Fontes principais

1. J. Jaja, An introduction to Parallel Algorithms, Addison Wesley, 92
   - ▷ Algoritmos paralelos
2. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução
   `http://www.ime.usp.br/~song/papers/jai01.pdf`

Modelo de Computação Paralela Distribuída

Modelo de Computação Paralela Distribuída

- ▷ Modelo de arquitetura MIMD
- ▷ P processadores executam em paralelo e estão interligados através de canais de comunicação
- ▷ Cada processador possui associado a ele uma memória (modelo de memória distribuída)
- ▷ Cada processador possui sua unidade de controle. Em um determinado instante, cada processador está executando uma instrução possivelmente diferente dos demais sobre dados diferentes.

Modelo de Computação Paralela Distribuída

▷ Cada processador possui seu relógio local.

▷ Processadores se comunicam através dos canais de comunicação usando troca de mensagens

▷ Os tempos de transmissão das mensagens são indeterminados, porém finitos.

Modelo de Computação Paralela Distribuída

Sincronização no envio de mensagens

- ▷ sistema síncrono (lock step)
  - ◦ regular: array, anel, hipercubo
  - ◦ irregular
- ▷ sistema assíncrono

Modelo de Computação Paralela e Distribuída

Uma rede pode ser vista como um grafo $G = (V, E)$, onde cada vértice $i \in V$ representa um processador, e cada aresta $(i, j) \in E$ representa uma ligação de comunicação nos dois sentidos entre i e j.

