Fontes principais

1. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução
   `http://www.ime.usp.br/~song/papers/jai01.pdf`

Modelos Realísticos

Modelos Realísticos

Anos 80: crise na área de computação paralela

▷ Vários resultados teóricos para máquinas específicas (Malhas e hipercubos).

▷ Resultados desapontadores, quando implementados em máquinas reais.

Modelos Realísticos

Anos 90: Surgem os modelos computação de granularidade grossa

- BSP - Bulk Synchronous Parallel Model.
- CGM - Coarse Grained Multicomputers.

Modelo BSP

Modelo BSP

O modelo BSP (Bulk Synchronous Parallel) foi proposto por Valiant em 1990

▷ Foi um dos primeiros modelos a considerar custo de comunicação.

▷ O modelo BSP consiste de um conjunto de p processadores com memória local.

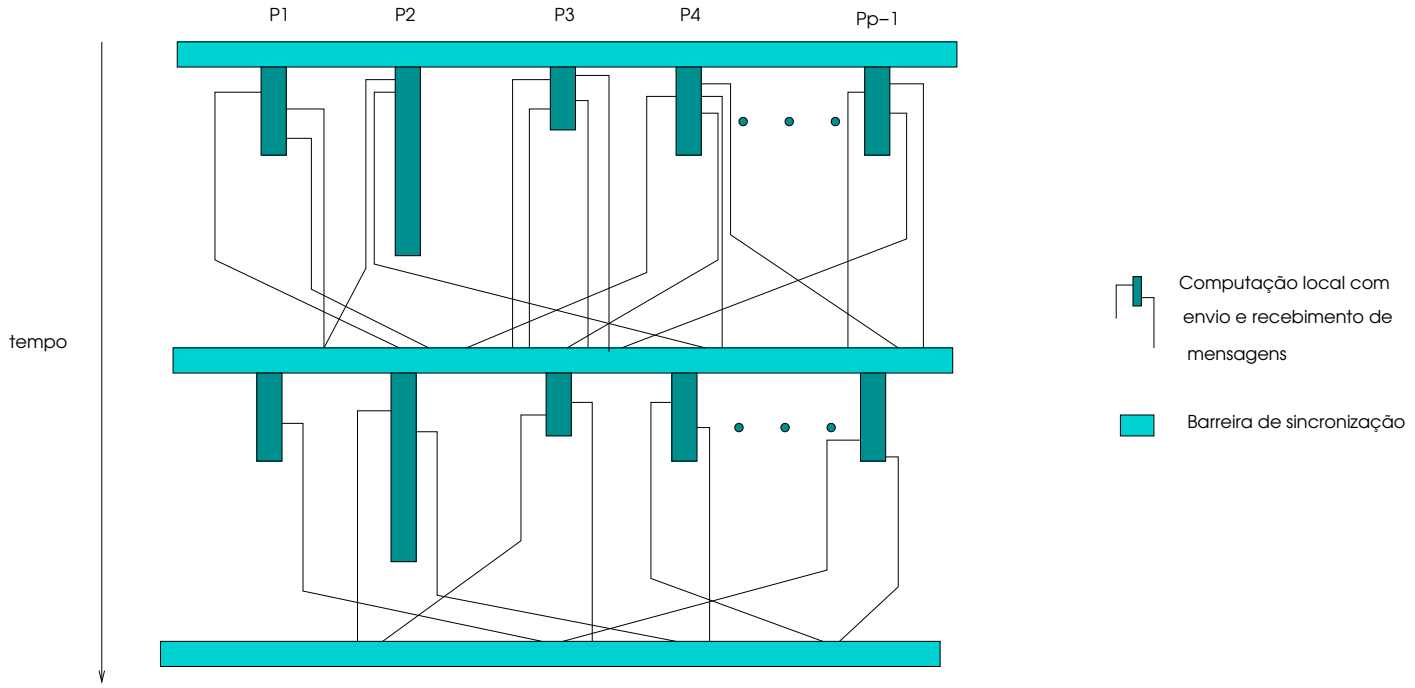
▷ A comunicação é feita por meio de rede de interconexão, gerenciados por roteador e com facilidades de sincronização global.

Modelo BSP

Um algoritmo BSP consiste de:

- ▷ Uma sequência de superpassos separados por barreiras de comunicação.
- ▷ Em cada superpasso cada processador executa uma combinação de:
  - ◦ passos de computação (computações locais), e;
  - ◦ passos de comunicação (através da transmissão e recebimentos de mensagens).

Modelo BSP


P1
P2
P3
P4
Pp−1
tempo
Computação local com
envio e recebimento de
mensagens
Barreira de sincronização

Modelo BSP

O modelo possui os seguintes parâmetros:

▷ n : tamanho do problema;

▷ p : número de processadores disponíveis, cada um com sua memória local;

▷ L : tempo mínimo de um superpasso (latência);

▷ g : taxa de eficiência da computação/comunicação.

