Processamento Paralelo
Aula 2 – Comunicação Ponto a Ponto

Adriano Wagner

18/01/2011

MPI - Comunicação

- Troca de mensagens entre participantes.
- Mensagem pode ser uma variável de qualquer tipo e dimensão
  - Quem envia
  - Quem recebe
  - Origem do dado
  - Destino do dado
  - Tamanho do dado
  - Tipo do dado
- Tipos de comunicação
  - Ponto a ponto
  - Coletiva

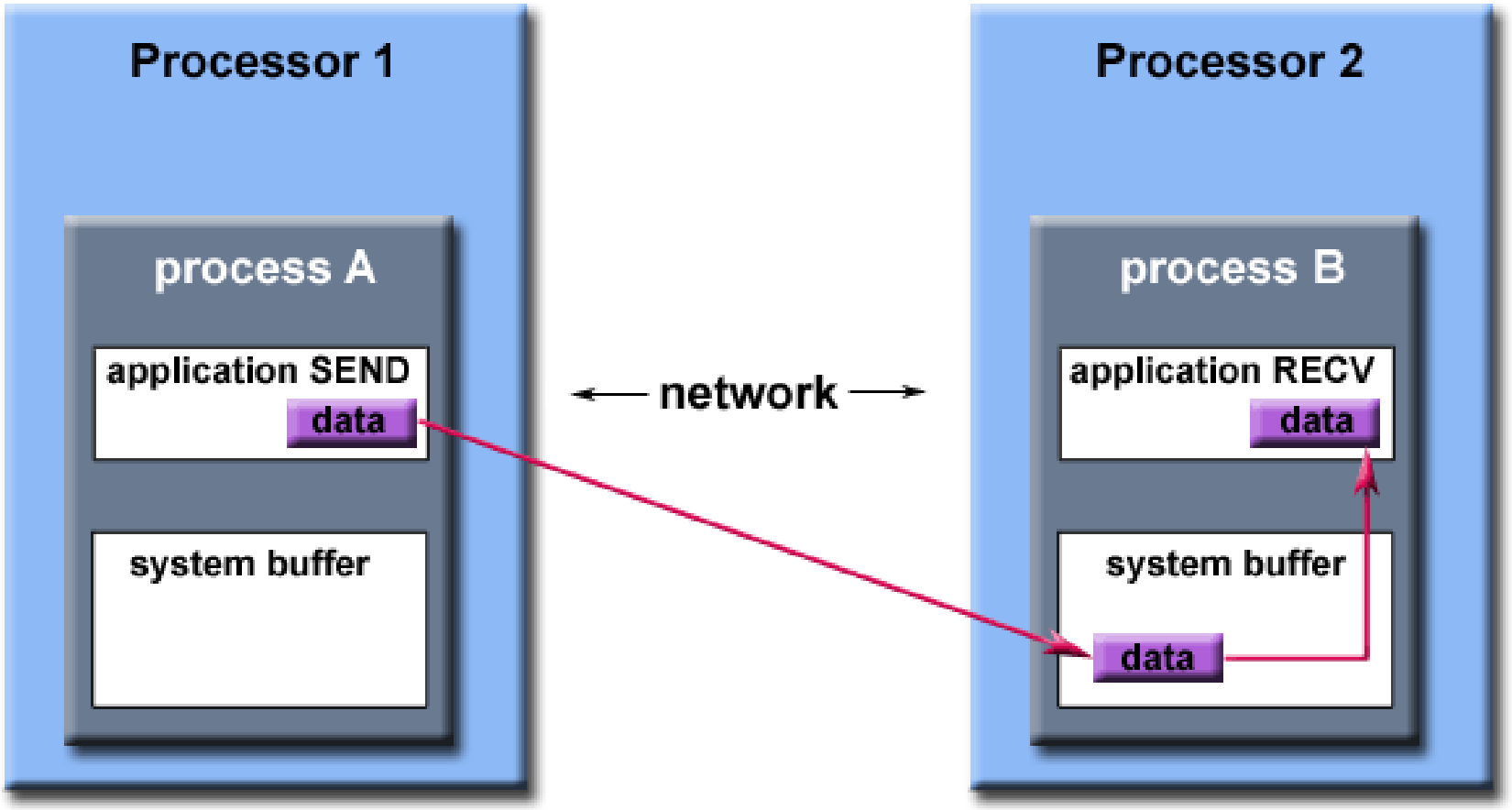
MPI – Comunicação Ponto a Ponto

- Mensagem passada de um remetente para um receptor.
- Apenas o remetente e o receptor precisam saber da mensagem.
- Ex: Carta, fax...
- Tipos de comunicação.
  - Síncrono x Assíncrono
  - *Blocking* x *Non-blocking*

MPI – Comunicação Coletiva

- Envolvem mais de dois participantes.
- Pode ter muitos receptores e/ou muitos remetentes.

