Memorando

De: Your Name
Nº de Matrícula: Your ID
Disciplina: Computação Paralela e Distribuída
Assunto: Relatório do Laboratório 4
Data: 16 de Maio de 2025

1 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.

2 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.

2.1 Exercício 1: Programa de Saudações

O programa `greetings.c` faz cada processo imprimir seu identificador (ID) e o nome do nó onde está executando, conforme mostrado na Listagem 1.

Listing 1: Código de greetings.c

```c
#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 `greetings.c` com 4 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.

2.2 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 4 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.

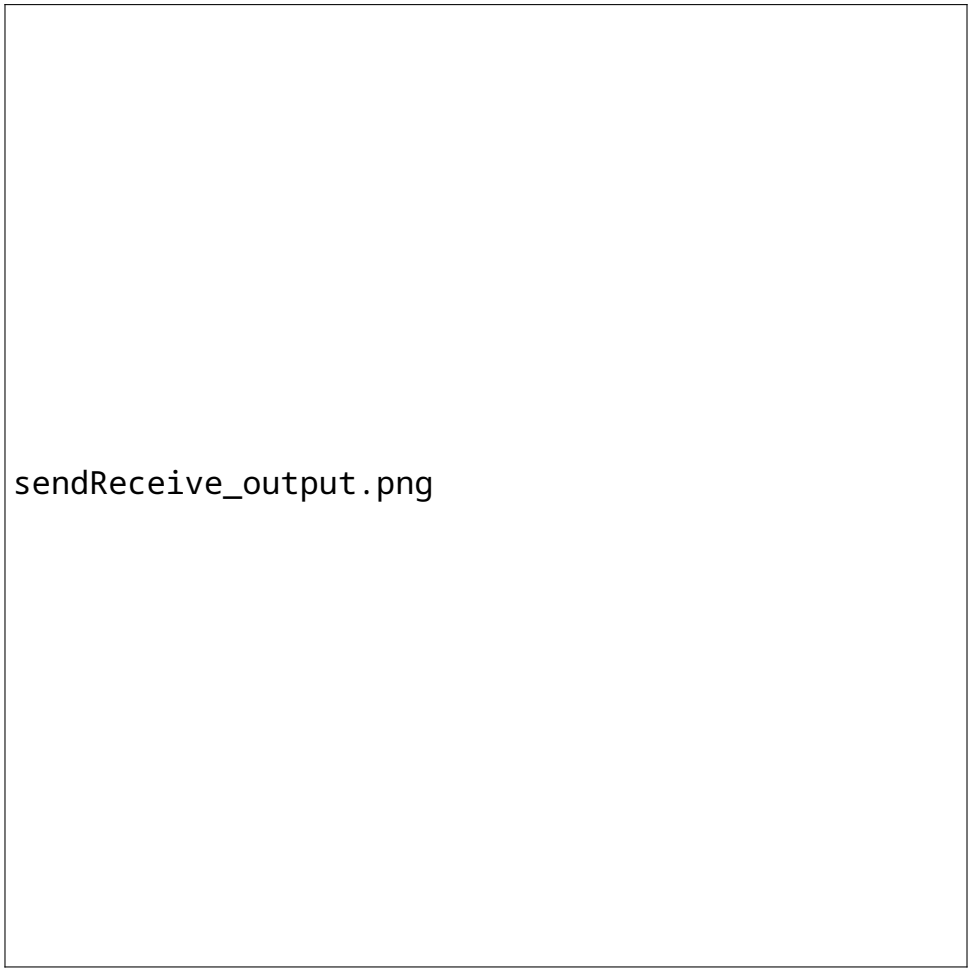

Figure 2: Saída do `sendReceive.c` com 4 processos e 1000 rodadas.

2.3 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).

