TRABALHO PRÁTICO 3
MPI

DATA DE ENTREGA: 25/07/2016.

INSTRUÇÕES:

1. Deve ser elaborado um relatório descrevendo suas implementações, resultados e análises dos exercícios abaixo. O relatório deve ser salvo no formato PDF.
2. O relatório e os códigos-fontes devem ser enviados para o e-mail: camata@nacad.ufrj.br.
3. Serão aceitos trabalhos enviados até às 23:59 do dia da entrega.
4. ATENCÃO: No assunto da mensagem, use a seguinte regra: CAD-2016-TRAB2-NOME.
5. O trabalho poderá ser feito em duplas.

Exercício 1: Crie, compile e execute um programa MPI "Olá mundo" .

- Use o arquivo de inclusão MPI apropriado.
- Identifique a tarefa zero como a tarefa mestre.
- Inicialize o ambiente MPI.
- Obtém o número total de tarefas.
- Obtém a identificação da tarefa (quem é).
- Imprime uma mensagem Olá, que inclui a identificação da tarefa.
- A tarefa mestre deve imprimir o número total de tarefas.
- Finaliza o ambiente MPI.

Exercício 2: Crie, compile e execute um programa de envio/recebimento bloqueante.

Supondo que você foi capaz de criar o programa "Olá mundo" no Exercício 1, copie o arquivo de origem para um novo arquivo renomeando-o como *helloBsend.c*.

Edite o novo arquivo de origem *helloBsend* e modifique-o para fazer o seguinte - depois da tarefa mestre ter impresso o número de tarefas, mas antes do *MPI_Finalize*:

- Faça com que cada tarefa determine sua tarefa parceira para envio. Uma maneira fácil de fazer isso:

```
if (taskid < numtasks/2) then partner = numtasks/2 + taskid
else if (taskid >= numtasks/2) then partner = taskid -
numtasks/2
```

- Cada tarefa envia para seu parceiro de uma mensagem com um inteiro: sua taskid
- Cada tarefa recebe do seu parceiro uma mensagem com um inteiro: taskid do parceiro
- Para confirmação, após o envio/ recebimento, cada tarefa imprime algo como "Tarefa # # tem parceria com # #", onde # # é o taskid da tarefa e de seu parceiro.

Usando o comando mpirun para executar o seu programa. Por exemplo:

```
mpirun  -np 8  helloBsend
```

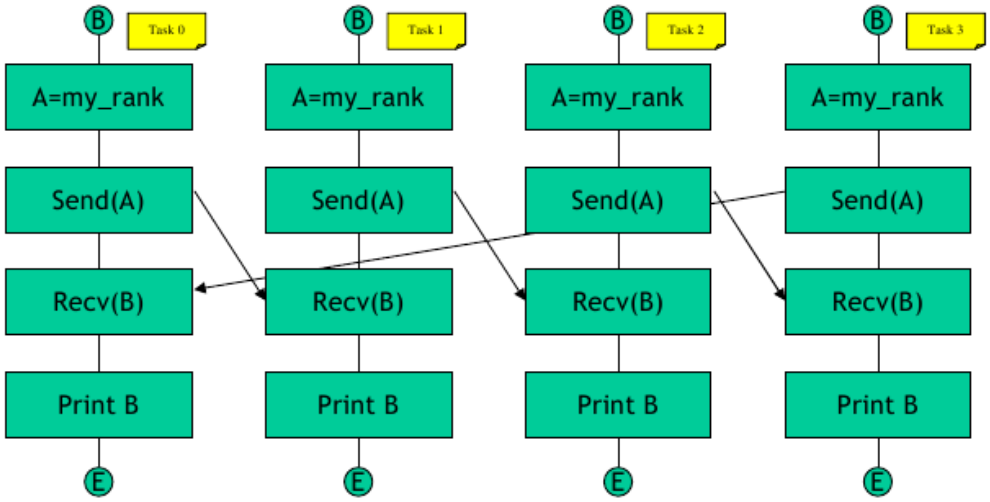
O seu trabalho executou com sucesso? Baseado na saída, ele se comportou conforme o esperado? Se não, tente descobrir os problemas, corrigindo-os antes de prosseguir.

- Copie o arquivo de origem *helloBsend* para um novo arquivo de origem *helloNBsend*. Em seguida, converta as rotinas bloqueante para rotinas não-bloqueantes.

Compile e execute novamente.

Exercício 3: Comunicação em anel.

