Comunicação Coletiva em MPI

Tópicos:

- Operação de *Broadcast*
- Operações de Redução
- Exemplo: Produto Escalar
- Operações de Redução Globais

Referência: Pacheco,P.S. *Parallel Programming with MPI* Morgan Kaufmann, San Francisco, 1997.

Operação de *Broadcast*

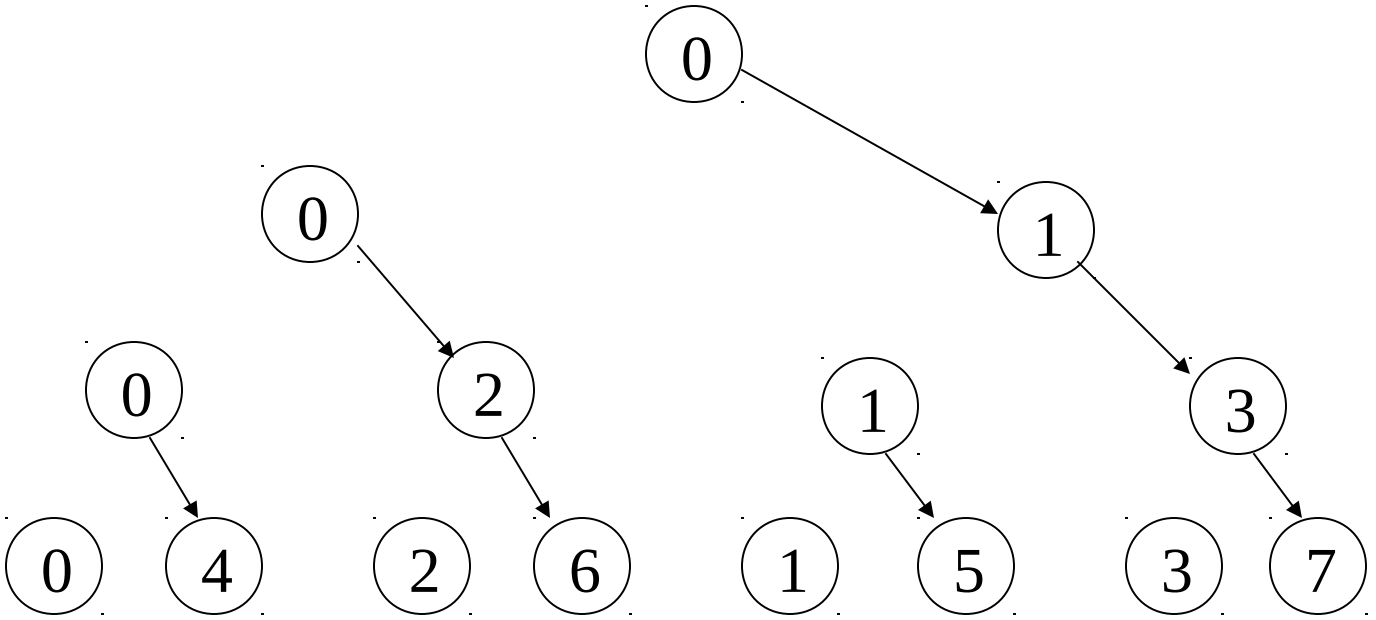
- **Objetivo**: Enviar mensagens com o mesmo conteúdo, de uma certa CPU para todas as demais
- Forma Trivial:

```
if (my_rank == 0) {
   for (dest=1; dest<P ; dest++)
     MPI_Send(&dado,1,MPI_INT,dest,tag,commun);
}
else {
   MPI_RECV(&dado,1,MPI_INT,0,tag,commun,stat);
} ;
```

- Problema: Execução serializada (pouco eficiente quando há muitos processadores)

Operação de *Broadcast* (cont.)

- **Otimização:** Criar uma árvore de transmissão



Operação de *Broadcast* (cont.)

- **Transmissão convencional:** P-1 passos
- **Transmissão otimizada**: $\log_2 P$ passos
- **Obs**: Número total de mensagens é o mesmo (P-1)
- Ganho de tempo só ocorre se mensagens puderem ser transmitidas simultaneamente
- Em MPI: `MPI_Bcast(...)`
  - Implementação interna dependente de cada sistema
  - Facilita a codificação para o programador

Operação de *Broadcast* (cont.)

- Tempos de broadcast (em ms, usando MPICH)

	nCUBE2		Paragon		SP2	
# CPU's	conv.	otim.	conv.	otim.	conv.	otim.
2	0.59	0.69	0.21	0.43	0.15	0.16
8	4.7	1.9	0.84	0.93	0.55	0.35
32	19.0	3.0	3.2	1.3	2.0	0.57

Operação de *Broadcast* (cont.)

- Função de *Broadcast* em MPI:

```
int MPI_Bcast( void*           buffer,
               int             count,
               MPI_Datatype    datatype,
               int             root,
               MPI_Comm        communicator )
```

- Observações:
  - CPU `root` envia dados; as demais recebem dados
  - Todas as CPU's devem participar da operação
  - NÃO há sincronização de tempo entre as CPU's
  - É importante observar a ordem em sucessivas chamadas a `MPI_Bcast( )`

Operação de *Broadcast* (cont.)

