Relatório: Sobreposição de computação e comunicação (Versões de Simulação de Difusão de Calor com MPI)

Aluno: Cristovão Lacerda Cronje

1. Introdução

Este relatório tem como objetivo analisar o desempenho de três versões distintas de um algoritmo de simulação de difusão de calor unidimensional (1D), utilizando a interface MPI (Message Passing Interface) para execução paralela em dois nós no cluster NPAD, com objetivo de comparar os tempos totais de execução, os tempos de computação local e os tempos de comunicação entre processos. As versões implementadas são:

1. **MPI bloqueante** (usando MPI_Send/MPI_Recv)
2. **MPI não bloqueante com MPI_Wait** (usando MPI_Isend/MPI_Irecv)
3. **MPI não bloqueante com MPI_Test**

2. Metodologia

A simulação resolve a equação de difusão de calor unidimensional, dividindo a barra entre os processos MPI. Cada processo cuida de uma parte do domínio e precisa trocar informações de fronteira com seus vizinhos.

Cada versão do programa foi implementada utilizando as funções principais do MPI:

- **MPI_Send / MPI_Recv**: realizam **comunicação bloqueante**. O processo emissor ou receptor fica bloqueado até que a operação termine. É simples de usar, mas pode causar ociosidade se a comunicação demorar.
- **MPI_Isend / MPI_Irecv**: realizam **comunicação não bloqueante**, permitindo que a computação prossiga mesmo enquanto a comunicação ainda está em andamento. Isso abre espaço para sobreposição de computação e comunicação.
- **MPI_Wait**: é utilizado para aguardar a finalização de uma operação não bloqueante iniciada com Isend ou Irecv. É essencial garantir que os dados recebidos sejam válidos antes de utilizá-los.
- **MPI_Test**: verifica de forma não bloqueante se uma operação foi concluída. Ideal para sobreposição avançada, mas pode causar overhead se utilizado em loops com verificação constante (polling intenso).

Além disso, o tempo foi medido usando MPI_Wtime() para separar as fases de computação e comunicação. O número total de processos, seu identificador (rank), e a troca de fronteiras entre vizinhos foram tratados com MPI_Comm_size, MPI_Comm_rank e comunicação ponto a ponto.

2.1 Conceitos Importantes

2.1.1 Subdomínio

É a parte da barra (domínio 1D) da qual um processo é responsável durante a simulação. O domínio total da simulação é dividido entre os processos, e cada um calcula a evolução térmica em sua fatia.

2.1.2 Troca de fronteiras

É a comunicação entre processos vizinhos que ocorre a cada iteração, permitindo que cada processo conheça a temperatura nas extremidades de seus vizinhos. Isso é essencial para manter a consistência da simulação física.

2.1.3 Células fantasmas (ghost cells)

São posições extras nas extremidades do vetor local de cada processo, usadas para armazenar temporariamente os valores recebidos dos vizinhos. Elas não fazem parte da solução do processo, mas permitem que o cálculo na borda seja feito corretamente.