Escreva um código que use comunicação ponto a ponto para realizar um envio/recebimento circular como representado na figura a seguir



Cada processo irá declarar duas matrizes de pontos flutuantes A e B, de dimensão fixa (10000). Cada elemento de A será inicializado com o rank do processo. Então, A e B serão usados como buffers de envio e recebimento, respectivamente. Cada processo envia apenas para o processo da sua direita e

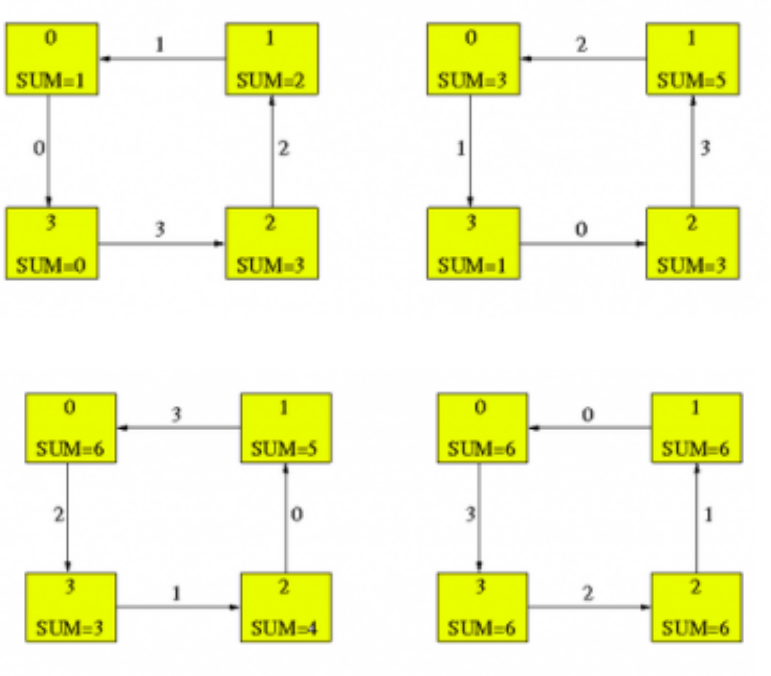
recebe apenas do processo à sua esquerda. Como você pode ver na figura acima, o último processo irá enviar sua matriz A para o primeiro processo. Evite deadlocks e certifique-se de que o código irá funcionar com um número qualquer de processos. O programa termina com cada processo imprimindo um elemento da matriz B.

Exercício 4: Soma Circular.

Escreva um código que executa uma soma circular dos ranks dos processos. Seja A um ponto flutuante inicializado para o rank do processo e B um ponto flutuante usado como um buffer de recebimento. Então, seja SUM uma variável que armazena a soma parcial. O código deve executar uma série de chamadas de envio/recebimento onde, em cada etapa:

a. SUM é atualizado com o que foi recebido em B
b. Ao buffer A é atribuído o valor recebido de B.

A seguinte figure representa o que o código deve fazer:



No final da execução, todos os processos devem armazenar em sua variável SUM, a soma de todos os *ranks*.

Exercício 5: Multiplicação matricial entre processos

Multiplicação de matrizes é um exemplo de um problema que é embaraçosamente paralelo. Isto significa que ele pode ser executado em paralelo sem a necessidade de comunicação entre os processos (exceto para a distribuição das matrizes e recolhimento dos resultados). Considere duas matrizes A e B de ordem N x N. O produto de A e B pode ser dado por

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} \times B_{kj} \quad \text{para todo } 0 \le i, j < N$$

Para facilitar, escolha N que seja múltiplo do número de processadores. Faça:

- O processo mestre deve criar as matrizes A e B.
- O processo mestre deve distribuir (particionar) as matrizes A e B para todos os processos.
- Cada processo deve calcular uma porção da multiplicação matricial AxB
- Recolha as porções dos cálculos para o processo mestre.
- O processo mestre deve imprimir o resultado.
- Informe o tempo decorrido no processo de distribuição das matrizes, do cálculo da multiplicação bem como do recolhimento do produto parcial.

Execute seu programa considerando diferentes tamanhos de matrizes (N = 1.000, 5.000 e 10.000, etc) e calcule o speedup e eficiência de sua implementação.

Exercício 6: Produto interno

Dado dois vetores a e b de tamanho N, o produto interno entre eles é definido como

$$\sum_{i=0}^{N-1} a_i \times b_i \ .$$

Faça:

- Crie os vetores a e b de tamanho N.
- Distribua os vetores entre os processos.
- Cada processo deve calcular a produto interno parcial para um bloco de valores dos dois vetores
- O resultado final dever ser a combinação desses resultados parciais.

Compare os resultados da versão sequencial com a paralela.

