecarga de comunicação, revelando o ponto em que o custo do paralelismo supera os seus benefícios.

Os resultados indicam que o modelo proposto é eficaz e adequado para aplicações que envolvem grandes volumes de dados e que se beneficiam da divisão de tarefas entre múltiplos processos. Além disso, o uso do ambiente de *HPC* do *NPAD/UFRN* permitiu explorar o comportamento do código em um cenário real de computação paralela, contribuindo para uma análise mais robusta e prática. Este experimento reforça a importância do paralelismo no contexto de aplicações científicas e a utilidade do *MPI* como ferramenta essencial para o desenvolvimento de soluções escaláveis.

## 5. ANEXOS

- Repositório no Github com o programa desenvolvido: https://github.com/ErnaneJ/parallel-programming-dca3703