Custo (superpasso i): $w_i + gh_i + L$, $w_i = \{L, t_1, t_2, \cdots, t_p\}$ e $h_i\{L, c_1, c_2, \cdots, c_p\}$

Custo Total: $W + gH + L$, $W = \sum_{i=0}^{T} w_i$ e $H = \sum_{i=0}^{T} h_i$, onde T é o número de superpassos

Modelo CGM

Modelo CGM

O modelo CGM foi proposto por Frank Dehne e é derivado do BSP. O CGM é definido em apenas dois parâmetros:

1. n : tamanho do problema
2. p : número de processadores $P_1, P_2, \cdots, P_p$, cada um com uma memória local de tamanho $O(n/p)$.

Modelo CGM

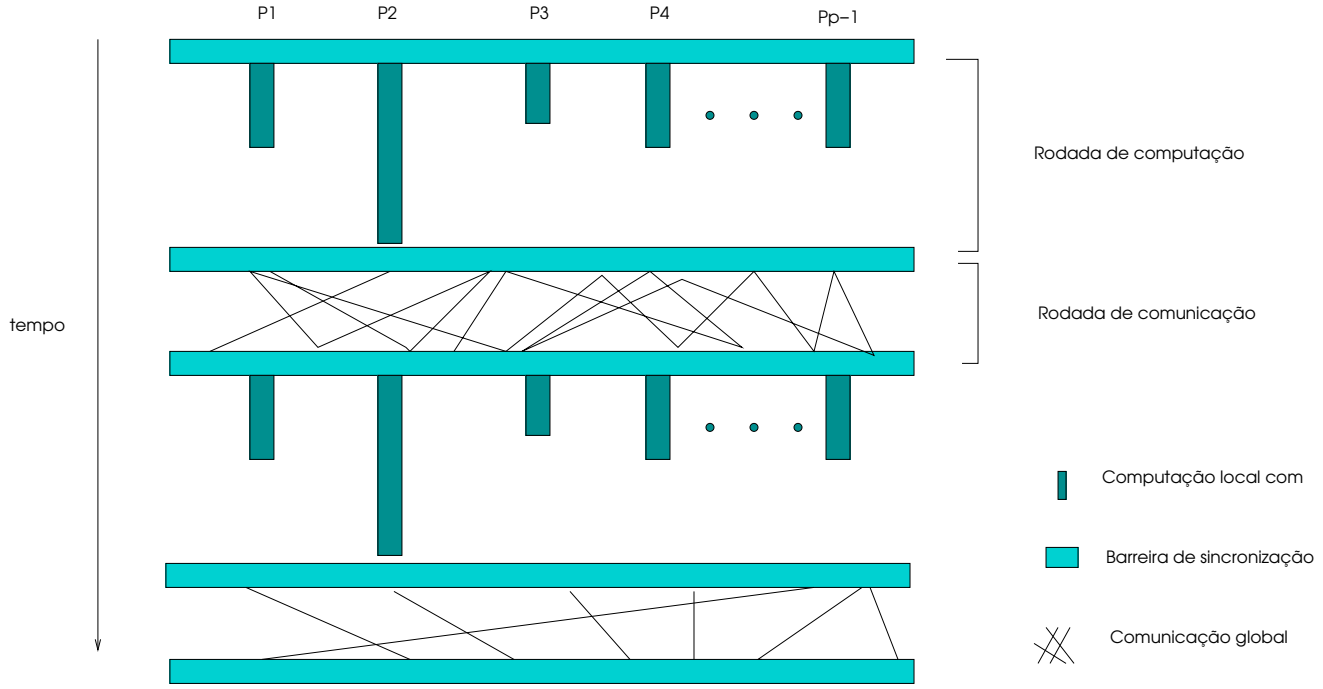
Um algoritmo CGM consiste de uma sequência alternada de **rodadas de computação** e **rodadas de comunicação** separadas por uma barreira de sincronização.

Na fase de comunicação quantidade de dados trocados por cada processador deve ser $O(n/p)$.

Modelo CGM

O objetivo de um algoritmo CGM é minimizar o número de superpassos e a quantidade de computação local.

Modelo CGM


P1
P2
P3
P4
Pp−1
tempo
Rodada de computação
Rodada de comunicação
Computação local com
Barreira de sincronização
Comunicação global

Soma de um vetor no Modelo BSP/CGM

Soma de um vetor no Modelo BSP/CGM

Seja A um vetor de ordem n, considere o problema de computar a soma $S = A(1)+\cdots+A(n-1)$ no modelo com p processadores, onde $p << n$.

Seja $r = n/p$. A é particionado como segue: $A = (A_1, A_2, \cdots, A_{p-1})$, onde cada A_i tem tamanho r.