Exercício 7: Jogo da Vida

O "Jogo da Vida" é uma simples simulação desenvolvida por John Conway. Nesse jogo, o domínio é uma matriz bidimensional de células, onde cada célula pode ter um de dois estados possíveis, geralmente referidos como "viva" ou "morta". A matriz é geralmente inicializada usando números aleatórios e, em seguida, o sistema evolui no tempo. Em cada intervalo de tempo, cada célula pode ou não alterar o seu estado, com base no número de células vivas, incluindo as diagonais adjacentes. Existem três regras:

1. Se uma célula tem três vizinhos que estão vivos, a célula ficará viva. Se ela está viva, continuará assim, e se ela estiver morta, ela se tornará viva.
2. Se uma célula tem dois vizinhos que estão vivos, não há nenhuma mudança para a célula. Se ela está morta, permanecerá morta, e se ela estiver viva, permanecerá viva.
3. Em todos os outros casos, a célula vai estar morta. Se estivesse viva, morreria e se estando morta permanece morta.

O código serial do Jogo da Vida encontra-se anexo a este documento.

Para este exercício, adicione as rotinas de inicialização e finalização do MPI no código serial "Jogo da Vida". Isso duplicará o mesmo cálculo em cada processador. A fim de mostrar que o código está funcionando como esperado, adicione instruções para imprimir o numero total de processos e o identificador do processo local. Não se esqueça de adicionar o arquivo de cabeçalho *mpi.h*.

> *Dica:* Uma questão que vem à tona quando paralelizamos um código é como ligado com I/O. Como você pode imaginar, ter vários processos de escrita para o mesmo arquivo ao mesmo tempo pode produzir resultados inúteis. Uma solução simples é fazer com que cada processo escreva um arquivo de saída chamado com o seu rank. A saída para esses arquivos separados elimina o problema. Aqui está como fazer isso em C:
>
> ```
> sprintf(outfilename,"found.data_%d",myrank);
> outfile = fopen(outfilename,"w") ;
> ```

A fim de executar realmente o programa "Jogo da Vida", em paralelo, devemos realizar a decomposição de domínio, ou seja, dividir o domínio em pedaços e enviar um pedaço para cada processador. Uma questão que você precisa considerar é a fronteira interna do domínio. A Figura 1 mostra a decomposição do domínio "esquerda-direita". Cada célula necessita de informações de todas as células adjacentes para determinar o seu novo estado. Com o domínio decomposto, algumas células necessárias já não estão disponíveis no processo local. Uma maneira comum de

resolver este problema é através do uso de células fantasmas. Nesse exemplo, uma coluna de células fantasmas é adicionada para o lado direito do domínio esquerdo, e uma coluna também é adicionada ao lado esquerdo do domínio direito (como mostrado na Figura 2). Após cada intervalo de tempo, as células fantasmas são preenchidas fazendo passar os dados apropriados do outro processador.

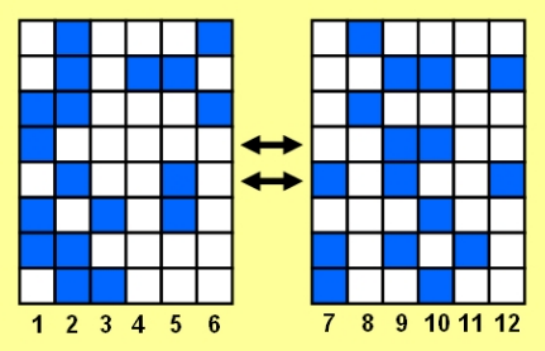


Figura 1: Decomposição Domínio

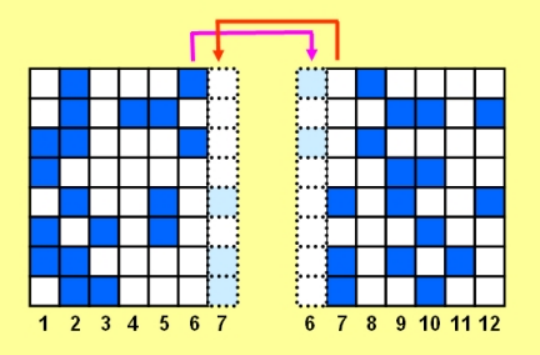


Figura 2: Células Fantasmas

Implemente a decomposição de domínio descrito acima e adicionar as rotinas de trocas de mensagem para obter os dados nas células fantasmas. Para quem programa em C, não se esqueça de dividir o domínio usando uma linha e não em coluna (como mostrado nas Figuras 1 e 2). Seu programa deve funcionar para qualquer número de processos MPI.