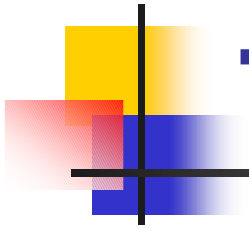
Algoritmos Paralelos usando CGM/MPI

Edson Norberto Cáceres e Siang Wun Song
DCT/UFMS e DCC/IME/USP
Aula 05

Algoritmos Paralelos BSP/CGM

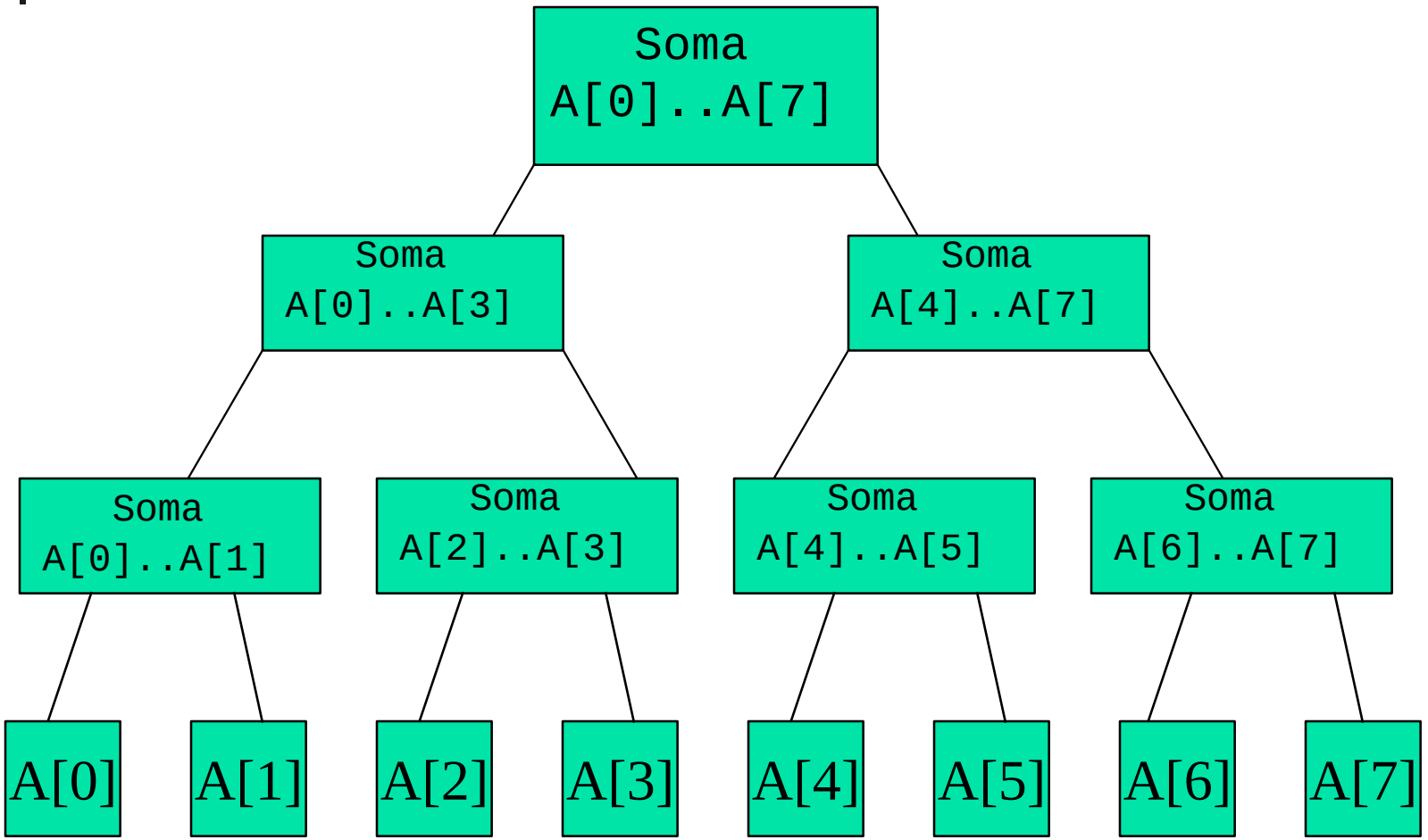
- **Objetivos**
  - **Descrever algumas das principais técnicas para o desenvolvimento de algoritmos paralelos.**
  - **Descrever uma algoritmo paralelo para Ordenação**
  - **Analisar a eficiência dos algoritmos apresentados.**



Técnicas Básicas

- Soma.
- Área sob uma Curva - Trapézios
- Soma de prefixos.
- Ordenação – Bucket Sort.