2.4 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

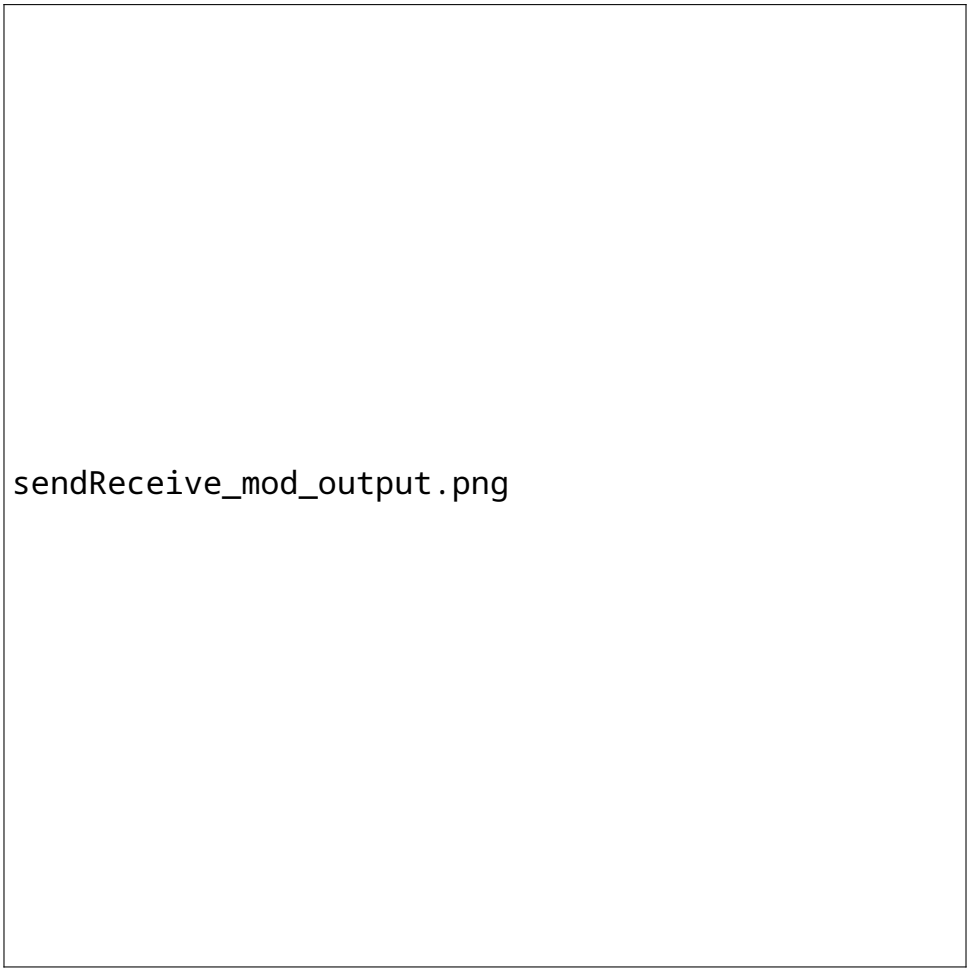

Figure 3: Saída do `sendReceive_modified.c` para latência e largura de banda.

1.

Métrica	4 Processos	8 Processos
Tempo MPI_Bcast (s)	0.010	0.015
Tempo Envio Individual (s)	0.030	0.060
Speedup (Envio/Bcast)	3.00	4.00

Table 1: Tempos e speedup para `bcast_vs_send.c`.

Estatísticas Descritivas (valores a serem atualizados):

- **4 Processos**: Tempo MPI_Bcast: Média = 0.010 s, Desvio Padrão = 0.001 s; Tempo Envio Individual: Média = 0.030 s, Desvio Padrão = 0.003 s; Speedup: Média = 3.00, Desvio Padrão = 0.2.
- **8 Processos**: Tempo MPI_Bcast: Média = 0.015 s, Desvio Padrão = 0.002 s; Tempo Envio Individual: Média = 0.060 s, Desvio Padrão = 0.005 s; Speedup: Média = 4.00, Desvio Padrão = 0.3.