MPI - Comunicação



Path of a message buffered at the receiving process

MPI - Mensagem

- Quem envia / Quem recebe
  - rank
- Origem do dado / Destino do dado
  - variável
- Tamanho do dado
- Tipo do dado

MPI - Tipos

- Equivalentes aos tipos do C/Fortran
- MPI_<TIPO>
- Ex:
  - MPI_INT – int
  - MPI_FLOAT – float
  - MPI_INTEGER – integer
  - MPI_REAL – real
  - MPI_COMPLEX – complex
- http://www.mcs.anl.gov/research/projects/mpi/www/www3/Constants.html

MPI - Exemplo

- exemplo1
- Envio de uma mensagem ponto a ponto
- MPI_Send / MPI_Recv

MPI – Elementos do envio

- <function>(buf, count, datatype, dest, tag, comm, [request])
- buf – Endereço da variável que será enviada.
- count – Número de elementos enviados.
- datatype – Tipo do MPI, equivalente a variável enviada.
- dest – Rank do destino da mensagem.
- tag – Número inteiro que identificará a mensagem.
- comm – Grupo de comunicação (MPI_COMM_WORLD).
- request – Possibilita que o programa teste no futuro se um envio foi concluído com sucesso.

MPI – Elementos do recebimento

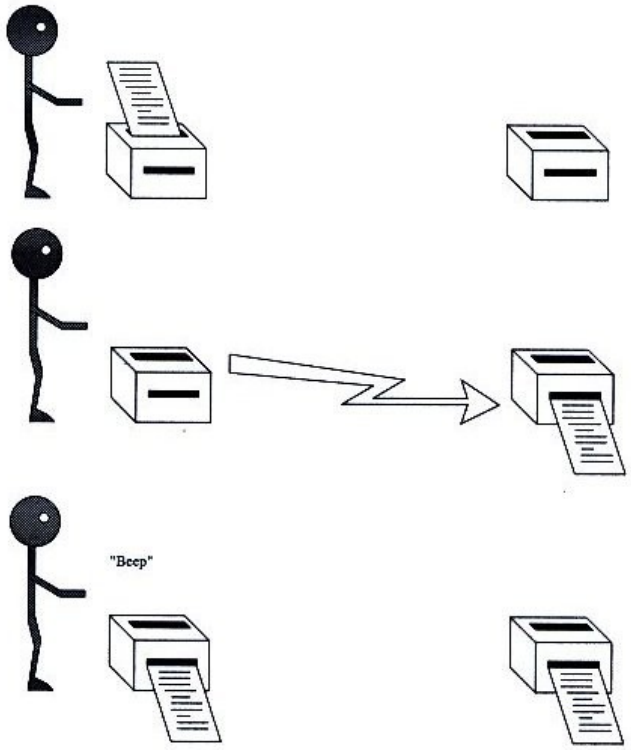
- <function>(buf, count, datatype, source, tag, comm, [request ou status])
- buf – Endereço da variável que receberá o conteúdo.
- count – Número de elementos esperados.
- datatype – Tipo do MPI, equivalente a variável recebida.
- source – Rank da origem da mensagem.
- tag – Identificador usado na mensagem de envio.
- comm – Grupo de comunicação (MPI_COMM_WORLD).
- request – Possibilita que o programa teste no futuro se um recebimento foi concluído com sucesso.
- status – Contém algumas informações sobre a mensagem, como a tag e o destino.

Exercício

- Alterar exemplo visto
- Mudar tipo da variável para real/float
- Após o recebimento do valor, multiplicar por 2.5 e enviar de volta
- Imprimir resultado

