Relatório: Programação em memória distribuída

Aluno: Cristovão Lacerda Cronje

1. Introdução

Nesta atividade, realizamos uma simulação de comunicação ponto a ponto, usando a biblioteca MPI (Message Passing Interface), em um único nó com múltiplos processos MPI, representando um ambiente de memória distribuída. O programa foi projetado para medir o tempo de comunicação entre dois processos MPI — utilizando um padrão de comunicação "ping-pong" — com diferentes tamanhos de mensagens, variando de 8 bytes até 1MB. Esta análise nos permite estudar o comportamento da latência e da largura de banda na troca de mensagens entre processos distribuídos.

2. Objetivos

- Compreender os conceitos básicos de comunicação ponto a ponto em MPI.
- Utilizar corretamente funções fundamentais como MPI_Init, MPI_Finalize, MPI_Comm_rank, MPI_Comm_size, MPI_Send, MPI_Recv e MPI_Wtime.
- Analisar experimentalmente o impacto do tamanho da mensagem no tempo de comunicação.
- Identificar empiricamente os regimes onde a **latência** domina e onde a **largura de banda** se torna o fator limitante.

3. Metodologia

3.1 Estrutura do Código

O código foi escrito em C com uso da biblioteca MPI. O programa é executado com **exatamente 2 processos**:

- O processo de **rank 0** envia uma mensagem ao processo de **rank 1**.
- O processo de **rank 1** imediatamente responde com a mesma mensagem.
- A troca é repetida 100000 vezes para garantir a estabilidade dos dados.
- O tempo total dessas trocas é medido com MPI_Wtime.

Para medir o tempo de comunicação entre dois processos MPI, foi implementado um teste de **ping-pong**, onde o processo de rank 0 envia uma mensagem de tamanho variável ao processo de rank 1, que a retorna imediatamente. Esse procedimento é repetido 1000 vezes para cada tamanho de mensagem, variando de 8 bytes a 1 megabyte. O tempo de ida e volta (*round-trip*) é medido com MPI_Wtime() e a média é calculada para garantir estabilidade nos resultados.

Antes de cada medição, é utilizada uma barreira de sincronização (MPI_Barrier) para garantir que ambos os processos iniciem a troca ao mesmo tempo. A função MPI_Send é bloqueante, o que permite observar os efeitos do *buffering* e da sincronização de forma clara nos resultados obtidos.

```
// Loop principal de testes
for (int msg_size = MIN_MSG_SIZE; msg_size <= MAX_MSG_SIZE; msg_size *= 2) {
    double total_time = 0.0;
    // Repete o teste várias vezes para obter média estável
    for (int rep = 0; rep < NUM_REPETITIONS; rep++) {
        MPI_Barrier(MPI_COMM_WORLD);
        double start = MPI_Wtime();
        if (rank == 0) {
            // Processo 0 envia e recebe de volta
            MPI_Send(message, msg_size, MPI_BYTE, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(message, msg_size, MPI_BYTE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        } else {
            // Processo 1 recebe e envia de volta
            MPI_Recv(message, msg_size, MPI_BYTE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(message, msg_size, MPI_BYTE, 0, 0, MPI_COMM_WORLD);
        }
        total_time += (MPI_Wtime() - start);
    }
    // Apenas o rank 0 imprime os resultados
    if (rank == 0) {
        double avg_time = (total_time / NUM_REPETITIONS) * 1e6; // µs
        printf("%-15d %15.2f\n", msg_size, avg_time);
    }
}
                    (char [44])"----------\n"
```

Apenas o processo de rank 0 imprime os resultados formatados ao final de cada conjunto de repetições.

