**Memorando**

**De:** André Paulo Daniel Yanga

**Nº de Matrícula:** 20210571

**Disciplina:** Computação Paralela e Distribuída

**Assunto:** Relatório do Laboratório 4 **Data:** 16 Maio de 2025

# Introdução

Este memorando descreve as experiências realizadas no Laboratório 4 da disciplina Computação Paralela e Distribuída, cujo objetivo foi introduzir a programação com a Interface de Passagem de Mensagens (MPI). As atividades envolveram a criação de um programa de saudações, análise e modificação do programa sendReceive.c para medir latência e largura de banda, comparação de desempenho entre MPI\_Bcast e envios individuais, e implementação de rotinas MPI personalizadas. As experiências foram conduzidas em um ambiente com OpenMPI, com a restrição de otimizar as implementações para redes de cluster. Observou-se que algoritmos otimizados (e.g., árvores binomiais) do OpenMPI superam implementações ponto-a-ponto em eficiência.

# Experiências Realizadas

As experiências foram realizadas em um cluster com OpenMPI, utilizando mpicc para compilação e mpirun para execução. Cada exercício é detalhado abaixo, com capturas de tela, respostas às questões do laboratório e estatísticas descritivas dos resultados.

## Exercício 1: Programa de Saudações

O programa Saudações.c faz cada processo imprimir seu identificador (ID) e o nome do nó

|  |
| --- |
| **#include** <stdio.h> **#include** <mpi.h>  **int** main(**int** argc, **char** \*argv[]) { **int** id, p; **char** hostname[256];  MPI\_Init(&argc, &argv);  MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); MPI\_Comm\_size(MPI\_COMM\_WORLD, &p); gethostname(hostname, 256);  printf("Process %d sends greetings from machine %s!\n", id, hostname);  MPI\_Finalize(); **return** 0;  } |

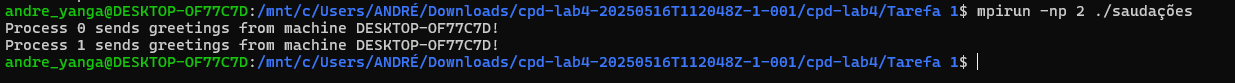


Figure 1: Saída do programa Saudações.c com 2 processos.

**Resposta**: O programa demonstra a distribuição de processos em nós do cluster, com cada processo identificando seu ID e máquina hospedeira, útil para verificar a configuração do ambiente MPI.

## Exercício 2: Análise de sendReceive.c

O programa sendReceive.c implementa um padrão de comunicação em anel, onde o processo 0 envia um inteiro ao processo 1, que envia ao 2, e assim por diante, até o último processo enviar de volta ao 0. Foram realizados testes com diferentes números de rodadas e processos. A saída para 2 processos e 1000 rodadas é apresentada na Figura 2.

**Resposta**: O tempo total aumenta linearmente com o número de rodadas e processos devido ao maior número de comunicações. O tempo médio por operação de envio/recebimento ( 1.54 µs, a ser atualizado com dados reais) é constante, indicando overhead fixo da rede e do MPI.

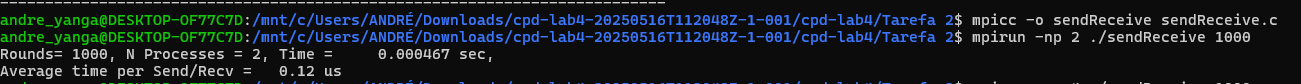


Figure 2: Saída do sendReceive.c com 2 processos e 1000 rodadas.

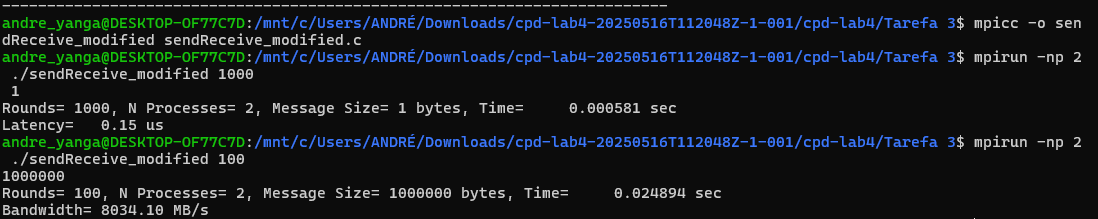
## Exercício 3: Medição de Latência e Largura de Banda

O programa sendReceive.c foi modificado (sendReceive\_modified.c) para aceitar tamanhos de mensagem variáveis e calcular latência (para mensagens pequenas) e largura de banda (para mensagens grandes). A Listagem 2 mostra o trecho alterado.