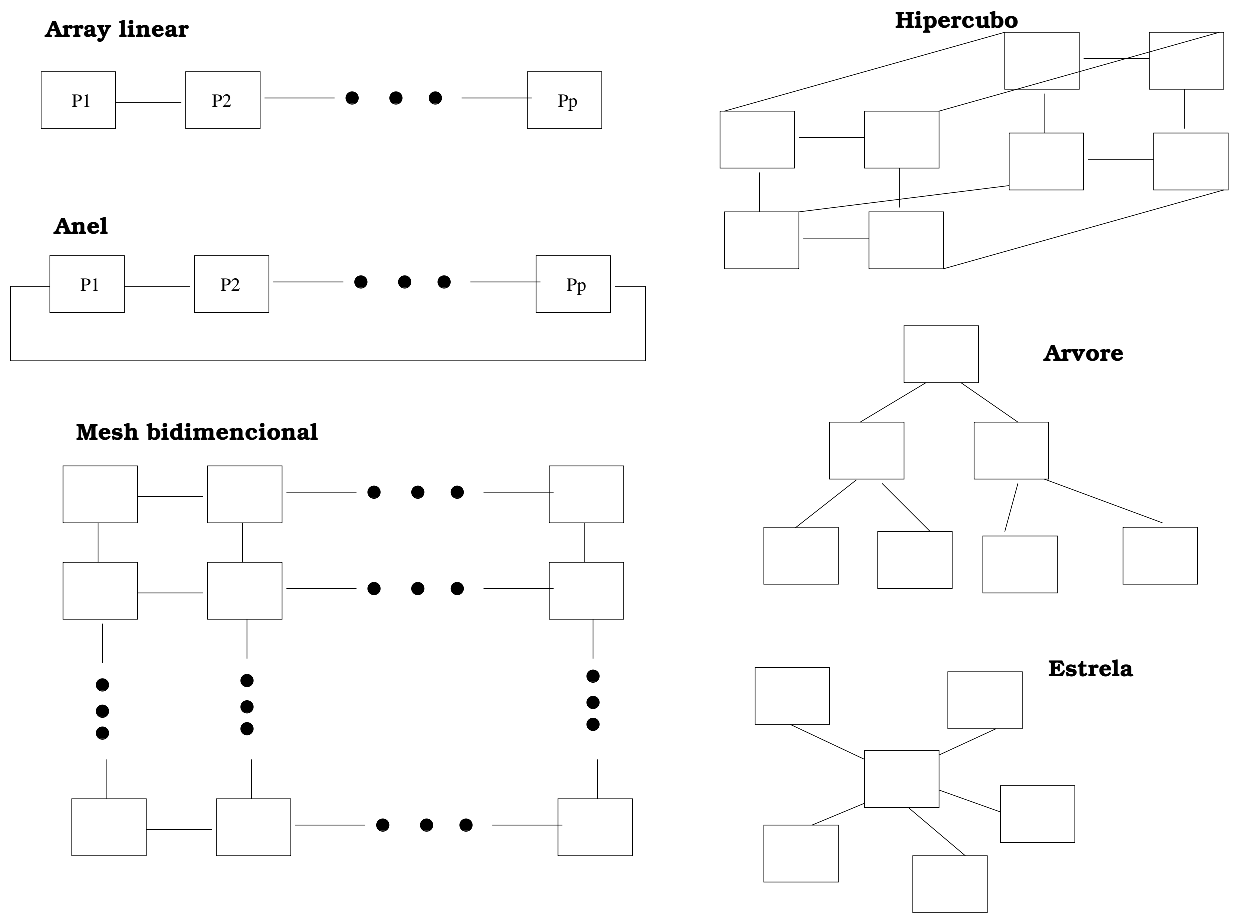
Topologia

O modelo de rede incorpora a topologia de interconexão entre os processadores no próprio modelo.

Parâmetros usados para avaliar a topologia de uma rede

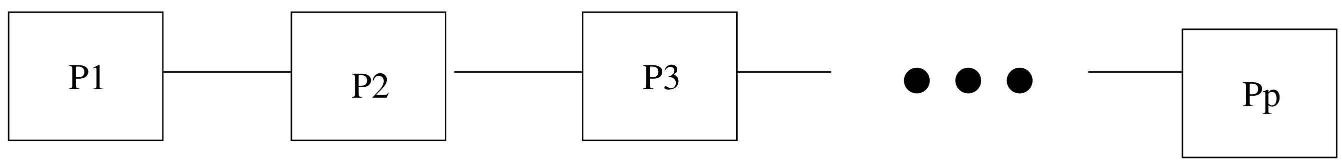
- ▷ Diâmetro: distância máxima entre qualquer par de vértices
- ▷ Grau máximo: Grau máximo de um vértice qualquer de G
- ▷ Conectividade: vértice/aresta

Topologia


Array linear
P1
P2
Pp
Anel
P1
P2
Pp
Mesh bidimencional
Hipercubo
Arvore
Estrela

Array Linear

Um **array linear** consiste de P processadores $P_1, P_2, \cdots P_p$ organizados de tal maneira que o processador P_i está conectado ao P_{i-1} e P_{i+1} desde que eles existam.

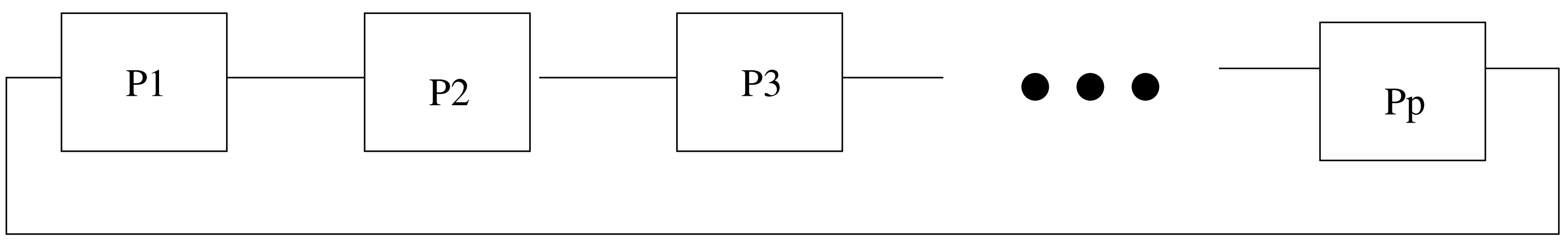


▷ Diâmetro $= p - 1$

▷ Grau máximo $= 2$

Anel

Um **anel** é um array linear com conexão entre o início e fim, isto é, os processadores P_1 e P_p estão conectados.



Soma de um vetor no Anel

Soma de um vetor no Anel

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

Saída: Processador P_i calcula $s = s_1 + s_2 + \cdots s_i$ e passa o resultado para a direita. Quando o algoritmo termina, P_1 terá a soma S.

Algoritmo: Soma de um vetor no Anel

$z = b[1] + \cdots + b[r]$

se $i = 1$ **então**

  $s := 0$

senão

  $recebe(s, esquerda)$

$s := s + z$

$envia(s, direita)$

se $i = 1$ **então**

  $recebe(s, esquerda)$

Complexidade

▷ $Com(n)$: Custo de transmitir n números entre processadores adjacentes

$$Com(n) = \sigma + n\tau$$

Onde

- ▷ σ é o tempo de inicialização
- ▷ τ é o tempo no qual a mensagem é transferida

Complexidade

Complexidade de tempo do algoritmo distribuído: Tempo de computação mais tempo de comunicação

$$T = T_{comp} + T_{comm}$$

Soma de um vetor no Anel

Complexidade:

- ▷ Tempo de execução local: $O(n/p)$,

digamos $T_{comp} = \alpha(n/p)$, onde α é uma constante

- ▷ Tempo total da execução:

$$T = T_{comp} + T_{comm} = \alpha(n/p) + p(\sigma + n\tau)$$

onde p é o número de processadores.