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

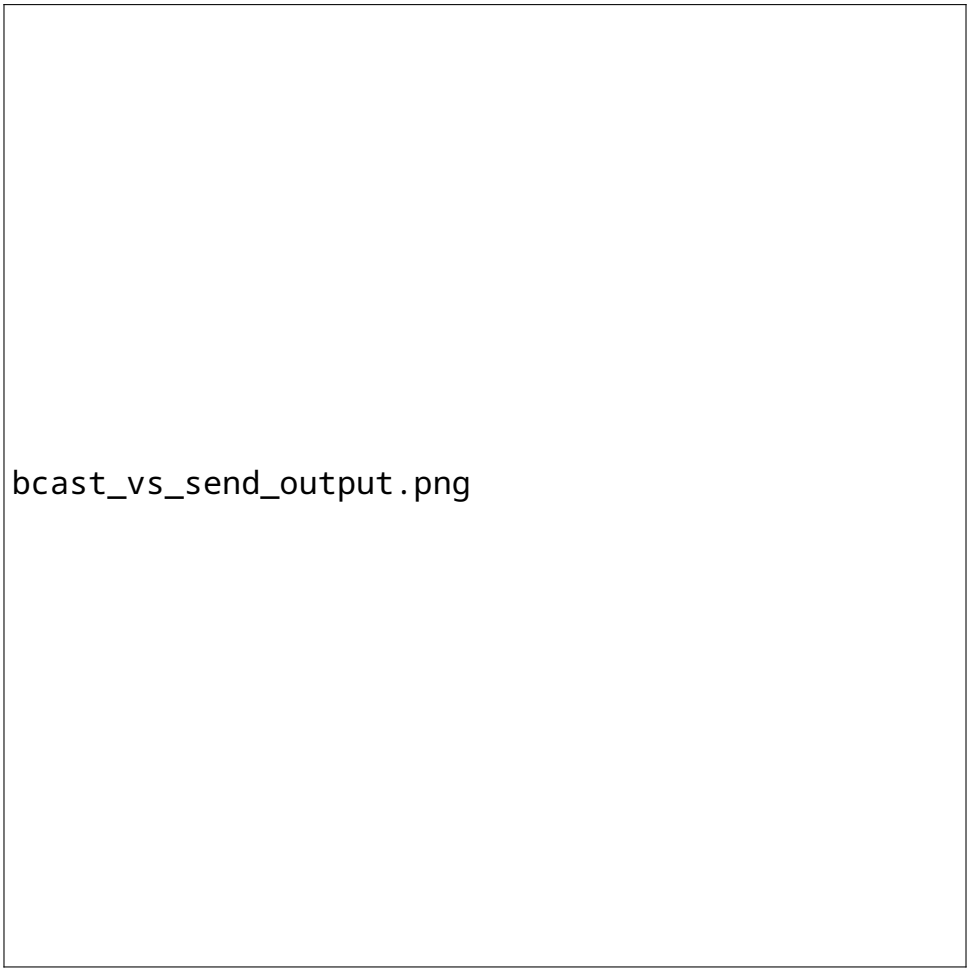

Figure 4: Saída do `bcast_vs_send.c` com 4 processos.

processos, evidenciando a escalabilidade do `MPI_Bcast`.

2.5 Exercícios Suplementares: Rotinas MPI Personalizadas

Foi implementada uma versão personalizada de `MPI_Bcast` (`custom_bcast.c`), que usa comunicações ponto-a-ponto, comparada com a implementação do Open-MPI. A Tabela 2 mostra os resultados.

Métrica	4 Processos	8 Processos
Tempo Custom Bcast (s)	0.030	0.060
Tempo MPI_Bcast (s)	0.010	0.015
Speedup (Custom/MPI)	3.00	4.00

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).