Soma de um vetor no Modelo BSP/CGM

Para determinar a soma S, cada processador P_i computa a $i-$ésima soma parcial $s_i = A_i((i-1)r+1) + \cdots + A_i(ir)$, para $1 \leq i \leq p$, e envia s_i, através de uma mensagem, para o processador P_1, que computa o total das somas parciais.

Soma de um vetor no Modelo BSP/CGM

Entrada:(1) O número do processador i; (2) O número p de processadores; (3) O i-ésimo sub-vetor $B = A((i-1)r + 1 : ir)$ de tamanho r, onde $r = n/p$.

Saída: Processador P_i calcula o valor $S = s_1 + \cdots + s_i$ e envia o resultado para P_1. Quando o algoritmo termina, P_1 terá a soma S.

Soma de um vetor no Modelo BSP/CGM

Algoritmo

1 $z := B[1] + \cdots B[r]$

2 **se** $i = 1$ **então** $S := z$
senão $envia(z, P_1)$

3 **se** $i = 1$ **então**
para $i := 2$ **até** p **faça**
$recebe(z, P_i)$
$S := S + z$

Soma de um vetor no Modelo BSP/CGM

Complexidade:

▷ Passo 1: Cada P_i efetua r operações.

▷ Passo 2: P_1 efetua uma operação e os demais processadores P_i enviam uma mensagem.

▷ Passo 3: P_1 recebe $p-1$ mensagens e efetua $p-1$ operações.

Soma de um vetor no Modelo BSP/CGM

Complexidade:

▷ Tempo de computação: $O(n/p)$

▷ P_1 recebe $p-1$ mensagens, todas no mesmo superpasso(BSP) ou rodada(CGM), logo o algoritmo utiliza $O(1)$ rodadas de comunicação.

Soma de prefixos de um vetor no Modelo BSP/CGM

Soma de prefixos de um vetor no Modelo BSP/CGM

A solução do problema de soma de prefixos no modelo BSP/CGM é semelhante ao da soma de n números.

Soma de prefixos de um vetor no Modelo BSP/CGM

A idéia é o de dividir a entrada em p (número de processadores) subconjuntos, cada um com n/p elementos e distribuir esses subconjuntos entre os processadores (um subconjunto para cada processador).

Soma de prefixos de um vetor no Modelo BSP/CGM

Entrada: (1) O número do processador i; (2) O número p de processadores; (3) O i-ésimo sub-vetor $B = A((i-1)r + 1 : ir)$ de tamanho r, onde $r = n/p$.

Saída: Cada processador P_i contém o valor das somas de prefixos $S[(i-1) * r + j]$, $1 \leq j \leq n/p$

Soma de prefixos de um vetor no Modelo BSP/CGM

Algoritmo

1 $s_i := B[1] + \cdots B[r]$
2 $broadcast(s_i, p_j \neq i)$
3 $S[(i-1) * r] := s_1 + \cdots + s_{i-1}$
4 **para** $k := 1$ **até** r **faça**
$S[(i-1) * r + k] := S[(i-1) * r + k - 1] + B[k]$

Soma de prefixos de um vetor no Modelo BSP/CGM

Complexidade:

▷ Passo 1: Cada P_i efetua r operações.

▷ Passo 2: Os processadores executam um *broadcast* de s_i para os demais processadores. Essa comunicação pode ser feita em uma única rodada de comunicação.

▷ Passo 3: Cada processador calcula o valor da soma dos $A(1 : (i-1)r)$ elementos do vetor.

▷ Passo 4: Utiliza o valor computado no passo 3 para calcular as somas dos prefixos dos $A((i-1)r+1 : ir)$ elementos do vetor A.

Soma de prefixos de um vetor no Modelo BSP/CGM

Complexidade:

▷ Tempo de computação: $O(n/p)$

▷ Cada processador P_i tem que receber $p-1$ mensagens, todas no mesmo superpasso(BSP) ou rodada(CGM), logo o algoritmo utiliza $O(1)$ rodadas de comunicação.

Algoritmo de ordenação no BSP/CGM

Algoritmo de ordenação split sort no BSP/CGM

O algoritmo *split sort*, ou ordenação por divisão, consiste em dividir um conjunto de números em cestos, e distribuir os cestos de forma adequada, para que se possa ordenar n números divididos em p processadores, utilizando $O(1)$ rodadas de comunicação para $\frac{n}{p} \geq p^2$

Algoritmo de ordenação split sort no BSP/CGM

Na divisão dos cestos, utilizamos a idéia de calcular um conjunto de separadores (*splitters*), denominados de $p - quartis$, baseado no cálculo de medianas de um conjunto de elementos.

Algoritmo de ordenação split sort no BSP/CGM

Entrada: (1) Um vetor A com n elementos. (2) p processadores $p_0, p_1, p_2 \cdots, p_{p-1}$. (3) Os elementos do vetor A são distribuídos entre os p processadores (n/p elementos por processador).