3.2 Cláusulas utilizadas no código

Cláusula MPI	Função	Efeito
MPI_Init	Inicializa o ambiente MPI	Necessário antes de qualquer chamada MPI
MPI_Finalize	Finaliza o ambiente MPI	Libera recursos e finaliza a aplicação
MPI_Comm_rank	Obtém o rank do processo	Usado para diferenciar o comportamento dos processos 0 e 1
MPI_Comm_size	Número total de processos	Verifica se o programa está sendo executado com exatamente dois processos
MPI_Send	Envia uma mensagem	Comunicação **bloqueante**: pode bloquear até o início do recebimento pelo destino
MPI_Recv	Recebe uma mensagem	Comunicação **bloqueante**: bloqueia até o recebimento completo da mensagem
MPI_Barrier	Sincroniza os processos	Garante que ambos os processos iniciem a troca ao mesmo tempo
MPI_Wtime	Mede tempo de execução	Fornece tempo com precisão de microssegundos

3.3 Considerações sobre Buffer e Comunicação

Durante a execução do código utilizando as funções MPI_Send e MPI_Recv, é essencial compreender o comportamento dessas operações em relação ao *buffering* interno e ao bloqueio na comunicação.

- **Funções bloqueantes:**
  As funções MPI_Send e MPI_Recv empregadas no experimento são do tipo **bloqueante** (*blocking*), o que significa que:
  - MPI_Send apenas retorna (continua a execução) quando é seguro reutilizar o buffer de envio, o que pode ocorrer após o início ou a conclusão da recepção pela parte do processo destinatário.
  - MPI_Recv só retorna após o recebimento completo da mensagem.

- **Mensagens pequenas e *buffering*:**
  Para mensagens de tamanho reduzido (como 1B, 2B, 4B etc.), a implementação do MPI pode optar por armazenar temporariamente os dados em um *buffer* interno. Isso permite que MPI_Send continue rapidamente, mesmo que o processo receptor ainda não tenha chamado MPI_Recv.
  Esse mecanismo resulta em tempos de comunicação baixos e quase constantes para mensagens pequenas, como evidenciado nos dados experimentais.

- **Mensagens grandes e bloqueio completo:**
  A partir de um determinado tamanho de mensagem (por exemplo, 8KB ou 16KB, dependendo da implementação do MPI e da arquitetura utilizada e se a mensagem **excede esse limite**, o MPI **não a envia imediatamente,** ele **bloqueia** o processo emissor **até que o receptor esteja pronto**, usando o chamado **rendezvous protocol)**, o *buffering* interno deixa de ser aplicado. Nesses casos:

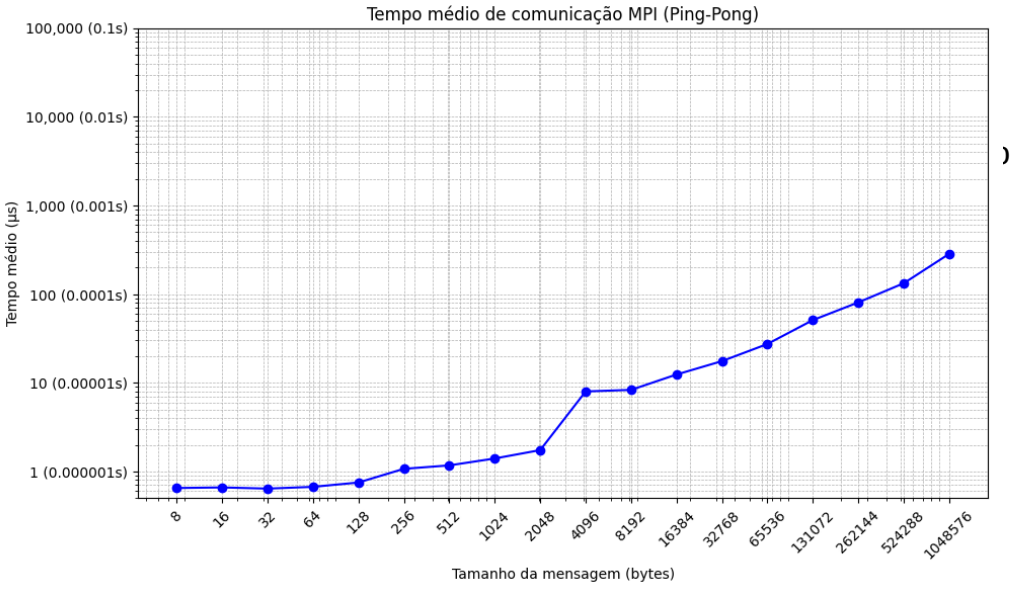
  - MPI_Send permanece bloqueado até que MPI_Recv esteja ativo e apto a receber a mensagem.
  - A comunicação torna-se completamente síncrona, aumentando o tempo de envio, pois exige que ambos os processos estejam sincronizados — o que pode não ocorrer imediatamente, devido à variação no escalonamento do sistema operacional.