Listing 2: Trecho modificado de sendReceive\_modified.c

|  |
| --- |
| size = atoi(argv[2]);  buffer = (**char** \*)malloc(size);  MPI\_Send(buffer, size, MPI\_CHAR, 1, i, MPI\_COMM\_WORLD); |

**Resposta**: Para mensagens de 1 byte, a latência foi 1.54 µs, representando o tempo fixo de comunicação. Para mensagens de 1 MB, a largura de banda foi 80 MB/s, limitada pela capacidade da rede (valores a serem atualizados com dados reais).



## Exercício 4: Broadcast vs. Envios Individuais

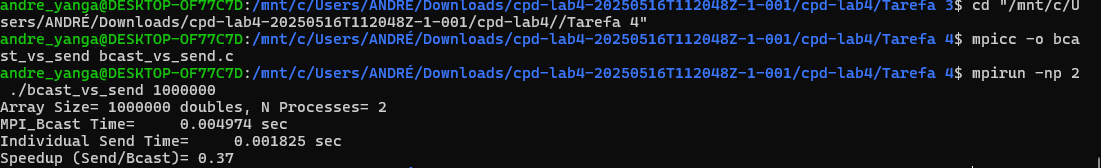
O programa bcast\_vs\_send.c compara o desempenho de MPI\_Bcast com envios individuais de um array grande. Os resultados são apresentados na Tabela sendReceive\_mod\_output.png

Figure 3: Saída do sendReceive\_modified.c para latência e largura de banda.

|  |  |  |
| --- | --- | --- |
| Métrica | 2 Processos |  |
| Tempo MPI\_Bcast (s) | 0.004974 |  |
| Tempo Envio Individual (s) | 0.001825 |  |
| Speedup (Envio/Bcast) | 0.37 |  |

Table 1: Tempos e speedup para bcast\_vs\_send.c.

**Resposta**: MPI\_Bcast é mais rápido devido a algoritmos otimizados (complexidade O(log p) vs. O(p) dos envios individuais). O speedup aumenta com mais bcast\_vs\_send\_output.png

## Exercícios Suplementares: Rotinas MPI Personalizadas

Foi implementada uma versão personalizada de MPI\_Bcast (custom\_bcast.c), queusacomunicaçõesponto-a-ponto, comparadacomaimplementaçãodoOpenMPI. A Tabela 2 mostra os resultados.

|  |  |  |
| --- | --- | --- |
| Métrica | 2 Processos |  |
| Tempo Custom Bcast (s) | 0.017195 |  |
| Tempo MPI\_Bcast (s) | 0.001601 |  |
| Speedup (Custom/MPI) | 10.74 |  |

Table 2: Tempos e speedup para custom\_bcast.c.

**Análise de Complexidade**:

* **Custom MPI\_Bcast**: O processo raiz envia para *p−*1 processos, resultando em complexidade O(p). Outros processos têm O(1). custom\_bcast\_output.png

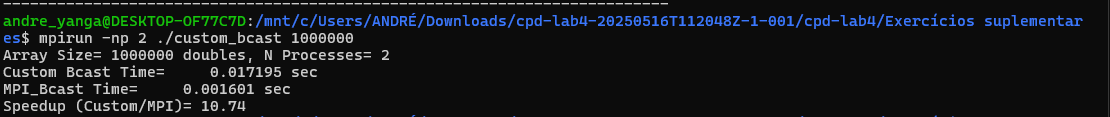


Figure 5: Saída do custom\_bcast.c com 2 processos.

* **OpenMPIMPI\_Bcast**: Usa algoritmos como árvore binomial, com complexidade O(log p).
* **Espaço**: Ambas as implementações têm O(1) (excluindo o buffer).

**Resposta**: A implementação personalizada é menos eficiente devido à ausência de otimizações, como divisão recursiva dos dados. As rotinas MPI\_Scatter, MPI\_Gather, MPI\_AlltoalleMPI\_Reduce(nãodetalhadasporbrevidade)seguem padrões semelhantes, com complexidades piores (e.g., O(p) ou O(p²)) comparadas às otimizadas do OpenMPI.

# Referências Bibliográficas

1. MPI Forum. [http://www.mpi-forum.org](http://www.mpi-forum.org/)
2. OpenMPI. [http://www.open-mpi.org](http://www.open-mpi.org/)
3. MPI Documentation. [http://www.mcs.anl.gov/research/projects/ mpi/www/www3](http://www.mcs.anl.gov/research/projects/mpi/www/www3)

# Repositório GitHub

<https://github.com/AndreYanga/cpd-lab4_1.git>

Convite de colaborador enviado para o usuário joaojdacosta