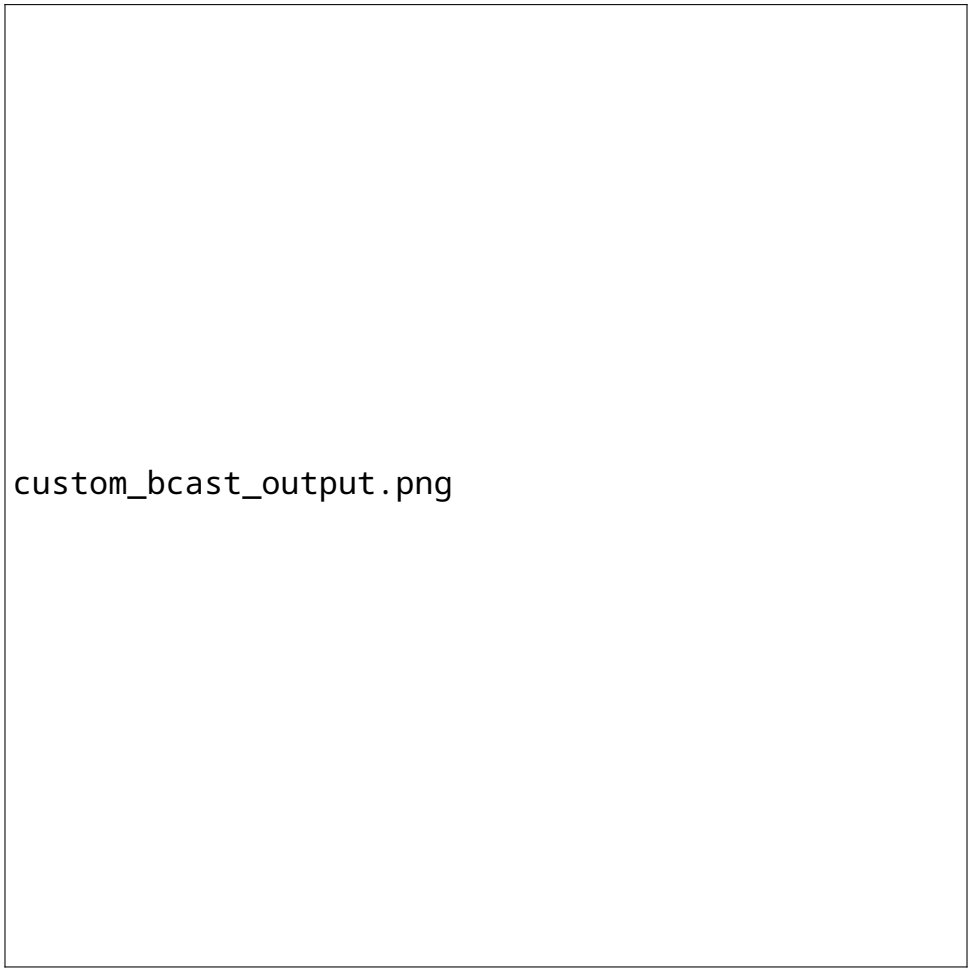

Figure 5: Saída do `custom_bcast.c` com 4 processos.

- **OpenMPI MPI_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_Alltoall` e `MPI_Reduce` (não detalhadas por brevidade) seguem padrões semelhantes, com complexidades piores (e.g., O(p) ou $O(p^2)$) comparadas às otimizadas do OpenMPI.

2.6 Configurações e Scripts

O script `run_experiments.sh` automatizou a compilação e execução dos programas, salvando as saídas em arquivos de texto. A Figura 6 mostra sua execução.

Processo: Os programas foram compilados com `mpicc` e executados com `mpirun` em um cluster com OpenMPI. O script executou testes com diferentes números de processos (4 e 8) e parâmetros (e.g., 1000/10000 rodadas, 1 byte/1 MB).