- **Efeitos no tempo de execução:**
  A análise dos resultados mostra que o tempo de execução cresce consideravelmente para mensagens maiores. Esse comportamento indica o ponto em que o MPI deixa de aplicar *buffering*, fazendo com que a comunicação dependa diretamente da sincronização entre os processos. Essa mudança afeta significativamente o desempenho de aplicações paralelas que manipulam grandes volumes de dados.

4. Resultados e Análise

4.1 Tabela de Tempo de Execução (µs) e Gráfico (Tempo vs Tamanho da Mensagem)

Tamanho da Mensagem (bytes)	Tempo Médio (µs)
8	0.65
16	0.66
32	0.64
64	0.67
128	0.75
256	1.07
512	1.17
1024	1.40
2048	1.74
4096	7.99
8192	8.32



Tamanho da Mensagem (bytes)	Tempo Médio (µs)
16384	12.40
32768	17.56
65536	27.40
131072	50.98
262144	80.73
524288	132.77
1048576	284.88

- Crescimento **lento** no tempo para mensagens **pequenas**.
- Crescimento **quase linear** para mensagens maiores, indicando que a **largura de banda** passa a ser o fator limitante.

4.3 Análise Qualitativa

- **Regime dominado por latência**: Para mensagens pequenas (até 1KB), o tempo de troca permanece quase constante. Isso mostra que o custo fixo da troca de mensagens (latência) é dominante.
- **Regime dominado por largura de banda**: A partir de 4KB, o tempo começa a crescer proporcionalmente ao tamanho da mensagem. Isso indica que o tempo de comunicação é limitado pela taxa de transmissão de dados — ou seja, pela largura de banda do canal de comunicação.

5. Conclusão

Foram observados dois regimes distintos de desempenho na comunicação MPI, que são influenciados por dois principais fatores: **latência** e **largura de banda**.

- **Latência domina para mensagens pequenas**:
  Para mensagens de tamanho pequeno, o tempo total de comunicação é praticamente constante e muito próximo do tempo mínimo necessário para iniciar a transferência dos dados. Isso ocorre porque a latência corresponde ao atraso fixo para o início da comunicação, independentemente do tamanho da mensagem — é o tempo que o sistema leva para preparar, enviar e confirmar o recebimento de uma mensagem. Portanto, para mensagens pequenas, o custo da latência se torna o fator limitante, e o tempo de comunicação não aumenta significativamente conforme o tamanho da mensagem varia dentro desse intervalo.
- **Largura de banda limita para mensagens grandes**:
  Para mensagens maiores, o tempo de comunicação cresce proporcionalmente ao tamanho dos dados transmitidos. Isso acontece porque, após o custo fixo da latência, a transferência da quantidade efetiva de dados é limitada pela **largura de banda** do sistema — ou seja, a quantidade máxima de dados que pode ser transmitida por unidade de tempo no canal de comunicação. Quando o volume da mensagem aumenta, o tempo gasto para transferir os bytes adicionais se torna dominante. A largura de banda funciona como um gargalo que impede que os dados sejam enviados mais rapidamente, mesmo que a latência permaneça a mesma. Assim, para mensagens grandes, o desempenho é limitado pela capacidade máxima do canal de comunicação, fazendo o tempo crescer quase linearmente com o tamanho da mensagem.