Algoritmo da Soma - PRAM



Soma no Modelo PRAM

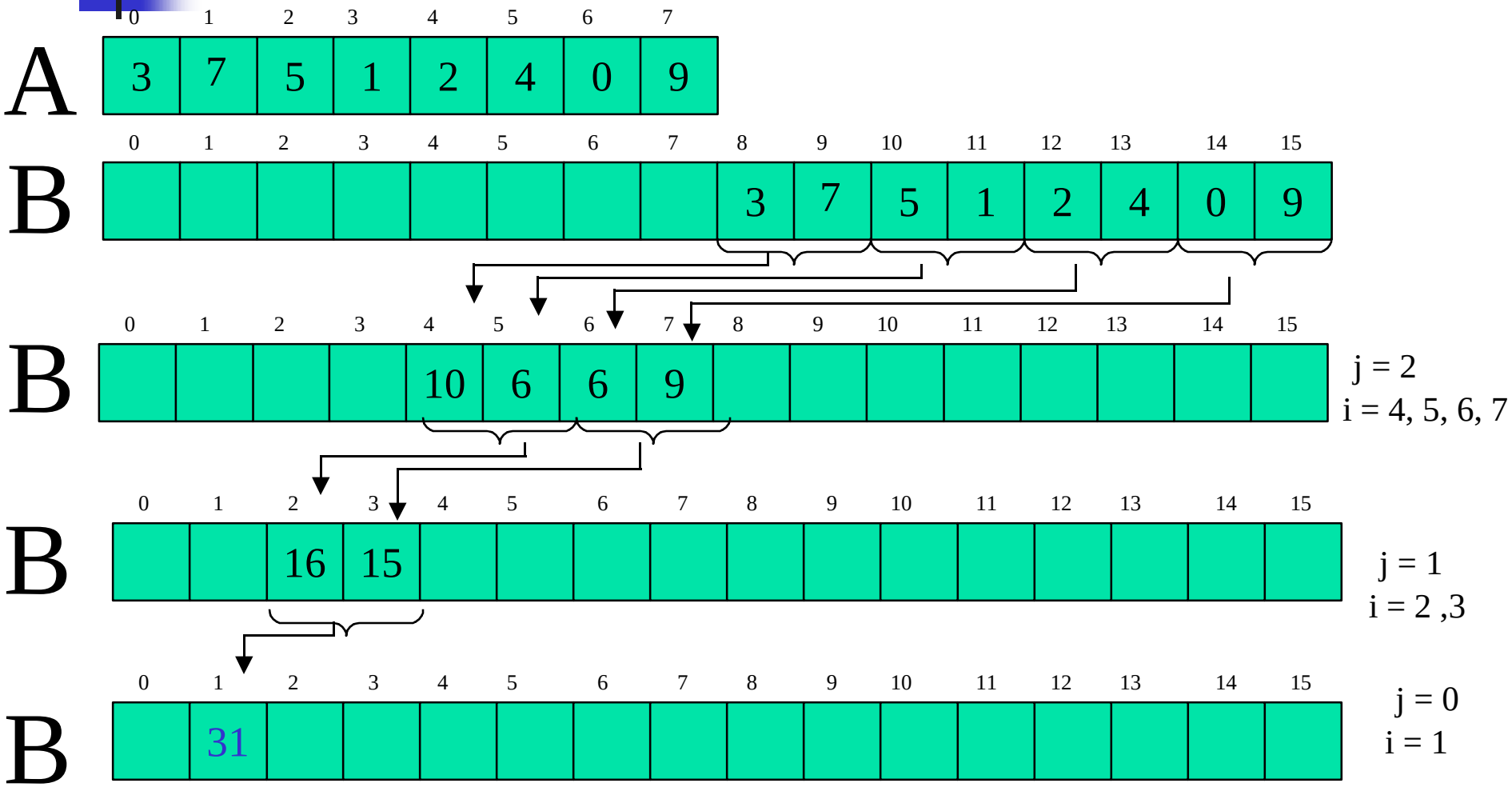
```
global read(A(i), a)
global write(a, B(i))
para h = 1 a log n faça
    se (i <= n/2^h) então
        início
            global read(B(2i-1). X)
            global read(B(2i), y)
            z := x + y
            global write(z, B(i))
        fim
    se i = 1 então global write(z, S)
```