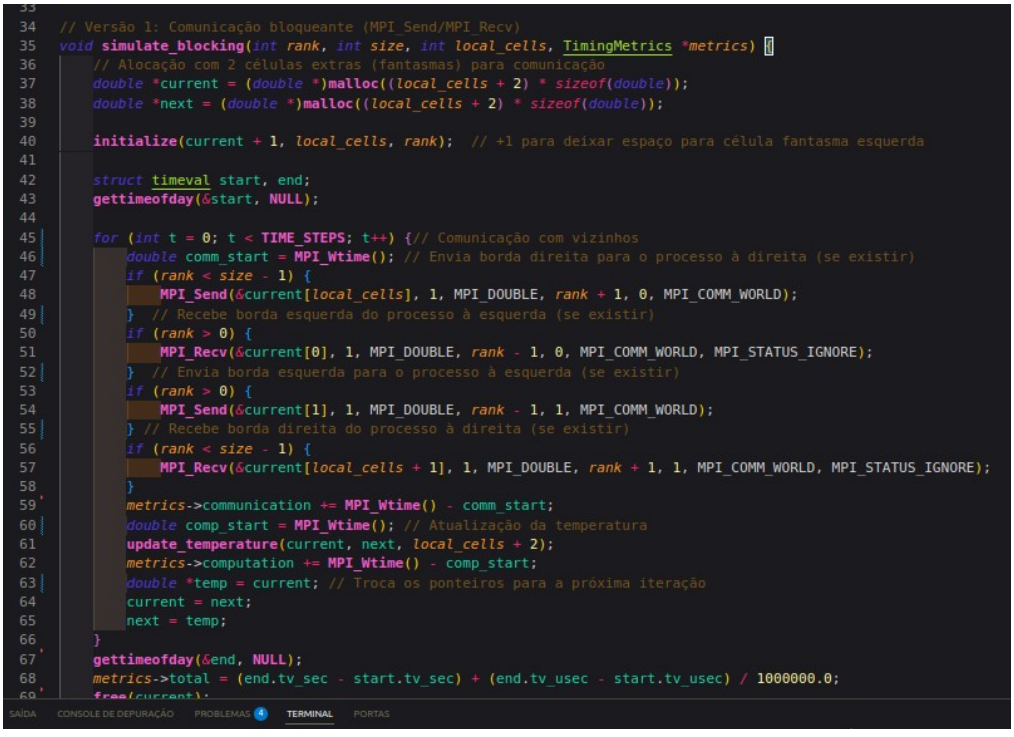
2.1.4 Halo

É o conjunto formado pelas células fantasmas. Ele representa a "borda de comunicação" do subdomínio e é necessário para que as operações de stencil (como a média das vizinhas) sejam corretas mesmo nas extremidades do subdomínio.

3. Código

Cada versão do código utiliza a mesma lógica de computação, variando apenas no modo como os processos trocam dados nas bordas de seus subdomínios. As três versões (bloqueante, não bloqueante com Wait e com Test) seguem os mesmos princípios de paralelização e sincronização, alterando somente o modo de realizar a comunicação:

- **Bloqueante:** os processos chamam MPI_Send e MPI_Recv e esperam a conclusão de cada chamada antes de prosseguir. Isso garante consistência, mas introduz períodos de espera desnecessários.



- **Não bloqueante (Wait):** utilizam MPI_Isend e MPI_Irecv para iniciar a comunicação e realizam computação local em paralelo. Em seguida, MPI_Wait assegura que a comunicação foi concluída antes que os dados recebidos sejam utilizados.
- **Não bloqueante (Test):** semelhante à versão com Wait, mas utiliza MPI_Test em um loop para verificar se a comunicação terminou. Essa abordagem tenta maximizar a sobreposição, mas, se MPI_Test for chamado repetidamente em um laço de verificação (busy waiting), pode introduzir um custo adicional de processamento (overhead).

4. Análise dos Resultados e Discussão

Tabela de resultados (em segundos)

Versão	Total	Computação	Comunicação
Bloqueante	1.501	1.488	0.013
Não bloqueante (Wait)	1.418	1.414	1.418
Não bloqueante (Test)	1.710	1.709	1.710

Interpretação dos tempos

- **Versão bloqueante:** embora simples e segura, obriga os processos a esperarem a comunicação terminar antes de continuar. Isso gera períodos de inatividade que afetam negativamente a performance, principalmente quando o volume de comunicação cresce.
- **Versão com MPI_Wait:** foi a mais eficiente, com uma redução de 5.56% no tempo total em comparação à versão bloqueante. Essa melhoria ocorre porque a computação e a comunicação ocorrem parcialmente em paralelo, diminuindo o tempo ocioso dos processos.
- **Versão com MPI_Test:** Foi menos eficiente. Embora o objetivo do Test seja permitir a verificação assíncrona do término da comunicação, sua implementação requer que o processo verifique repetidamente se a operação foi concluída, o que introduz overhead e aumenta o tempo total.

5. Conclusões

- A versão **não bloqueante com MPI_Wait** demonstrou ser a mais eficiente neste caso, permitindo um uso mais eficaz do tempo através da sobreposição entre computação e comunicação.
- A versão **bloqueante** tem desempenho razoável, mas apresenta tempo ocioso durante as trocas de mensagens.
- A versão com **MPI_Test** foi a menos eficiente devido à sobrecarga do teste frequente sem cálculo efetivo. Pode ser mais vantajosa em cenários onde **a** comunicação demora relativamente mais que a computação. ou quando se consegue balancear bem a frequência das verificações para não chamar MPI_Test em excesso.

Portanto, para simulações semelhantes que permitem computação local durante a comunicação, a abordagem com MPI_Isend/Irecv combinada com MPI_Wait é altamente recomendada para melhor aproveitamento dos recursos de paralelismo de MPI.