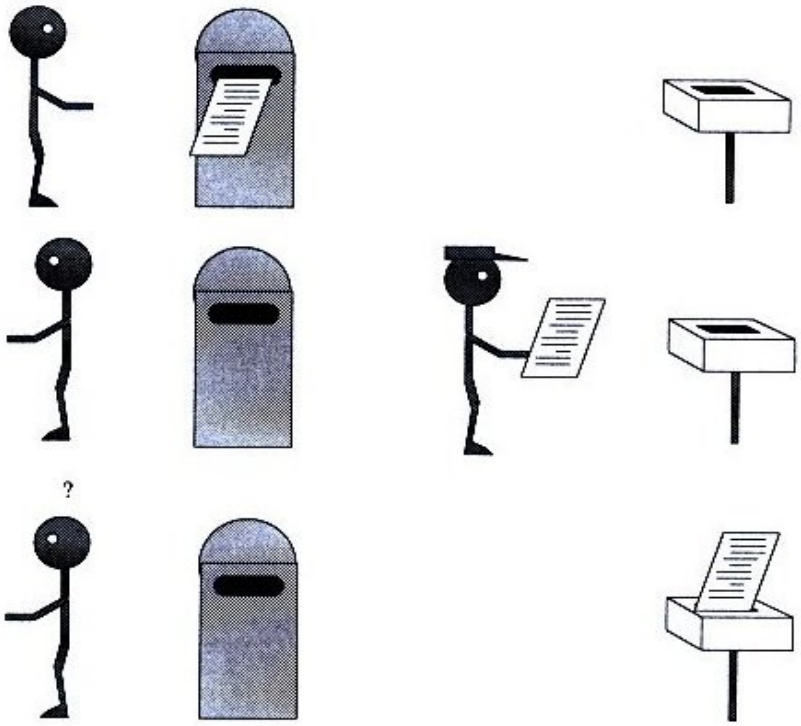
Comunicação Síncrona

- Remetente tem conhecimento da entrega da mensagem.
- Ex: Mensagem por fax e carta registrada.



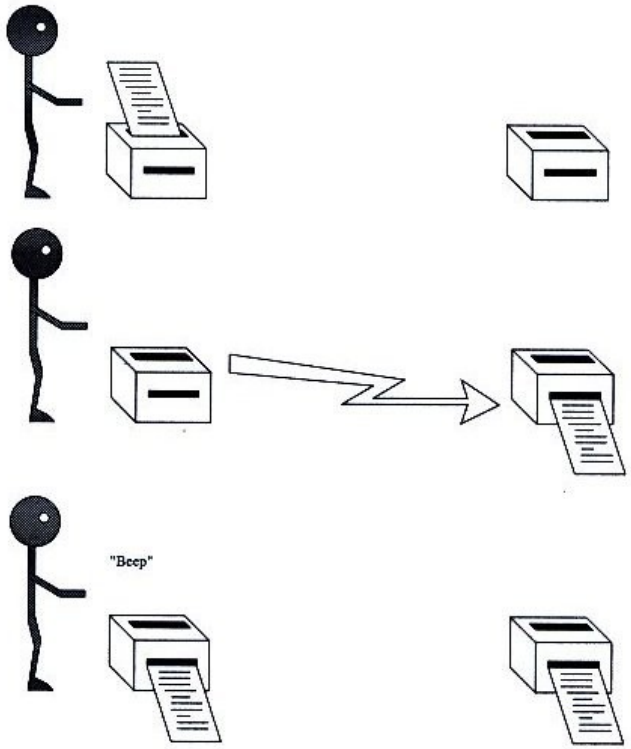
Comunicação Assíncrona

- Remetente não tem conhecimento da entrega da mensagem.
- Ex: E-mail e cartão-postal.



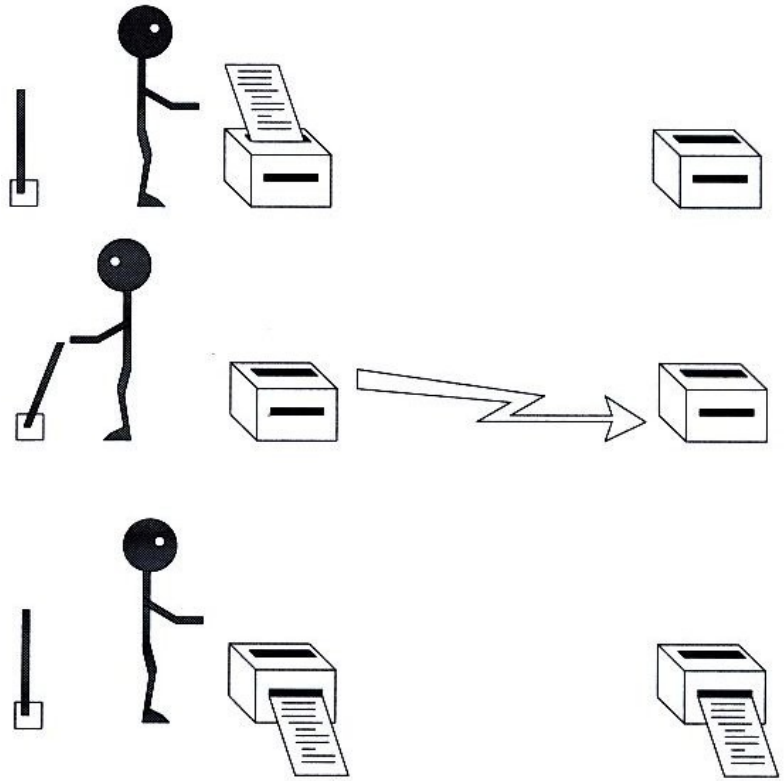
Blocking Communication

- Remetente deve esperar o término do envio da mensagem.
- Ex: Aparelho de fax sem memória.