Algoritmo da Soma - PRAM

Algoritmo da Soma de um Vetor

```
/* Passo 1: vetor A é copiado para 2ª metade de B */
PARA 0 <= i <= n-1 FAÇA EM PARALELO
        B[n+i] := A[i];
/* Passo 2: loop sequencial, para cada nível da árvore */
PARA j = log2n - 1 ATÉ 0 FAÇA
    /*loop paralelo alocando um processador para cada
    subproblema deste nível*/
        PARA 2^j <= i <= 2^(j+1) -1 FAÇA EM PARALELO
                B[i] := SOMA(B[2i], B[2i+1]);
```

Algoritmo da Soma - PRAM



Algoritmo da Soma - PRAM

- Submodelo: EREW
- Tempo: O ($\log_2 n$)
- Processadores – O(n)
- Custo (Tempo x Processadores): O (n $\log_2 n$)

Algoritmo da Soma - CGM

- **Entrada:** n elementos v(1),...,v(n)
- **Saída:** $S = v(1)+...+v(n)$
- p processadores
- Cada processador p_i recebe n/p elementos, efetua a soma S_i e envia o resultado para p_1
- p_1 efetua a soma de $S=S_1+...+S_p$
- **Detalhes**

Soma – CGM

entrada: número do processador i; o número p de processadores; $B = A((i-1)r+1:ir)$, $r = n/p$.

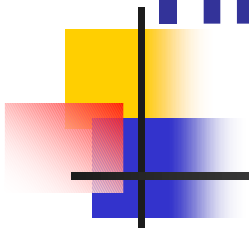
saída: P_i calcula $z = B(1)+...+B(n)$ e envia o resultado para P_1. P_1 calcula $S = z_1+...+z_p$.

1) z := B[1] + ... + B[r];
2) se i = 1 então S := z;
   caso contrário send(z,P_1);
3) se i = 1 então
   para i = 2 até p faça
   receive(s[i],P_i);
4) S := S + s[2] + ... S[p];

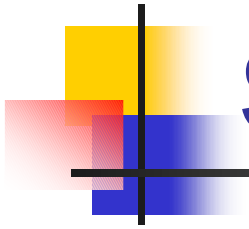
Soma – CGM – Complexidade

- **Passo 1: cada P_i efetua r operações.**
- **Passo 2: P_1 efetua uma operação e os demais P_i's enviam uma MSG.**
- **Passo 3: P_1 recebe p-1 mensagens.**
- **Passo 4: P_1 efetua p-1 operações.**
- **Uma rodada de comunicação.**
- **$O(n/p)$**

Implementação - SPMD - MPI



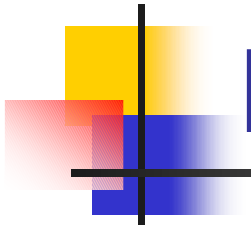
- **Troca de Mensagens**
  - **SPMD**
    - **Processo 0 faz a distribuição dos dados.**
    - **Processo 0 faz a soma.**
- **MPI – Message Passing Interface**
- **Beowulf**



Soma

```
#include<mpi.h>

// Passo 1. Inicializar
// Passo 2. Envie os dados as tarefas (0)
// Passo 3. Receba uma MSG da tarefa 0 (!=0)
// Passo 4. Calcule a Soma
// Passo 5. Receba as Somas Parciais (0)
// Passo 6. Imprima o valor da Soma (0)
// Passo 7. Finalize o MPI
```



Passo 1

- **Inicializar o MPI**

  MPI_Init(int *argc, char ***argv);
- **Verificar quantas tarefas estão sendo executadas**

  MPI_Comm_size(MPI_Comm comm, int *size);
- **Identificar a tarefa**

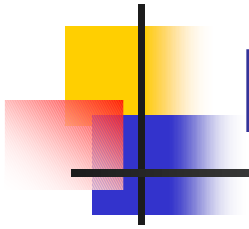
  MPI_Comm_rank(MPI_Comm comm, int *rank);

```
// Passo 1. Inicialização
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
tam = (int) TAMMAX/size; // tamanho subvetor
```

Passo 2 – Enviando dados

- Dividir a entrada em *p* sub-vetores
- Enviar os dados para as tarefas !=0: MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);

```
// Passo 2. Envie os dados as tarefas
if (rank == 0) {
  for (i = 1; i < size; i++) {
     for (j = 0; j < tam; j++)
      SubVetor[j]=VetorDados[tam*i + j];}
      MPI_Send(SubVetor, tam, MPI_INT, i, MSGTAG,
                    MPI_COMM_WORLD);
   } // fim for
   for (j = 0; j < tam; j++) //  Vetor Tarefa 0
       SubVetor[j]=VetorDados[j];
```



Enviando dados no MPI

- Não é necessário inicializar o buffer de envio.
- Não é necessário empacotar os dados.

```
MPI_Send(SubVetor, tam, MPI_INT, i,
         MSGTAG, MPI_COMM_WORLD);
```

Passo 3 – Recebendo dados

- **Receber dados:** MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);

```
else {
// Passo 3. Recebe uma MSG da tarefa 0
    MPI_Recv(SubVetor, tam, MPI_INT, 0,
        MSGTAG, MPI_COMM_WORLD, &status);
} // fim if
```

- Não é necessário desempacotar os dados.