Medidas de eficiência de Algoritmos distribuídos

▷ Complexidade de mensagens: número de mensagens enviadas (ao todo)

▷ Tamanho das Mensagens

▷ Complexidade de espaço: espaço de memória necessário para um processo.

Medidas de eficiência de Algoritmos distribuídos

▷ Complexidade tempo local: número de passos do algoritmo sequencial de um processo (supor que comunicação leva um passo)

▷ Complexidade tempo assíncrono: comprimento da maior cadeia causal

Medidas de eficiência de Algoritmos distribuídos

- Causalidade: ao receber uma mensagem, um processo pode executar algo e enviar outras mensagens, em consequência deste recebimentos.
- Speed Up: medida experimental

Medidas de eficiência de Algoritmos distribuídos

$$\text{Speed Up} = \frac{\text{tempo gasto pelo melhor algoritmo sequencial}}{\text{tempo gasto pelo algoritmo distribuído}}$$

$$\text{Speed Up} = \text{número de processadores utilizados}$$

Soma de um vetor no Anel

Complexidade de mensagens: $O(p)$

Tamanho das mensagens: $O(1)$

Complexidade de tempo local: $O(\frac{n}{p})$

Complexidade de tempo assíncrono: $O(p)$

Produto entre matriz e vetor no Anel

Produto entre matriz e vetor no Anel

Entrada: O número do processador i; O número p de processadores; A i-ésima sub-matriz $B = A(1 : n, (i-1)r+1 : ir)$ de tamanho $n \times r$, onde $r = n/p$; O i-ésimo subvetor $w = x((i-1)r+1 : ir)$ de tamanho r.

Saída: Processador P_i computa o vetor $y = A_1x_1 + \cdots A_ix_i$ e passa o resultado para a direita. Quando o algoritmo termina, P_1 terá o produto Ax.