- Exemplo de utilização (supondo `root`=A) :

Tempo	CPU A	CPU B	CPU C
1	MPI_Bcast(x)	proc.local	proc.local
2	MPI_Bcast(y)	proc.local	proc.local
3	proc.local	MPI_Bcast(y')	MPI_Bcast(x")
4	proc.local	MPI_Bcast(x')	MPI_Bcast(y")

- CPU A envia dados (x e y)
- CPU C recebe dados na ordem certa (x"←x, y"←y)
- CPU B recebe dados invertidos (y'←x, x'←y)

Operações de Redução

- **Conceito**: Operação que combina diversos valores, produzindo um único resultado
- **Exemplos**:
  - Soma total de N números: $S = k_1 + k_2 + k_3 + ... + k_N$
  - Máximo de N números: $M = \max\{k_1 , k_2 , k_3 , ... , k_N\}$
- Requisito: Operação deve ter propriedade associativa
  - Exemplos: $+$, $\times$, max , min , $\wedge$, $\vee$, $\oplus$
  - Contra-Exemplos: $-$, $\div$

Operações de Redução (cont.)

- Redução em MPI: `MPI_Reduce( )`

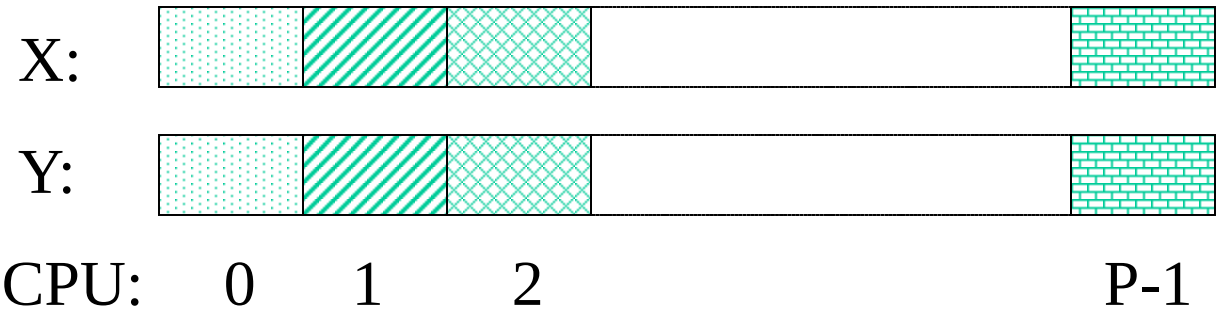
```
int MPI_Reduce( void*           operand,
                void*           result,
                int             count,
                MPI_Datatype    datatype,
                MPI_Op          operator,
                int             root,
                MPI_Comm        communicator )
```

- Todas as CPU's entram com seus valores de `operand`
- Apenas a CPU `root` sai com o valor de `result`
- Exemplos de `operator`: `MPI_SUM`, `MPI_MAX`, ...

Exemplo: Produto Escalar

$\mathbf{X} \cdot \mathbf{Y} = x_0 y_0 + x_1 y_1 + x_2 y_2 + \ldots + x_{N-1} y_{N-1}$

- Supondo P CPU's, com N divisível por P:
  - CPU 0 armazena $x_0, x_1, \ldots, x_{N/P-1}, y_0, y_1, \ldots, y_{N/P-1}$
  - CPU 1 armazena $x_{N/P}, x_{N/P+1}, .., x_{2N/P-1}, y_{N/P}, y_{N/P+1}, .., y_{2N/P-1}$
  - . . .



Exemplo: Produto Escalar (cont.)

- Algoritmo:
  - Realizar produto escalar interno em cada CPU
  - Combinar (somar) resultados entre as várias CPU's

```
float Parallel_dot(
          float  local_x[]  /* in */,
          float  local_y[]  /* in */,
          int  n_bar        /* in */) {
    float  local_dot;
    float  dot = 0.0;
    float  Serial_dot(float x[], float y[], int m);
    local_dot = Serial_dot(local_x, local_y, n_bar);
    MPI_Reduce(&local_dot, &dot, 1, MPI_FLOAT,
        MPI_SUM, 0, MPI_COMM_WORLD);
    return dot;
} /* Parallel_dot */
```

Operações de Redução Globais

- **Objetivo**: Produzir o resultado da redução em todas as CPU's → `MPI_Allreduce( )`

```
int MPI_Allreduce( void*          operand,
                   void*          result,
                   int            count,
                   MPI_Datatype   datatype,
                   MPI_Op         operator,
                   MPI_Comm       communicator )
```

- Todas as CPU's entram com seus valores de `operand`
- Todas as CPU's saem com o mesmo valor de `result`
- Possível implementação interna: $\log_2 P$ passos