Passo 4 – Efetuando a Soma

```
// Passo 4. Calcule a Soma
for (i = 0; i < tam; i++)
    SomaP = SomaP + SubVetor[i];
```

Passo 5 -Enviando/Recebendo

2) se i = 1 então
   para i = 2 até p faça
   receive(z,P_i);
   S := S + z;
3) se i = 1 então S := z;
   caso contrário send(z,P_1);

- Enviando/Recebendo Dados: MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm);

```
// Passo 5. Receba as Somas Parciais
MPI_Reduce(&SomaP, &Soma, 1, MPI_INT,
                    MPI_SUM,  0, MPI_COMM_WORLD);
```

Passo 6 e 7 - Finalizando

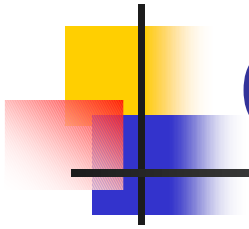
```
// Passo 6. Imprima o valor da Soma
   (Processador 0)
if (rank == 0)
     printf("Soma: %d\n",Soma);
// Passo 7. Finalize o MPI
  MPI_Finalize();
  return 0;
```

MPI - Inicialização

- Criar um arquivo lamhosts

```
# uma LAM com 4 nós
node01
node02
node03
node04
```

- Inicializar a LAM
  - $ lamboot –v lamhosts
  - LAM X.X.X – University of Notre Dame
  - ...
  - $



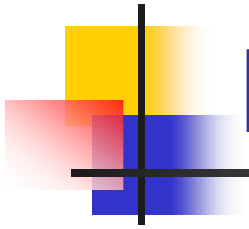
Compilando com MPI

$ mpicc –o somampi soma.c <enter>

Executando com MPI

- Para compilar um programa a LAM não necessita estar em execução.
- Para executar um programa que utilize o mpi, todos os hosts que forem ser utilizados devem estar no arquivo lamhosts.

$ mpirun –np N somampi <enter>

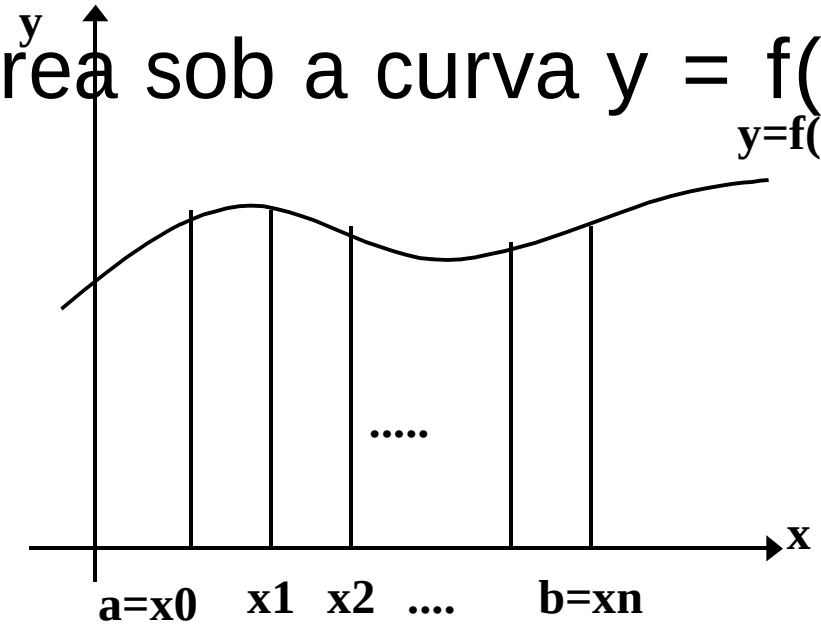


Finalizando o MPI - LAM

- $ lamclean –v <enter>
- $ lamhalt –v <enter>

Computação Paralela

- Área sob a curva y = f(x)



