Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 15: Difusão de calor usando MPI
Aluno: Daniel Bruno Trindade da Silva

1 Introdução

Este relatório tem como objetivo apresentar os conhecimentos adquiridos durante a realização da Tarefa 15 da disciplina de **Programação Paralela**. A atividade teve como objetivo mensurar o tempo de comunicação entre os processos usando ferramentas do MPI. Para isso será realizada a implementação e análise de um programa que simulasse a difusão de calor em uma barra 1D.

2 Enunciado

Implemente uma simulação da difusão de calor em uma barra 1D, dividida entre dois ou mais processos MPI. Cada processo deve simular um trecho da barra com células extras para troca de bordas com vizinhos. Implemente três versões: uma com `MPI_Send/MPI_Recv`, outra com `MPI_Isend/MPI_Irecv` e `MPI_Wait`, e uma terceira usando `MPI_Test` para atualizar os pontos internos enquanto aguarda a comunicação. Compare os tempos de execução e discuta os ganhos com sobreposição de comunicação e computação.

3 Desenvolvimento

Para atender aos requisitos da atividade, foram desenvolvidas três versões de um programa em C com MPI para simular a difusão de calor unidimensional.

Em todas as implementações, a barra 1D de tamanho N é dividida entre os P processos MPI, onde cada processo é responsável por calcular a temperatura de um segmento de N/P células. Para permitir a troca de informações de contorno com os processos vizinhos, cada processo aloca duas células extras (ghost cells), uma no início e outra no fim de seu subdomínio local. A inicialização da barra atribui um valor de temperatura elevado a uma célula central da barra global e zero às demais. As condições de contorno globais da barra são fixas em 0.0. O tempo de execução de cada simulação é medido utilizando `MPI_Wtime()` e o tempo máximo entre todos os processos é obtido com `MPI_Reduce`.

As três versões implementadas diferem na forma como a comunicação das células de borda é realizada da seguinte forma:

- **Versão 1:** - Nesta implementação, a troca de dados das células de borda é realizada utilizando as funções bloqueantes `MPI_Send` e `MPI_Recv`. A natureza bloqueante dessas chamadas implica que um processo pode ficar ocioso enquanto espera a conclusão de uma operação de envio ou recebimento.

- **Versão 2:** - A segunda implementação utiliza as versões não bloqueantes das chamadas de comunicação, `MPI_Isend` e `MPI_Irecv`, para a troca das células de borda. Após iniciar todas as operações de comunicação necessárias, a função `MPI_Waitall` é chamada. Esta função bloqueia o processo até que todas as comunicações não bloqueantes (tanto envios quanto recebimentos) especificadas sejam concluídas. Somente após o retorno de `MPI_Waitall`, o cálculo da nova temperatura para todas as células locais é efetuado. Nesta abordagem, embora as operações sejam não bloqueantes, a computação dos pontos da malha só ocorre após a finalização de toda a comunicação.

- **Versão 3 (mpi_test.c):** - Esta terceira versão também emprega chamadas de comunicação não bloqueantes, `MPI_Isend` e `MPI_Irecv`, para a troca de dados das células de borda. No entanto, adota uma estratégia para tentar sobrepor o cálculo com a comunicação. Após iniciar as operações de envio e recebimento não bloqueantes, o programa prossegue para calcular os valores dos pontos internos do domínio local (aqueles que não são imediatamente adjacentes às células fantasmas), que não dependem dos dados das células fantasmas que estão sendo trocadas naquele momento. Em seguida, utiliza a função `MPI_Test` em laços dedicados para verificar ativamente a conclusão das operações de recebimento (`MPI_Irecv`) das células de borda. Uma vez que o recebimento de dados de uma borda específica é confirmado por `MPI_Test` (indicando que a célula fantasma correspondente foi preenchida), o cálculo para o ponto da malha adjacente a essa borda é realizado. Finalmente, a função `MPI_Wait` é chamada para cada operação de envio (`MPI_Isend`) para garantir que estas também tenham sido concluídas. Esta abordagem visa maximizar a utilização da CPU, permitindo que os cálculos dos pontos internos ocorram enquanto as comunicações das bordas estão em andamento, e processando os pontos de borda assim que seus dados necessários se tornam disponíveis, antes de garantir a finalização dos envios.

