Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 16: Comunicação Coletiva com MPI
Aluno: Daniel Bruno Trindade da Silva

1 Introdução

Este relatório tem como objetivo apresentar os conhecimentos adquiridos durante a realização da Tarefa 16 da disciplina de **Programação Paralela**. A atividade teve como objetivo mensurar o tempo de comunicação entre os processos usando ferramentas do MPI. Para isso será realizada a implementação e análise de um programa que simule a difusão de calor em uma barra 1D.

2 Enunciado

Implemente um programa MPI que calcule o produto $y = A \cdot x$, onde A é uma matriz M×N e x é um vetor de tamanho N. Divida a matriz A por linhas entre os processos com `MPI_Scatter`, e distribua o vetor x inteiro com `MPI_Bcast`. Cada processo deve calcular os elementos de y correspondentes às suas linhas e enviá-los de volta ao processo 0 com `MPI_Gather`. Compare os tempos com diferentes tamanhos de matriz e número de processos.

3 Desenvolvimento

O código desenvolvido tem como objetivo realizar o produto matriz-vetor de forma paralela utilizando a biblioteca MPI (Message Passing Interface). Para otimizar o uso de memória, optamos por utilizar o tipo de dado `uint8_t` (inteiro sem sinal de 8 bits), uma vez que os elementos da matriz e do vetor são pequenos, e o uso desse tipo reduz significativamente o consumo de memória. Essa escolha foi especialmente importante devido ao tamanho expressivo das matrizes processadas, que podem chegar a 32768 x 32768 elementos, representando mais de 2 GB de dados se fossem utilizados tipos de maior tamanho, como int ou double.

3.1 Comunicação entre Processos com MPI

Na implementação, utilizamos três operações fundamentais de comunicação da biblioteca MPI:

- `MPI_Bcast`: Esta função foi utilizada para realizar o broadcast do vetor x para todos os processos. Como o vetor x é necessário para o cálculo local de cada processo, foi essencial garantir que todos os processos tivessem acesso à mesma cópia deste vetor. O processo com rank 0 inicializa o vetor e, em seguida, o `MPI_Bcast` propaga os dados a todos os demais processos de forma eficiente.

- `MPI_Scatter`: Utilizamos esta função para distribuir as partes da matriz A entre os processos. Como cada processo é responsável pelo cálculo de uma parte do vetor resultado y, foi necessário dividir as linhas da matriz A igualmente entre eles. O `MPI_Scatter` possibilitou essa divisão automática e eficiente, enviando a cada processo apenas os dados necessários para o seu cálculo.

- `MPI_Gather`: Após o cálculo paralelo, cada processo obteve uma parte do vetor y. Utilizamos o `MPI_Gather` para coletar essas partes e reunir o vetor resultado completo no processo com rank 0, que então pode realizar o processamento final ou análise dos dados.

Além disso, para avaliar o desempenho, foi desenvolvido um script automatizado que permite executar o programa variando tanto o tamanho da matriz quanto o número de processos utilizados. Esse script facilitou a realização de diversos experimentos, possibilitando a coleta de dados para analisar o comportamento do código em diferentes cenários de execução, como o impacto da divisão de trabalho entre processos e a eficiência do paralelismo.

4 Resultados

Tamanho da Matriz	**Número de Processos**				
	2	**4**	**8**	**16**	**32**
2048×2048	0.007393	0.012269	0.010754	0.017535	0.032846
4096×4096	0.024350	0.019467	0.026092	0.018572	0.033169
8192×8192	0.223196	0.174600	0.149724	0.073371	0.080450
16384×16384	0.336517	0.201913	0.159433	0.076215	0.075465
32768×32768	1.314407	0.773659	0.510369	0.249389	0.193285