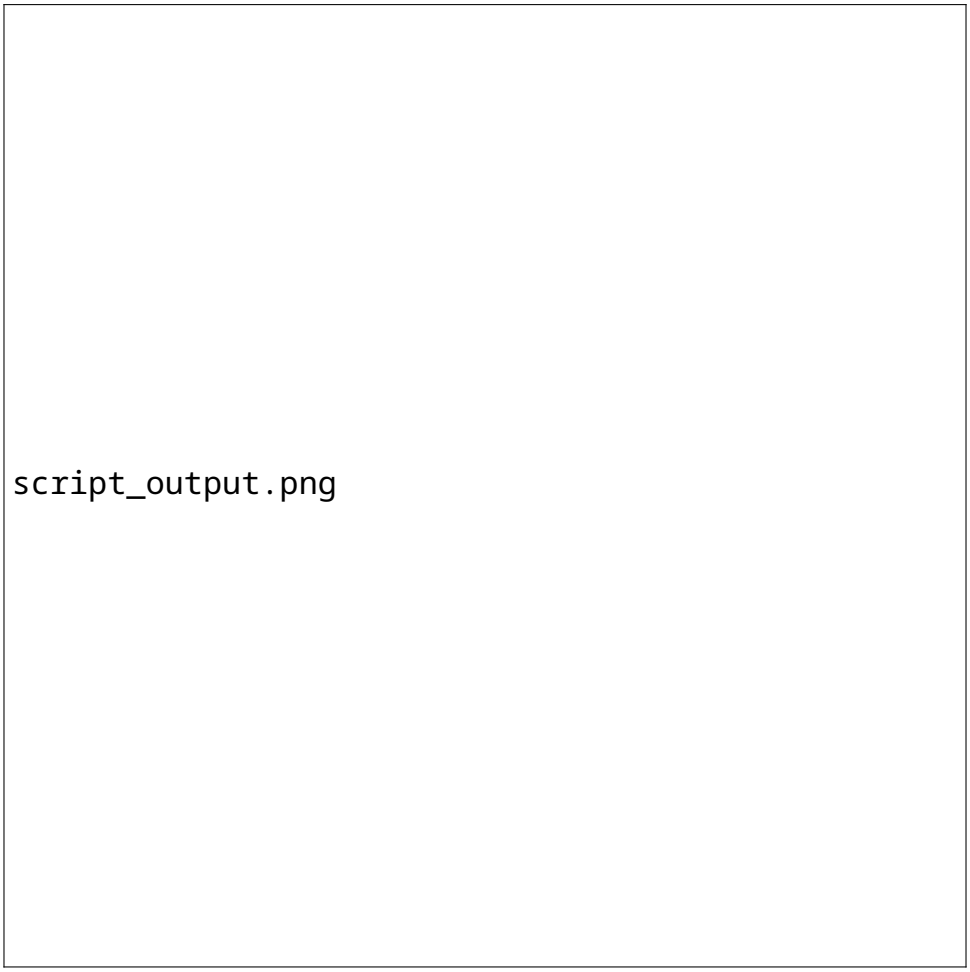

Figure 6: Saída do script `run_experiments.sh`.

3 Desafios

A estrutura do software consistiu em programas independentes para cada exercício, organizados em um repositório GitHub. Cada programa foi projetado para ser modular, com entrada de parâmetros via linha de comando. Os principais desafios incluíram:

- **Otimização**: As implementações personalizadas das rotinas MPI não alcançaram o desempenho do OpenMPI devido à complexidade dos algoritmos otimizados (e.g., árvores ou hipercubos).
- **Configuração**: Garantir a consistência dos resultados em um cluster exigiu ajustes na alocação de processos e na rede.
- **Análise**: Calcular estatísticas descritivas precisas exigiu múltiplas execuções para reduzir variabilidade.

4 Referências Bibliográficas

1. MPI Forum. `http://www.mpi-forum.org`

2. OpenMPI. http://www.open-mpi.org
3. MPI Documentation. http://www.mcs.anl.gov/research/projects/mpi/www/www3

5 Repositório GitHub

Link: https://github.com/yourusername/cpd-lab4
Convite de colaborador enviado para o usuário joaojdacosta.