Saída: Todos os elementos ordenados dentro de cada processador e por processador, ou seja, se $i < j$, temos que os elementos em p_i são menores que os elementos pertencentes a p_j.

Split sort no BSP/CGM

Algoritmo

1 Compute um conjunto divisor $S = \{s_1, s_2, \cdots, s_{p-1}\}$
2 $broadcast(S, p_i) \triangleright p_0$ envia S para todos os processadores
3 Particionar os elementos de p_i em buckets B_j^i de acordo com S
4 $envia(B_j^i, p_j) \triangleright$ cada processador P_i envia B_j^i para p_j, $1 \leq i, j \leq p$
5 Ordene $B_i^k = B_i^0 \cup B_i^1 \cup \cdots \cup B_i^{p-1}$

Split sort no BSP/CGM

É fácil veriricar que este algoritmo ordena qualquer entrada, visto que não foi efetuado nenhuma restrição ao tamanho dos buckets.

Como no modelo CGM temos que cada processador tem $O(n/p)$ memória local, devemos escolher cuidadosamento o conjunto S, pois isso influenciará no tamanho dos buckets.

Split sort no BSP/CGM

Vamos apresentar um algoritmo CGM para computar o conjunto S (conjunto splitter), que utiliza apenas $O(p)$ espaço de memória por processador.

O método divide a entrada em p subconjuntos de mesmo tamanho como segue.

Split sort no BSP/CGM

Definição 1. *A **mediana** de um conjunto ordenado de n números é o $(n+1)/2$-ésimo elemento de n para n ímpar ou a média do $n/2$-ésimo com $(n+1)/2$-ésimo elemento para n par.*

Split sort no BSP/CGM

Definição 2. *Os **p-quartis** de um conjunto ordenado A de tamanho n são os $p-1$ elementos, de índice $\frac{n}{p}, \frac{2n}{p}, \cdots, \frac{(p-1)n}{p}$, que dividem A em p partes de igual tamanho.*

Os p-quartis podem ser facilmente computados de forma sequencial usando um algoritmo recursivo em tempo $O(n \log p)$

Algoritmo p-quartis sequencial

Entrada: (1) Um vetor A com n elementos. (2) p o número de quartis

Saída: O conjunto A dividido em p-quartis

Algoritmo p-quartis sequencial

Algoritmo

1. Compute a mediana de A
2. Usando a mediana, divida A em dois subconjuntos A_1 e A_2
3. Aplique o algoritmo recursivamente, até que $p-1$ splitter sejam encontrados

Algoritmo p-quartis no BSP/CGM

Entrada: (1) Um vetor A com n elementos. (2) p processadores $p_0, p_1, \cdots, p_{p-1}$ (3) Os elementos do vetor A são distribuídos entre os p processadores (n/p elementos por processador)

Saída: O conjunto A dividido em p-quartis

Algoritmo p-quartis no BSP/CGM

Algoritmo

1 $Q_i := p - quartis(A_i)$ ▷ Cada processador p_i calcula
▷ sequencialmente seus $p - quartis$

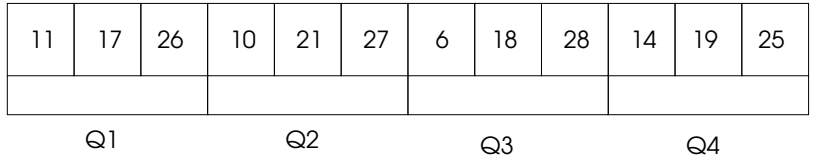
2 $envia(Q_i, p_0)$ ▷ Todos os processador p_i enviam Q_i para p_0