Área sob uma Curva

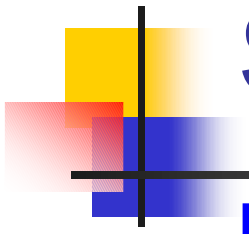
$$\int_a^b f(x)dx$$

$$\frac{1}{2}h[f(x_{i-1}) + f(x_i)]$$

$$[f(x_0)/2 + f(x_n\}/2 + f(x_1) + \cdots + f(x_n)]h$$

- Processador p_i calcula um conjunto de $f(x_i)$
- Algum processador calcula o Resultado Final

Soma de Prefixos



Dado um vetor *A* de *n* elementos *A*[0],...,*A*[*n*-1],

a computação de prefixos calcula os valores:

***A*[0]**

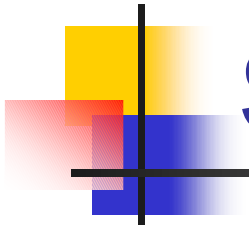
***A*[0] *op* *A*[1]**

***A*[0] *op* *A*[1] op *A*[2]**

...

***A*[0] *op* ... *op* *A*[n-1]**

onde *op* é uma operação binária associativa.



Soma de Prefixos

- **Exemplo: op é a adição.**
- **Vetor de Entrada:**
  - **[3 7 5 1 2 4 0 9]**
- **Vetor Resultante:**
  - **[3 10 15 16 18 22 22 31]**

Algoritmo Seqüencial

```
SOMA_PREFIXO(In, Out)
/* Passo 1: Out[0] recebe a soma do primeiro prefixo de In. Neste caso é
   o próprio In[0]. */
soma := In[0]
Out[0] := soma
/* Passo 2: Calcule os demais prefixos */
PARA i = 1 ATÉ i = n-1 FAÇA
    /* Passo 2.1. A soma do i-ésimo prefixo é a i-ésima posição do vetor
    de entrada mais soma. soma contém a soma do (i-ésimo)-1 prefixo. */
    Out[i] := In[i] + soma
    soma := Out[i]
```

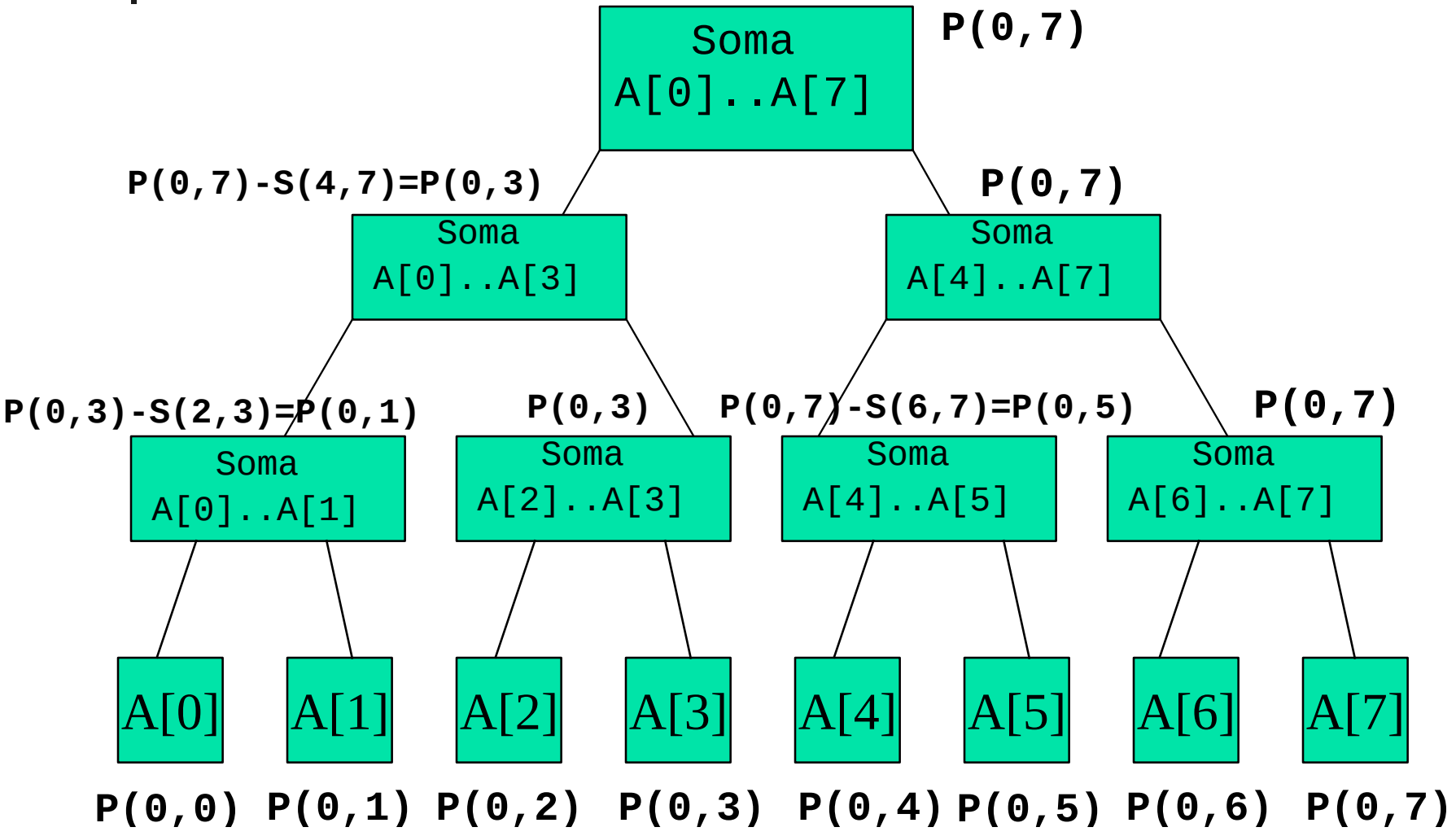
Complexidade: O(n)