4 Resultados

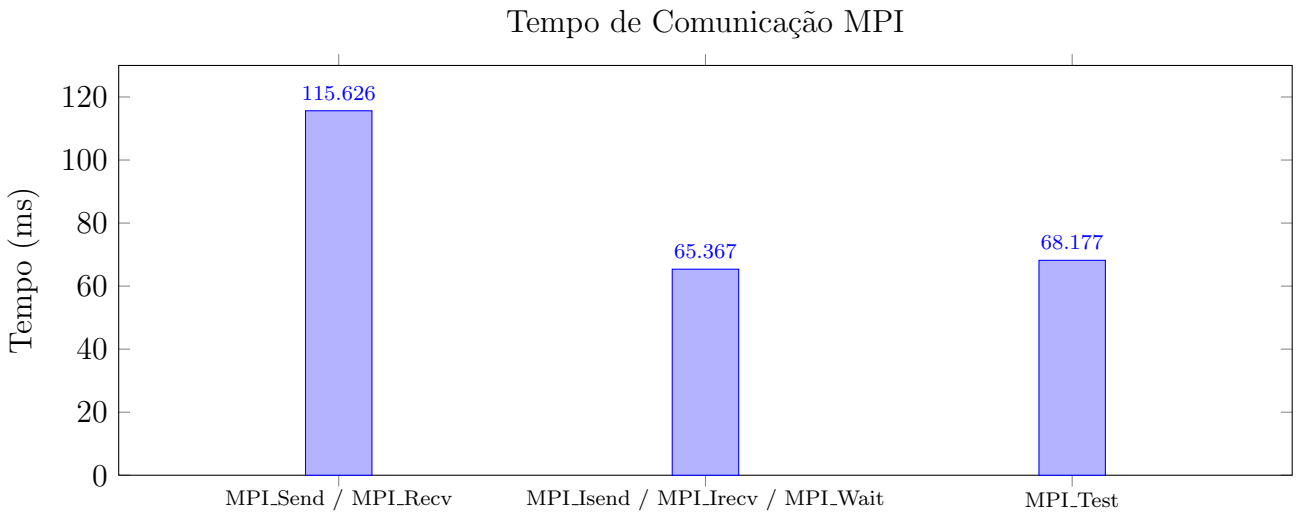

Figure 1: Tempo de execução do programa usando tipos de comunicação diferentes em milissegundos.

5 Análise dos Resultados

A Figura 1 apresenta os tempos de execução medidos para as três versões implementadas da simulação de difusão de calor, utilizando diferentes estratégias de comunicação com MPI. Observa-se que a versão que utiliza `MPI_Send/MPI_Recv`, com comunicação bloqueante, apresentou o maior tempo de execução, aproximadamente **115,626 ms**. Isso ocorre devido ao comportamento bloqueante dessas funções, que força o processo a aguardar a conclusão de cada envio ou recebimento antes de prosseguir, resultando em períodos de inatividade (ociosos) e menor aproveitamento do processador.

Em contrapartida, as versões que utilizam comunicação não bloqueante (`MPI_Isend/MPI_Irecv`) apresentaram desempenhos superiores. A versão intermediária, que utiliza `MPI_Waitall` para aguardar a conclusão de todas as comunicações antes de iniciar o cálculo, apresentou um tempo de execução de aproximadamente **65,367 ms**, representando uma redução de cerca de **43,5%** em relação à versão bloqueante. Isso evidencia que, mesmo sem explorar plenamente a sobreposição entre comunicação

e computação, a simples adoção de operações não bloqueantes reduz significativamente o tempo de execução, ao evitar bloqueios desnecessários durante o envio e recebimento de mensagens.

Por fim, a versão mais otimizada, que utiliza `MPI_Test` para verificar dinamicamente a conclusão das comunicações de borda e iniciar o cálculo dos pontos internos enquanto a comunicação ainda está em andamento, apresentou um tempo de execução de **68,177 ms**. Embora esse valor seja ligeiramente superior ao obtido com `MPI_Waitall`, ainda representa uma redução significativa em relação à versão bloqueante. A pequena diferença entre as versões com e sem sobreposição pode ser atribuída ao tamanho relativamente pequeno da malha processada ou ao custo associado à verificação contínua das operações com `MPI_Test`, o que pode não ter proporcionado um ganho expressivo no cenário considerado.

Além da redução do tempo de execução, foi possível observar o impacto positivo da sobreposição de comunicação e computação no **throughput** do programa, medido em milhões de operações de ponto flutuante por segundo (Mflops/s). A versão bloqueante apresentou um throughput de aproximadamente **43,0 Mflops/s**, enquanto as versões não bloqueantes atingiram **76,1 Mflops/s** (`MPI_Isend`/`MPI_Irecv`/`MPI_Wait`) e **73,0 Mflops/s** (`MPI_Test`), evidenciando um aumento significativo na eficiência computacional quando se reduz o tempo de espera na comunicação.

Em síntese, os resultados demonstram claramente os benefícios da utilização de comunicações não bloqueantes em programas paralelos com MPI, tanto na redução do tempo de execução quanto no aumento do throughput. A versão com sobreposição total de comunicação e computação, embora não tenha superado em desempenho absoluto a versão com `MPI_Waitall`, apresenta vantagens conceituais importantes, especialmente em cenários com maior carga computacional ou comunicações mais custosas, nos quais a sobreposição pode resultar em ganhos ainda mais expressivos.

6 Conclusão

A realização desta atividade permitiu compreender, na prática, como diferentes estratégias de comunicação impactam o desempenho de aplicações paralelas. As versões que utilizam comunicação não bloqueante apresentaram tempos de execução inferiores à versão bloqueante, confirmando a eficiência da sobreposição entre comunicação e computação. Além disso, a versão com `MPI_Test` mostrou-se vantajosa ao permitir o aproveitamento do tempo de espera para realizar cálculos, evidenciando a importância de técnicas que exploram a simultaneidade entre comunicação e processamento em sistemas distribuídos.