3 **se** $i = 0$ **então** $S := Ordena(Q_0 \cup Q_1 \cup \cdots \cup Q_{p-1})$

4 $broadcast(S, p_i)$

Algoritmo p-quartis no BSP/CGM

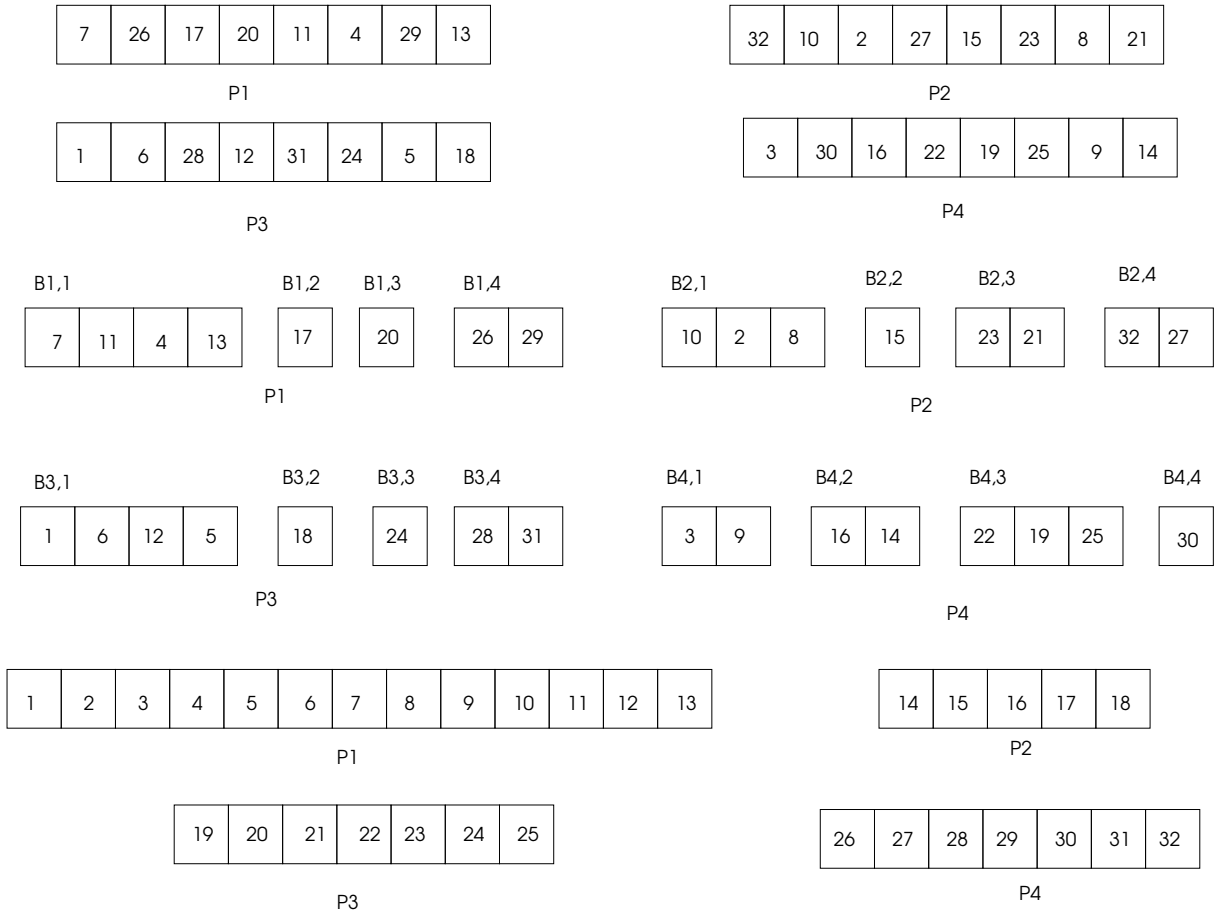
(a)



(b)

(c)

Algoritmo split-sort no Modelo BSP/CGM



Algoritmo p-quartis no BSP/CGM

Complexidade:

- ▷ Tempo de computação local: $O(\frac{n \log p}{p})$, onde $\frac{n}{p} \geq p^2$
- ▷ Rodadas de comunicação: $O(1)$

Algoritmo p-quartis no BSP/CGM

Complexidade:

- ▷ Tempo de computação local: $O(\frac{n \log p}{p})$, onde $\frac{n}{p} \geq p^2$
- ▷ Rodadas de comunicação: $O(1)$

Este tempo de computação local pode ser melhorado de tal forma que $\frac{n}{p} \geq p$

List Ranking no Modelo BSP/CGM

List Ranking

Seja L uma lista representada por um vetor s tal que $s[i]$ é o nó sucessor de i na lista L, para u, o último elemento da lista L, $s[u] = u$. Denominamos i e $s[i]$ por vizinhos.

A **distância** entre i e j, $d_L(i, j)$, é o número de nós entre i e j mais 1.