Soma de Prefixos - PRAM

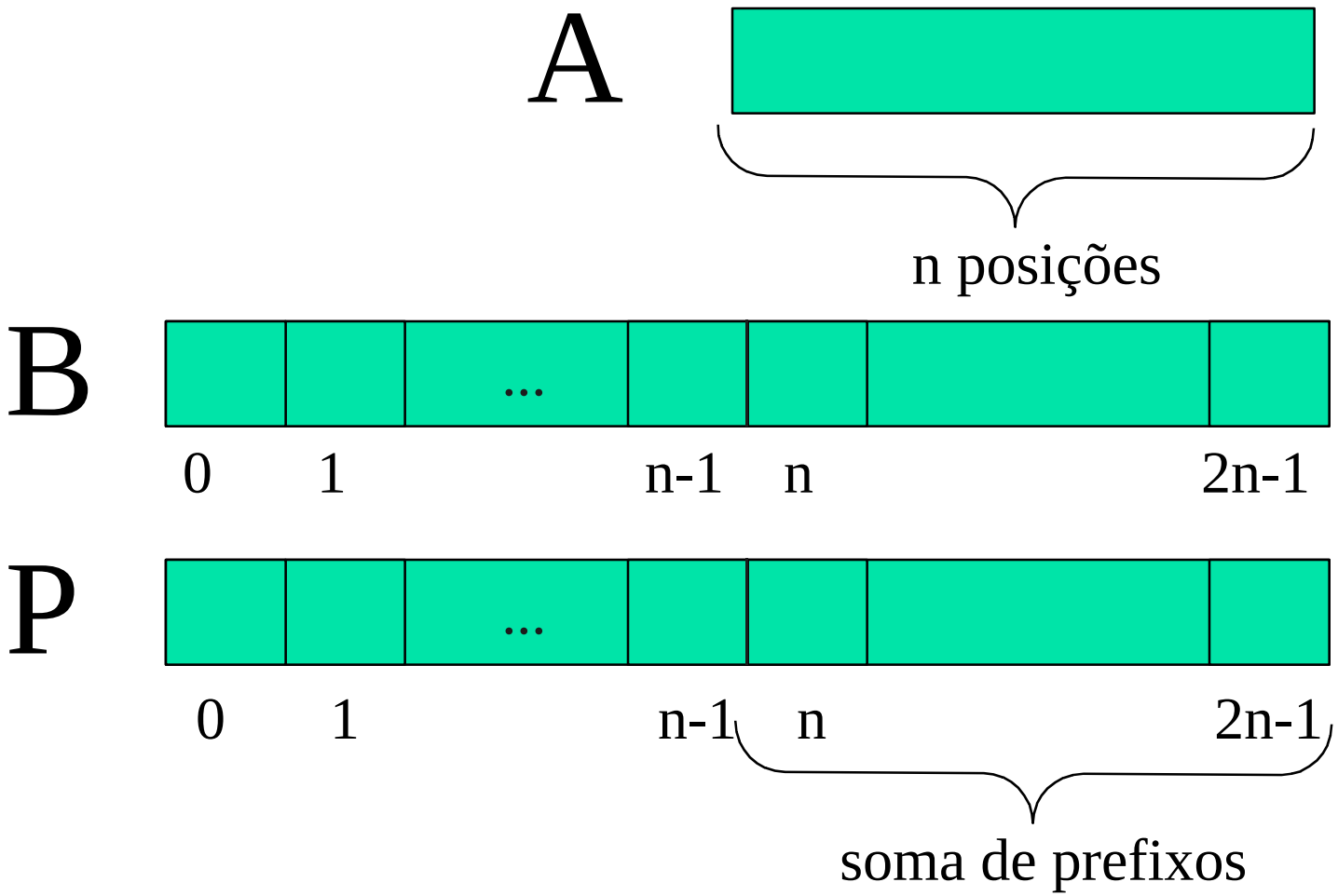
Este método utiliza a árvore binária balanceada em dois passos:

