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

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

```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;
}
```

```
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 1$ mpirun -np 2 ./saudações
Process 0 sends greetings from machine DESKTOP-OF77C7D!
Process 1 sends greetings from machine DESKTOP-OF77C7D!
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 1$
```

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.

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 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.

```
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 2$ mpicc -o sendReceive sendReceive.c
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 2$ mpirun -np 2 ./sendReceive 1000
Rounds= 1000, N Processes = 2, Time =     0.000467 sec,
Average time per Send/Recv =   0.12 us
```

Figure 2: Saída do sendReceive.c com 2 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.

```
size = atoi(argv[2]);
buffer = (char *)malloc(size);
MPI_Send(buffer, size, MPI_CHAR, 1, i, MPI_COMM_WORLD);
```

Listing 2: Trecho modificado de sendReceive_modified.c

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).

```
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 3$ mpicc -o sen
dReceive_modified sendReceive_modified.c
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 3$ mpirun -np 2
 ./sendReceive_modified 1000
 1
Rounds= 1000, N Processes= 2, Message Size= 1 bytes, Time=     0.000581 sec
Latency=   0.15 us
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 3$ mpirun -np 2
 ./sendReceive_modified 100
1000000
Rounds= 100, N Processes= 2, Message Size= 1000000 bytes, Time=     0.024894 sec
Bandwidth= 8034.10 MB/s
```

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

sendReceive_mod_output.png

```
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 3$ cd "/mnt/c/U
sers/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4//Tarefa 4"
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 4$ mpicc -o bca
st_vs_send bcast_vs_send.c
andre_yanga@DESKTOP-OF77C7D:/mnt/c/Users/ANDRÉ/Downloads/cpd-lab4-20250516T112048Z-1-001/cpd-lab4/Tarefa 4$ mpirun -np 2
 ./bcast_vs_send 1000000
Array Size= 1000000 doubles, N Processes= 2
MPI_Bcast Time=    0.004974 sec
Individual Send Time=    0.001825 sec
Speedup (Send/Bcast)= 0.37
```

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

2.5 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 $O(p^2)$) comparadas às otimizadas do OpenMPI.

2 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

3 Repositório GitHub

https://github.com/AndreYanga/cpd-lab4.git

Convite de colaborador enviado para o usuário joaojdacosta