List Ranking

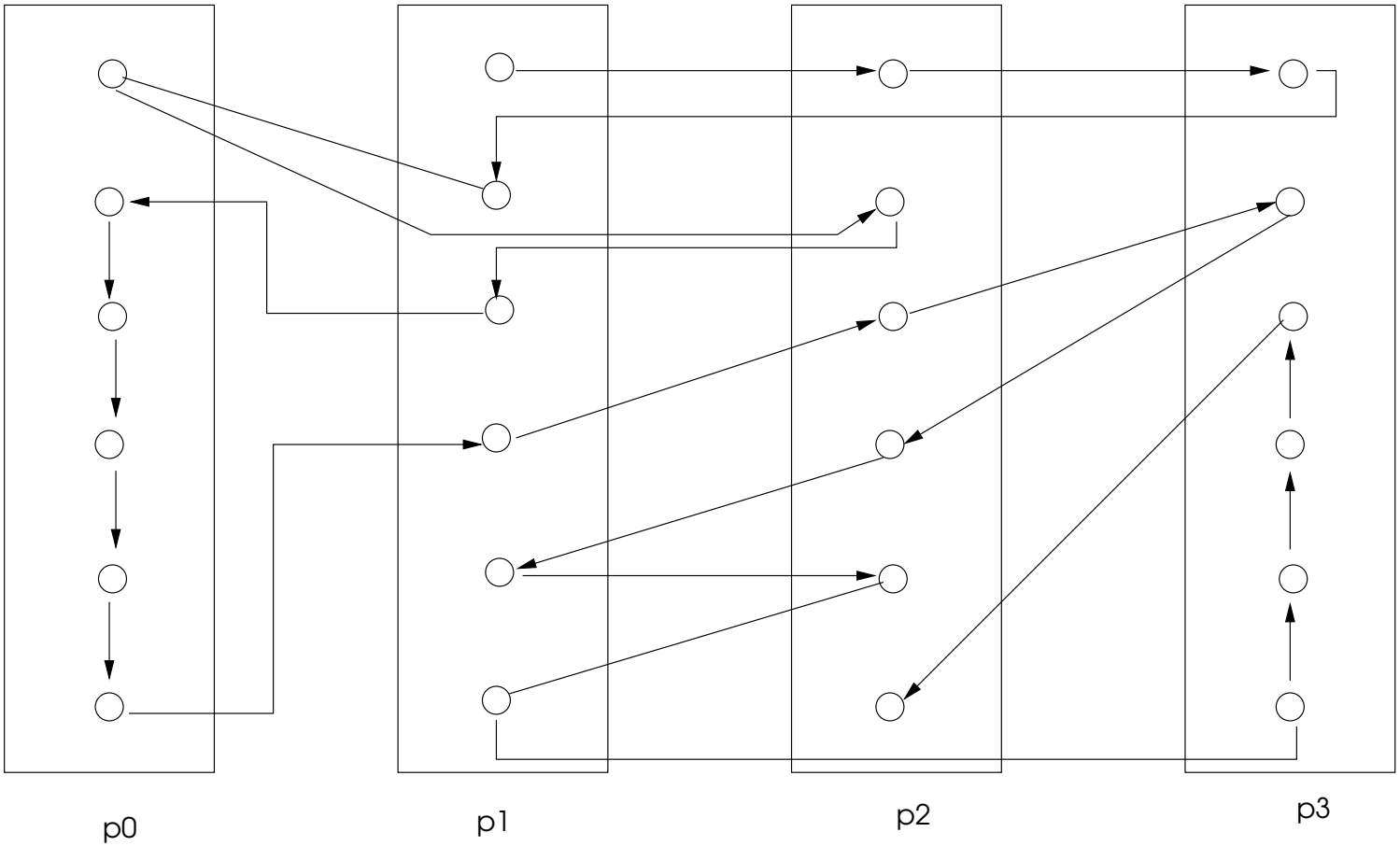
O problema do **list ranking** consiste em computar para cada $i \in L$, a distância entre i e o último elemento j, denotado por $dist_L(i) = d_L(i, j)$.

List Ranking

O número de nós da lista cujos sucessores não estão armazenados no mesmo processador pode variar de 0 a n/p.

Mesmo se todos os sucessores estiverem em um dado processador, após a aplicação da duplicação recursiva (*pointer jumping*), não há garantia que isto ocorra nos passos seguintes.

List Ranking



List Ranking

O número de rodadas de comunicação pode chegar a $O(\log n)$, uma vez que pode ser necessária a comunicação para obter o sucessor de um dos seus elementos.

A simples aplicação da duplicação recursiva não leva a um algoritmo CGM eficiente.

List Ranking

Para diminuir o número de rodadas de comunicação, a idéia é a de selecionar um conjunto de elementos $i^* \in L$ bem distribuido em L, de tal forma que a distância de qualquer $i \in L$ a i^* possa ser computada em $O(\log^k p)$ aplicações de pointer jumping.

r-ruling set

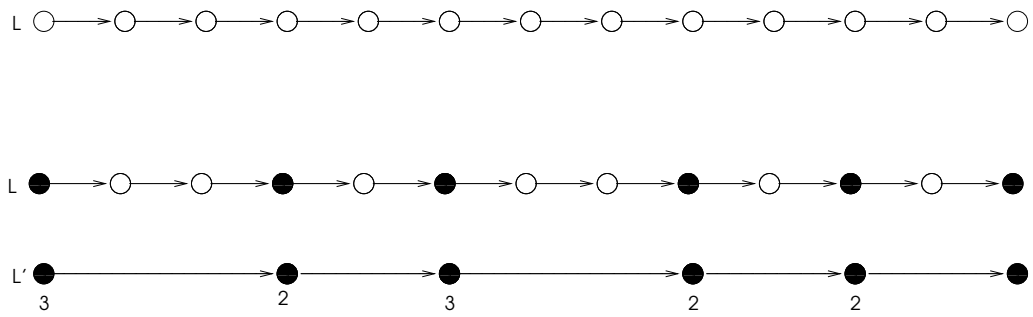
Um **r-ruling set** de L é um subconjunto de elementos selecionados da lista L com as seguintes propriedades:

(1) Dois vizinhos nunca são selecionados.