- **1o. passo: sobe na árvore de maneira equivalente ao método da árvore binária balanceada.**
- **2o. passo: desce na árvore calculando as operações op parciais, ou seja, desce na árvore subtraindo alguns elementos das somas intermediárias para obter as somas de prefixos. Quando desce para o filho esquerdo, subtrai o valor do filho direito. Quando desce para filho direito, passa valor direto.**

Soma de Prefixos - PRAM



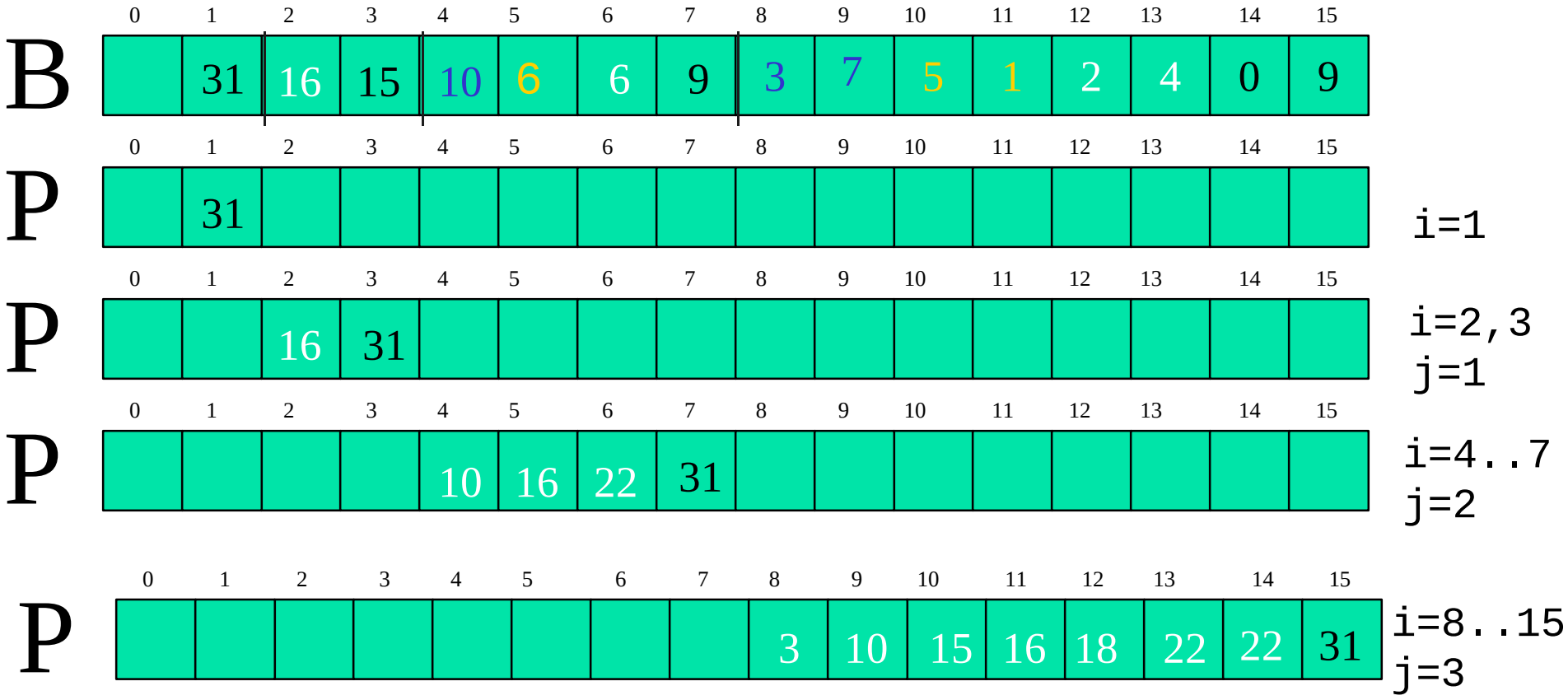
Soma de Prefixos - PRAM



Soma de Prefixos - PRAM

```
/* Passo 1: Soma do vetor (subida da árvore) */
/* Passo 2: Descida da árvore */
P[1] := B[1]
/* loop seqüencial para cada nível da árvore */
PARA j := 1 ATÉ log2n FAÇA
   /* loop paralelo alocando um processador para cada
   vértice deste nível */
   PARA 2^j <= i <= 2^(j+1) - 1 FAÇA EM PARALELO
        SE (i mod 2) = 0 ENTÃO      /* filho esquerdo */
          P[i] := P[i/2] - B[i+1]
        SENÃO                       /* filho direito */
          P[i] := P[(i-1)/2]
```

Soma de Prefixos - PRAM



Estudo da Complexidade

- EREW
- Tempo: O ($\log_2 n$)
- Processadores: n
- Custo: O($n \log_2 n$)

Eficiente, não ótimo!

Soma de Prefixos

- Complexidade: O(log n) com n processadores.
- Como utilizar apenas p << n processadores???
- Executar com um máximo de $O(\log^k p)$ rodadas (superpassos) de comunicação???
- Tamanho máximo da mensagem de O(n/p)???

Soma de Prefixos - CGM

- Entrada: n elementos v(1), ..., v(n)
- Saída: S(k) = v(1)+...+v(k), 1<= k <= n
- p_0 envia n/p elementos para cada p_i
- Cada p_i calcula a soma T(i) de seus elementos e envia o resultado para o p_0
- p_0 calcula a soma ST(i) = T(1)+...+T(i), 1 <= i <= p
- p_0 envia para cada p_i a soma ST(i-1)
- Cada p_i calcula a soma de prefixos local S(k) = ST(i - 1) + v((i-1)*n/p + 1) + ...+ v(k), (i-1)*n/p + 1 <= k <= i*n/p