Non-blocking Communication

- Remetente não precisa esperar o término do envio da mensagem.
- Ex: Aparelho de fax com memória.



Comunicação ponto a ponto

	Síncrono	**Assíncrono**
Blocking	• Bloqueia o processo que enviou a mensagem, liberando-o apenas quando o destinatário receber a mensagem.	• Bloqueia o processo até que o envio seja concluído, ou seja, o processo possa utilizar o buffer de envio de forma segura. • Não há conhecimento sobre o recebimento.
Non-Blocking	• Não bloqueia o processo. • Envio só tem sucesso quando o destinatário receber a mensagem.	• Não bloqueia o processo. • Envio terá sucesso assim que o buffer de envio tenha sido copiado.

Comunicação ponto a ponto - Funções

- Envio

	Síncrono	**Assíncrono**
Blocking	• MPI_Ssend(buf, count, datatype, dest, tag, comm)	• MPI_Send(buf, count, datatype, dest, tag, comm)
Non-Blocking	• MPI_Issend(buf, count, datatype, dest, tag, comm, request)	• MPI_Isend(buf, count, datatype, dest, tag, comm, request)

- Recebimento

Blocking	• MPI_Recv(buf, count, datatype, source, tag, comm, status)
Non-Blocking	• MPI_Irecv(buf, count, datatype, source, tag, comm, request)

Exemplo

- exemplo2
- Envio síncrono e blocking de uma mensagem.
- Origem desconhecida.
- MPI_Status contém informações sobre a origem.

Exemplo

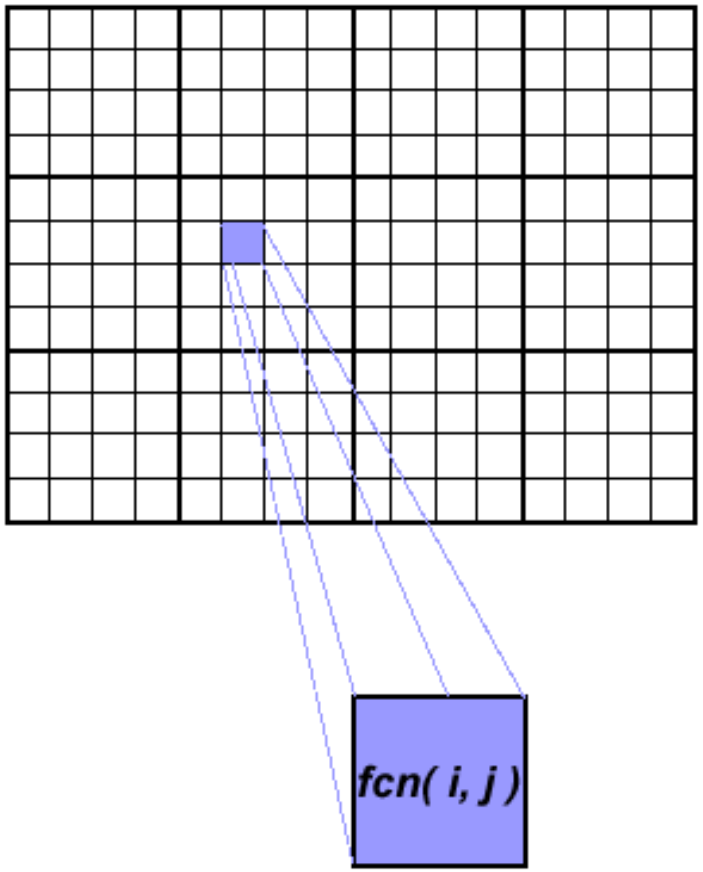
- exemplo3
- Envio assíncrono e non-blocking de uma mensagem
- MPI_Wait – Espera até que um request seja completado
- MPI_Test – Testa se um request foi completado

Processamento Paralelo - Estratégias

- Melhor forma de dividir o problema
  - Custo adicional
    - Troca de mensagens
    - Sincronização dos processos
  - Ganho obtido
    - Redução do tempo de execução
    - Redução do conjunto de dados
- Escalabilidade – manutenção do desempenho com o aumento de nós
- Depende do hardware disponível