(2) A distância entre qualquer elemento não selecionado ao próximo elemento não selecionado é no máximo r.

r-ruling set

Uma lista L e um 3-ruling set.



Algoritmo List Ranking determinístico

Algoritmo List Ranking determinístico

Entrada: Uma lista ligada L de comprimento n onde cada processador armazena n/p elemento $i \in L$ e seus respectivos ponteiros $s_L[i]$.

Saída: Para cada elemento i seu *rank* $dist(i)$ em L.

Algoritmo List Ranking determinístico

1. Calcular $O(p^2)$-*ruling set* R com $|R| = O(n/p)$.

2. Fazer um *broadcast* de R para todos os processadores. O subconjunto R é uma lista ligada onde cada elemento i é atribuído um ponteiro para o próximo elemento j em R com respeito à ordem induzida por L.

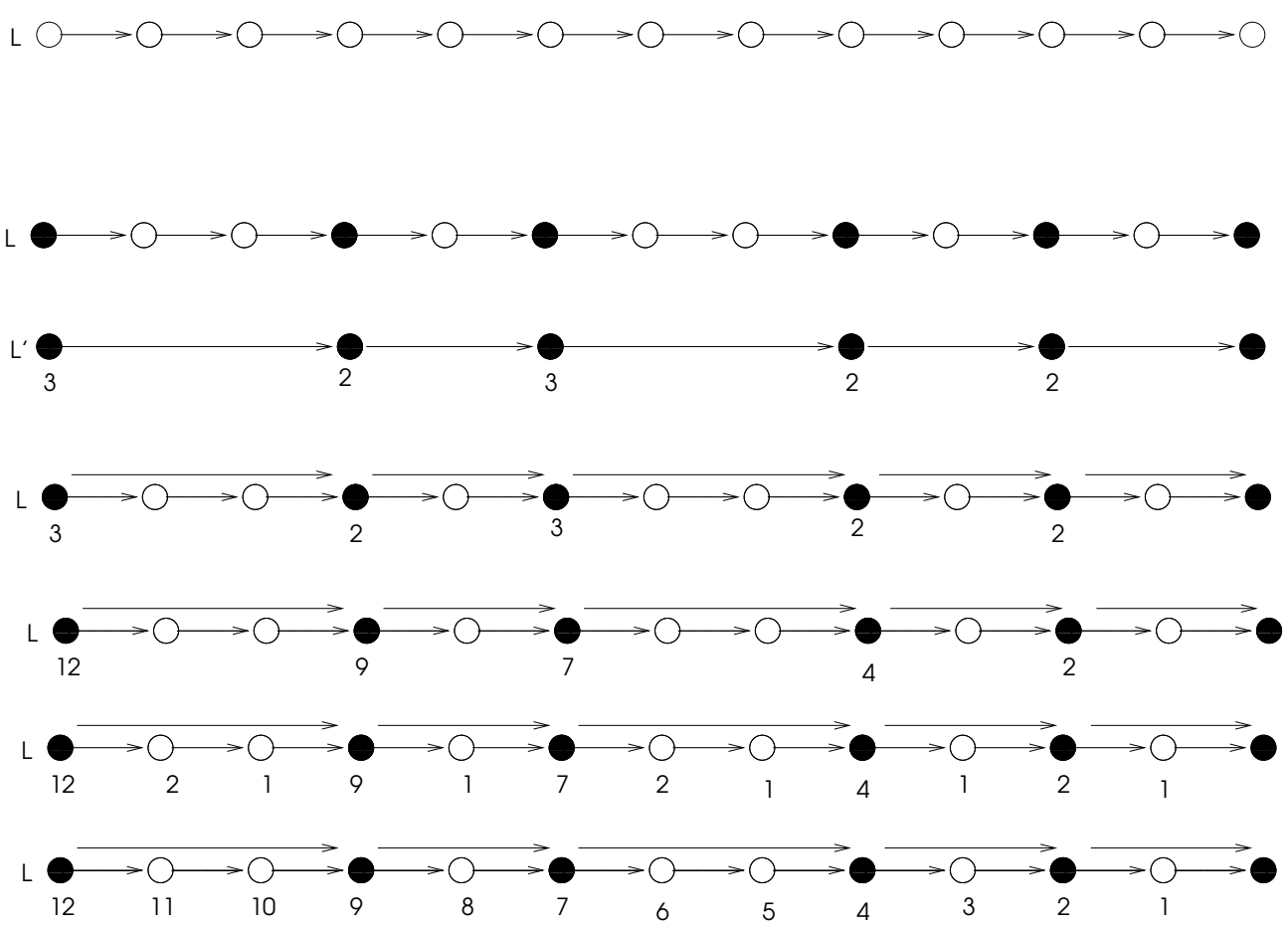
3. Calcular sequencialmente em cada processador o *List Ranking* de R, isto é, calcular para cada $j \in R$ seu $dist_L(j)$ em L

4. Obter para cada elemento $i \in L - R$ sua distância $d_L(i, s_R[i])$ ao próximo elemento $s_R[i]$ em R através da duplicação recursiva.