Soma de Prefixos - CGM - v.0.1

- entrada: n elementos v(1), ..., v(n)
- saída: S(k) = v(1)+...+v(k), 1<= k <= n
- p_0 envia n/p elementos para cada p_i
- Cada p_i calcula a soma T(i) de seus elementos e envia o resultado para os processadores
- Cada p_i calcula a soma ST(i-1) = T(1)+...+T(i-1)
- Cada p_i calcula a soma de prefixos local:
  - S(k) = ST(i - 1) + v((i-1)*n/p + 1) + ...+ v(k),
  - (i-1)*n/p + 1 <= k <= i*n/p

Soma de Prefixos - CGM - v.0.2

- entrada: n elementos v(1), ..., v(n)
- saída: S(k) = v(1)+...+v(k), 1<= k <= n
- Processo 0 envia n/p elementos para cada p_i
- Cada p_i calcula a soma T(i) de seus elementos e envia o resultado para os processadores j > i
- Cada p_i calcula a soma ST(i-1) = T(1)+...+T(i-1)
- Cada p_i calcula a soma de prefixos local:
  - S(k) = ST(i - 1) + v((i-1)*n/p + 1) + ...+ v(k),
  - (i-1)*n/p + 1 <= k <= i*n/p

Soma de Prefixos - MPI

```
// Passo 1. Inicialização: MPI, núm. de tarefas e id. Tarefa
   e tam
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
tam = (int) TAMMAX/size; // tamanho subvetor
// Passo 2. Envie os dados as tarefas filhos
MPI_Scatter(VetorDados, tam, MPI_INT, SubVetor, tam,
                          MPI_INT, root, MPI_COMM_WORLD);
// Passo 3. Calcule a Soma em cada tarefa
for (i = 0; i < tam; i++)
   SomaP = SomaP + SubVetor[i];
```

Soma de Prefixos MPI

```
// Passo 4. Receba as Somas Parciais
MPI_Scan(&SomaP, &Soma, 1, MPI_INT, MPI_SUM,
    MPI_COMM_WORLD);
// Passo 5. Calcule as Somas Parciais em Pi
SomaPre[0] = Soma - SomaP + SubVetor[0];
for (i = 1; i < tam; i++)
   SomaPre[i] = SomaPre[i-1] + SubVetor[i];
```

Soma de Prefixos - MPI

```
// Passo 6. Imprima o valor da Soma P0
for (i = 0; i < tam; i++)
    printf("rank %d e SomaPre[%d]: %d\n", rank, i,
     SomaPre[i]);
// Passo 7. Finalize o MPI
MPI_Finalize();
return 0;
```