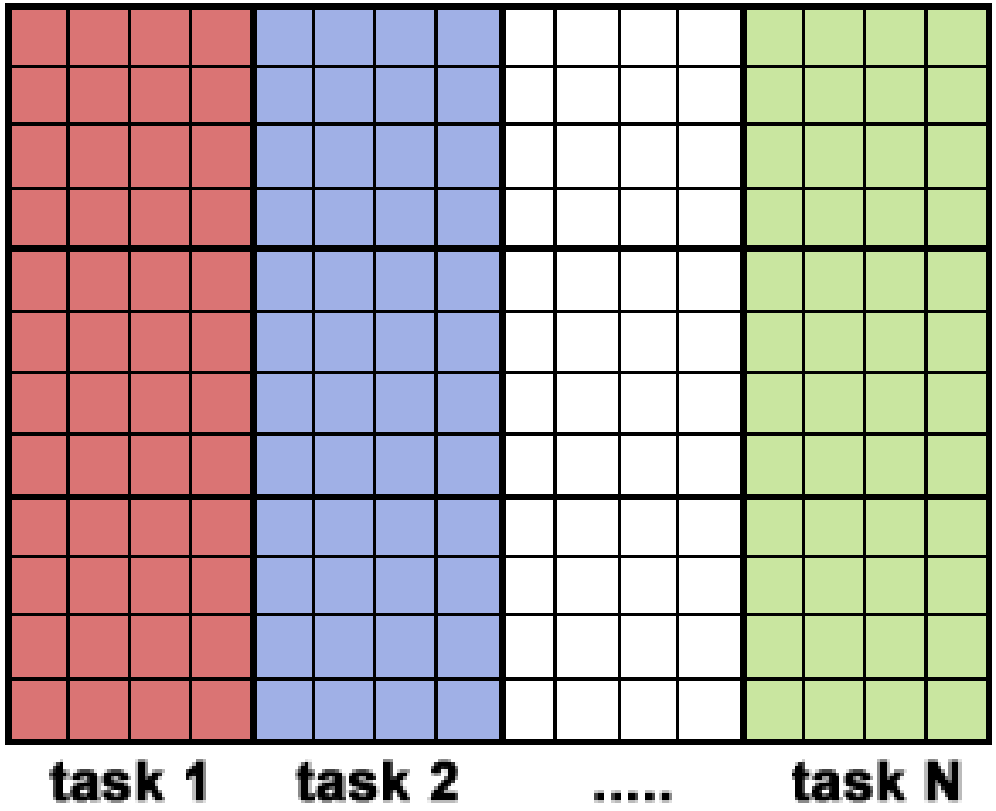
Processamento Paralelo - Exemplo

- Problema : Processamento de uma matriz de valores
- Cálculo de um ponto não depende dos demais



Solução 1

- Dividir a matriz e delegar uma parte a cada nó



Solução 1

```
find out if I am MASTER or WORKER

if I am MASTER

  initialize the array
  send each WORKER info on part of array it owns
  send each WORKER its portion of initial array

  receive from each WORKER results

else if I am WORKER
  receive from MASTER info on part of array I own
  receive from MASTER my portion of initial array

  # calculate my portion of array
  do j = my first column,my last column
  do i = 1,n
    a(i,j) = fcn(i,j)
  end do
  end do

  send MASTER results

endif
```

Solução 1

- Comunicação é feita apenas no início e no fim do processo
- Pode causar desbalanceamento

Solução 2

- Delegar o cálculo de uma posição da matriz assim que o nó estiver disponível.

Solução 2

```
find out if I am MASTER or WORKER

if I am MASTER

  do until no more jobs
    send to WORKER next job
    receive results from WORKER
  end do

  tell WORKER no more jobs

else if I am WORKER

  do until no more jobs
    receive from MASTER next job

    calculate array element: a(i,j) = fcn(i,j)

    send results to MASTER
  end do

endif
```

Solução 2

- Mantém o processo balanceado, reduzindo o tempo ocioso
- Aumento no número de comunicações

Exercíco

- Criar um programa com um vetor de tamanho 7
- Preencher
- Mestre (rank == 0)
  - Enviar cada valor para um nó
  - Receber resultados
  - Imprimir valores finais
- Escravo
  - Receber um valor
  - Multiplicar pelo rank
  - Enviar de volta
- Rodar com 8 processadores

Links úteis

- http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortran.html
- http://www-teaching.physics.ox.ac.uk/Unix+Prog/hargrove/tutorial_77/
- http://www.ead.cpdee.ufmg.br/cursos/C/
- https://computing.llnl.gov/tutorials/parallel_comp/
- https://computing.llnl.gov/tutorials/mpi/