Algoritmo: Produto entre matriz e vetor no Anel

$\triangleright$ Compute o produto entre a submatriz B e o subvetor w

$z := Bw$

se $i = 1$ **então**

  $y := 0$

senão

  $recebe(y, esquerda)$

$y := y + z$

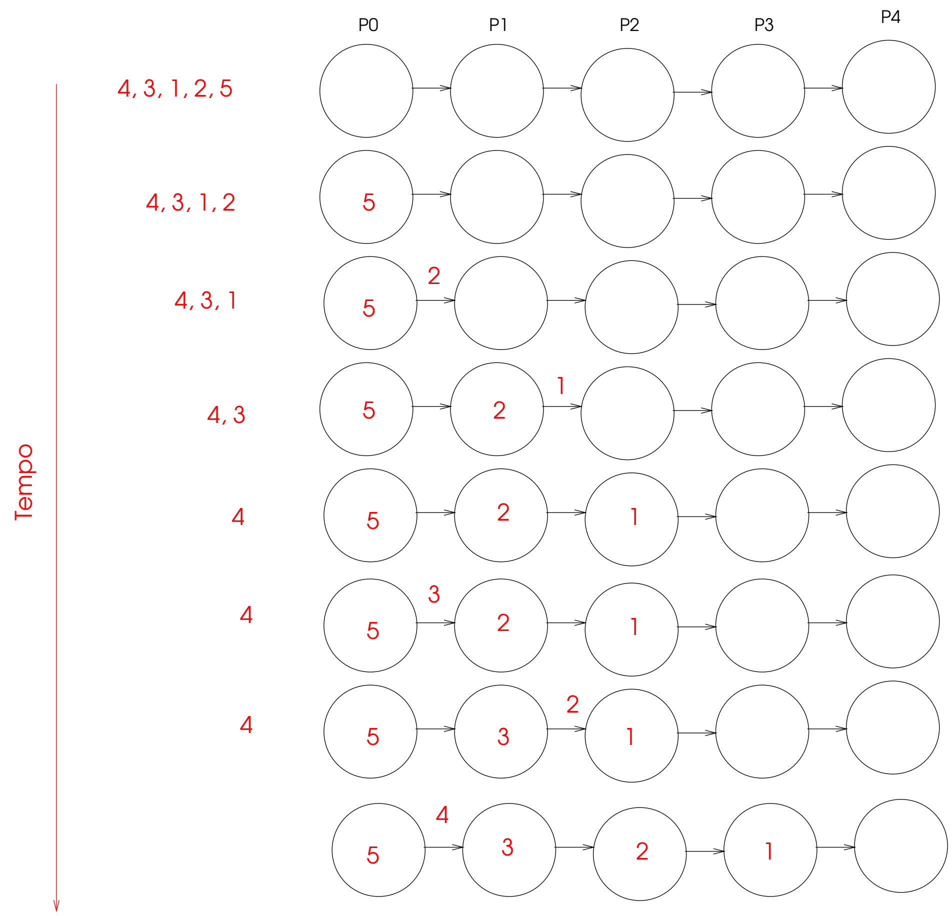
$envia(y, direita)$

se $i = 1$ **então**

  $recebe(y, esquerda)$

Ordenação: Insertion sort distribuído

Ordenação: Insertion sort distribuído



Algoritmo: Insertion sort distribuído

se $i = 1$ **então**
  para $k := 0$ **até** n **faça** $envia(v[k], direita)$
senão
  $numProcs := n - i - 1$
  $recebe(x, esquerda)$
  para $k := 0$ **até** $numProcs$ **faça**
    $recebe(numero, esquerda)$
    se $numero > x$ **então**
      $envia(x, direita)$
      $x := numero$
    senão $envia(numero, direita)$