Table 1: Tempo de Execução (s) por Tamanho da Matriz e Número de Processos

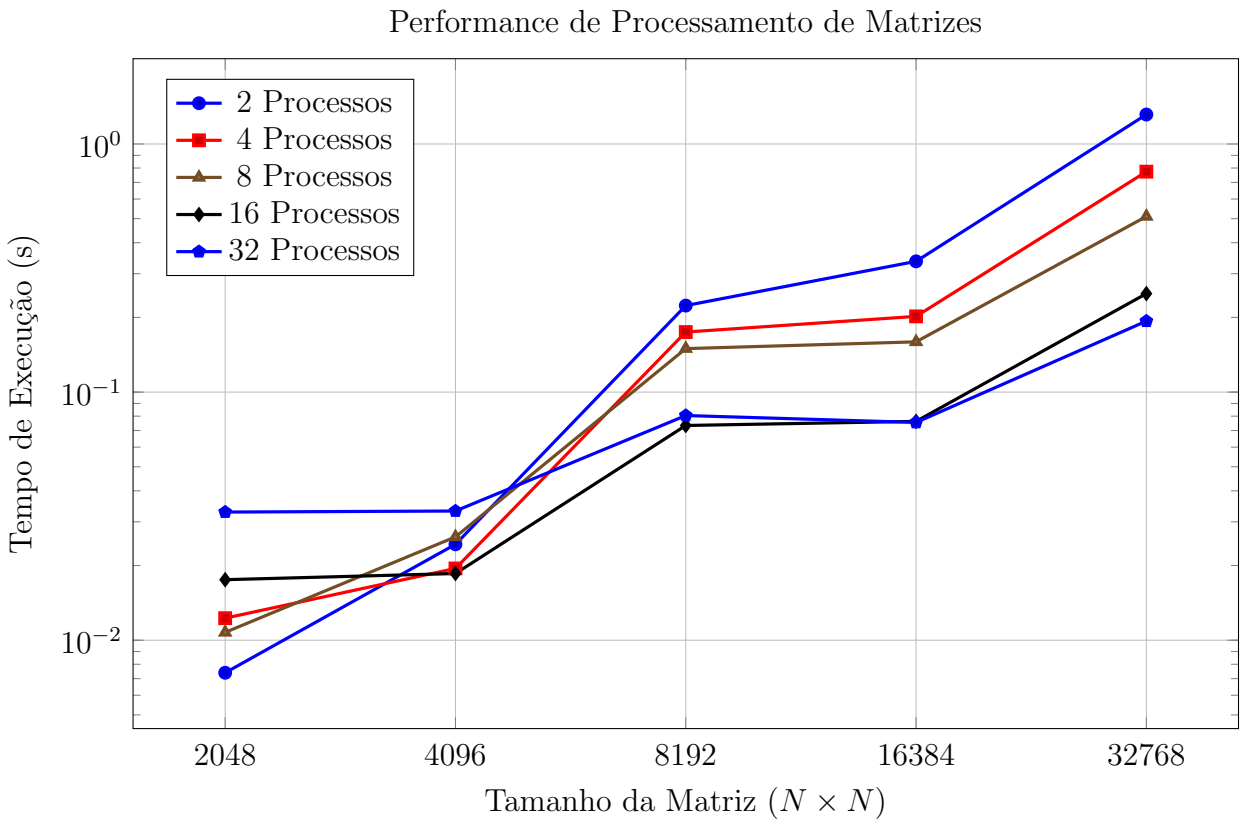

Figure 1: Gráfico de Linhas do Tempo de Execução pelo Tamanho da Matriz para Diferentes Números de Processos.

5 Analise dos Resultados

5.1 Ganhos de Desempenho

Observa-se que, conforme o número de processos aumenta, o tempo de execução tende a diminuir, principalmente para matrizes de maior dimensão. Por exemplo, para a matriz de 32.768 x 32.768, o tempo de execução reduziu de 1,31 segundos com 2 processos para apenas 0,19 segundos com 32 processos. Esse resultado evidencia a eficiência do paralelismo e a capacidade de dividir a carga computacional entre múltiplos processos, reduzindo significativamente o tempo total de execução.

Esse comportamento é característico de algoritmos que possuem grande potencial de paralelização, como o produto matriz-vetor, onde o trabalho pode ser dividido de forma relativamente equilibrada entre os processos. A função `MPI_Scatter` foi fundamental para distribuir de forma eficiente as linhas da matriz, enquanto o uso de `MPI_Bcast` garantiu que todos os processos tivessem acesso ao vetor x com um custo de comunicação relativamente baixo.

5.2 Efeito do Overhead de Comunicação

No entanto, para matrizes menores, como a de 2.048 x 2.048, o aumento do número de processos nem sempre resulta em uma melhora significativa no tempo de execução. Por exemplo, o tempo com 2 processos foi de 0,0073 segundos, enquanto com 32 processos foi de 0,0328 segundos, representando uma piora no desempenho.

Esse comportamento é explicado pelo overhead de comunicação: quando a quantidade de dados processados por cada processo é muito pequena, o custo das operações de comunicação (`MPI_Scatter`, `MPI_Bcast` e `MPI_Gather`) acaba sendo mais relevante do que o tempo gasto efetivamente no cálculo. Assim, para matrizes pequenas, o uso de um número elevado de processos não compensa, podendo inclusive degradar a performance.

Em síntese, os resultados obtidos demonstram que:

- O paralelismo com MPI proporciona ganhos de desempenho significativos para o produto matriz-vetor, principalmente com matrizes de grande dimensão.
- Existe um ponto de equilíbrio entre o número de processos e o tamanho da matriz, sendo que, para matrizes pequenas, o excesso de processos pode aumentar o tempo de execução devido ao overhead de comunicação.

6 Conclusão

A realização desta atividade permitiu consolidar conhecimentos importantes sobre a programação paralela utilizando a biblioteca MPI, com foco no uso de operações de comunicação coletiva como `MPI_Bcast`, `MPI_Scatter` e `MPI_Gather`. A implementação do produto matriz-vetor demonstrou na prática como a divisão de tarefas entre múltiplos processos pode acelerar significativamente a execução de operações computacionalmente intensivas.

Os experimentos realizados evidenciaram que o paralelismo é altamente eficaz para problemas de grande dimensão, proporcionando uma redução expressiva no tempo de execução à medida que o número de processos aumenta. Por outro lado, observou-se que para problemas menores, o *overhead* associado à comunicação entre processos pode superar os ganhos obtidos pela divisão do trabalho, o que reforça a importância de avaliar cuidadosamente o balanceamento entre carga computacional e custo de comunicação em aplicações paralelas.

Por fim, esta tarefa reforçou a importância do paralelismo na solução de problemas de larga escala e destacou o papel crucial das técnicas de comunicação coletiva para o desenvolvimento de aplicações eficientes e escaláveis.