5. Calcular em cada processador os *ranks* dos seus elementos $i \in L - R$ com:

$$dist_L[i] = d_L(i, s_R[i]) + dist_L(s_R[i])$$

Algoritmo List Ranking determinístico



Compressão determinística de listas

Para computar um $O(p^2)$-*ruling set* em $O(\log p)$ rodadas de comunicação, usaremos uma técnica chamada **compressão determinística de lista**.

Compressão determinística de listas

Na compressão determinística da lista aplica-se uma sequência alternada de fases de **compressão** e de **concatenação**.

Compressão determinística de listas

Na fase de compressão, seleciona-se um subconjunto de elementos da lista L, utilizando um esquema de rotulação (*deterministic coin tossing*).

A fase de concatenação consiste da construção de uma lista ligada, através da duplicação recursiva, com os elementos selecionados na fase de compressão.

Compressão determinística de listas

Um intervalo-s de comprimento k é uma sequencia $I = (i_1, i_2, \cdots, i_k)$ de elementos da lista tal que $s[i_j] = i_{j+1}$, $1 \leq j \leq k - 1$.

Os dois vizinhos n_1 e n_2 do intervalo-s I são tais que $s[n_1] = i_1$ e $s[i_k] = n_2$.

Compressão determinística de listas

Um intervalo-s maximal I de elementos da lista cujos elementos estão todos no mesmo processador é dito intervalo-s local.

Um intervalo-s maximal I de elementos da lista tal que quaisquer dois elementos consecutivos não estão no mesmo processador é dito intervalo-s não-local.

Compressão determinística de listas

O rótulo $l(i)$, $\forall i \in L$, na fase de compressão é o número do processador p que armazena o nó i.

Neste esquema, cada elemento de L tem no máximo p rótulos distintos.

Seja $M = \{i, i+1, \cdots, i+k\} \subseteq L$, tal que $l(i) \neq l(s[i])$, $\forall i \in L$, onde o $s[i]$ é o máximo local se $l(i) < l(s[i]) > l(s[s[i]])$.

Compressão determinística de listas

Selecionando apenas máximos locais não há garantia de distância menor que $O(p)$.

Pode haver $L' = \{j, j+1, \cdots, j+k\} \subseteq L$, onde $l(s[i]) = l(j), \forall j \in L'$ e $k > p$

Selecionamos todos os segundos elementos.

Algoritmo p^2-*ruling set*

Entrada: L representada pelo vetor s.

Saída: $R \subset L$ de nós selecionados.

Algoritmo p^2-*ruling set*

(1) Cada processador localmente marca todos seus elementos como **não-selecionado**

(2) Cada processador executa para cada um dos elementos armazenados:

(2.1) se $l(i) < l(s[i]) > l(s[s[i]])$ então $s[i]$ é **selecionado**.

(3) Cada processador localmente determina seus intervalos-s locais. Para cada intervalo-s de comprimento maior que dois, todo segundo elemento é marcado como **selecionado**. Se

algum intervalo-s tem comprimento menor que dois e nenhum de seus vizinhos tem um rótulo menor, então ambos elementos são marcados como **não-selecionados**.

(4) para $k = 1$ até $\log p$ faça

(4.1) Cada processador localmente executa para cada elemento i da lista:

se $s[i]$ é **não-selecionado** então $s[i] = s[s[i]]$.

(4.2) Cada processador localmente executa para cada elemento i da lista

se (i, $s[i]$ e $s[s[i]]$ estão **selecionados**) E ($l(i) < l(s[i]) > l(s[s[i]])$) E ($l(i) \neq l(s[i])$) E ($l(s[i]) \neq l(s[s[i]])$) então marcar $s[i]$ como **não-selecionado**.

(4.3) Cada processador examina seus intervalos-s locais. Para cada intervalo-s de comprimento maior que dois, todo segundo elemento é marcado como **não-selecionado**. Se um intervalo-s tem comprimento dois e nenhum de seus vizinhos tem um rótulo menor, então ambos elementos são marcados como **não-selecionado**.

(5) O processador que armazena o último elemento de L é marcado como **selecionado**.

Algoritmo p^2-*ruling set*

O algoritmo computa um p^2-*ruling set* R onde $|R| = O(n/p)$ usando $O(\log^2 p)$ rodadas de comunicação e $O(n/p)$ computação local por rodada.

Algoritmo List Ranking determinístico

O problema do list ranking para uma lista L com n vértices pode ser resolvido no modelo CGM com p processadores e $O(n/p)$ memória local por processador usando $O(\log p)$ rodadas de comunicação e $O(n/p)$ computação local por rodada.

Fim