Ordenação: Insertion sort distribuído

A ordenação por inserção é um exemplo de computação sistólica ou pipeline

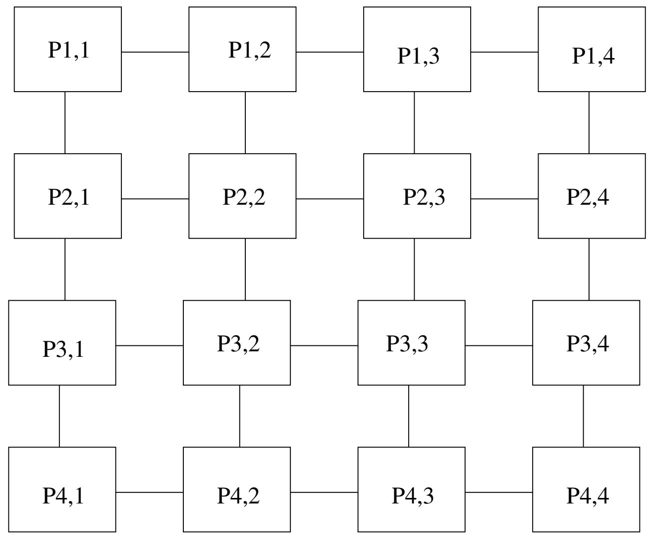
Computação prossegue em frente de ondas

Tem a vantagem de exigir comunicação com poucos vizinhos

Mesh

Mesh

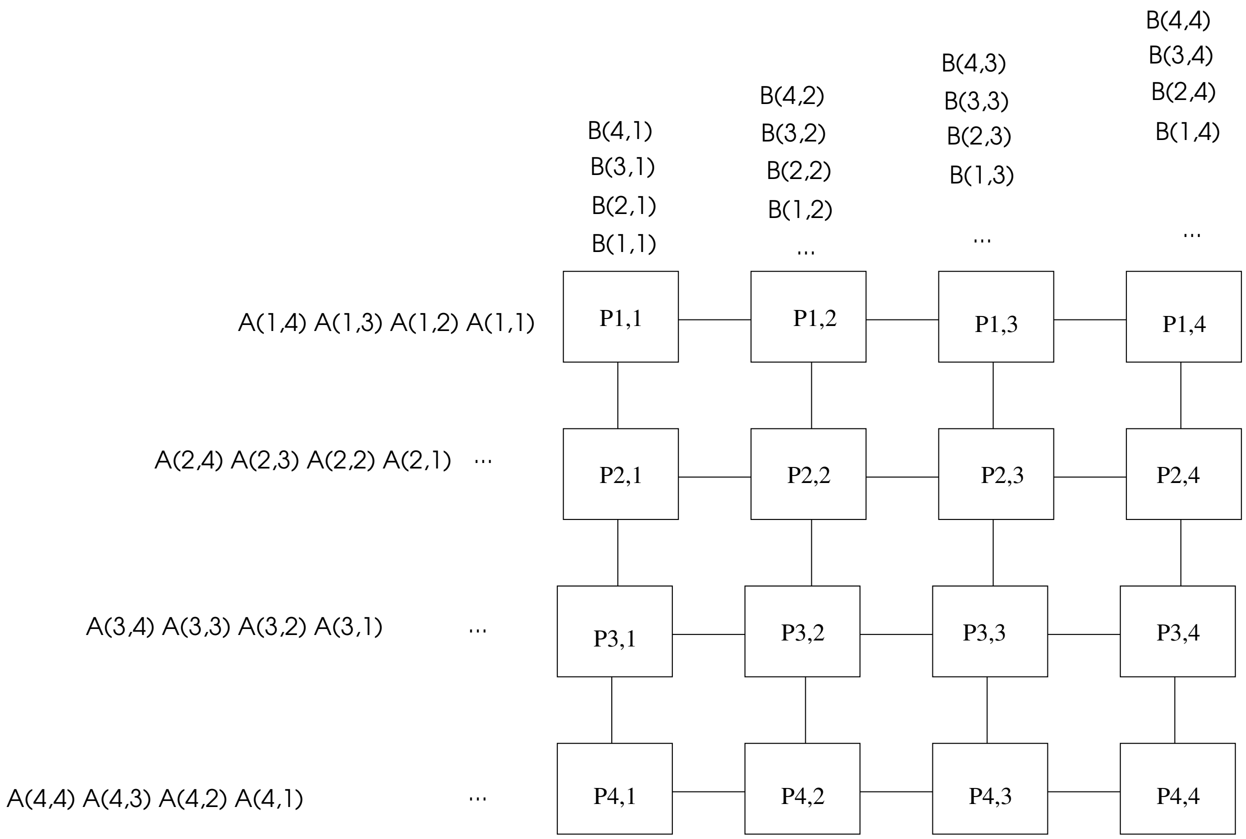
▷ O mesh é uma versão bidimensional do array linear, que consiste de $p = m^2$ processadores arranjados em uma matriz $m \times m$, dado que o processador P_{ij} é conectado aos processadores $P_{i\pm1j}$ e $P_{ij\pm1}$ sempre que eles existirem.



▷ O diâmetro é $\sqrt{p}$ e o grau máximo: 4

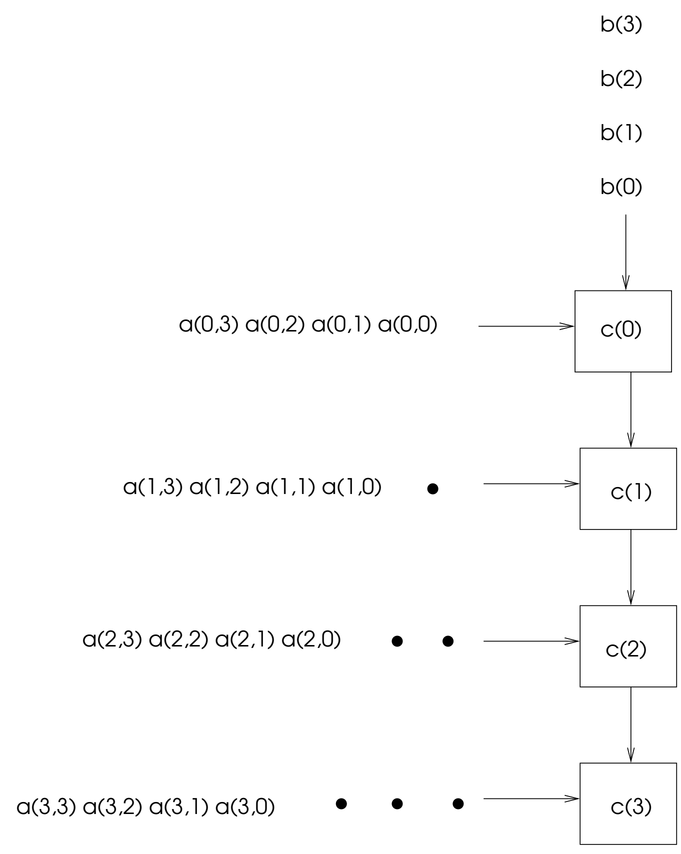
Algoritmo Sistólico

Multiplicação de matrizes



Algoritmo Sistólico

Multiplicação de matrizes por vetor



Hipercubo

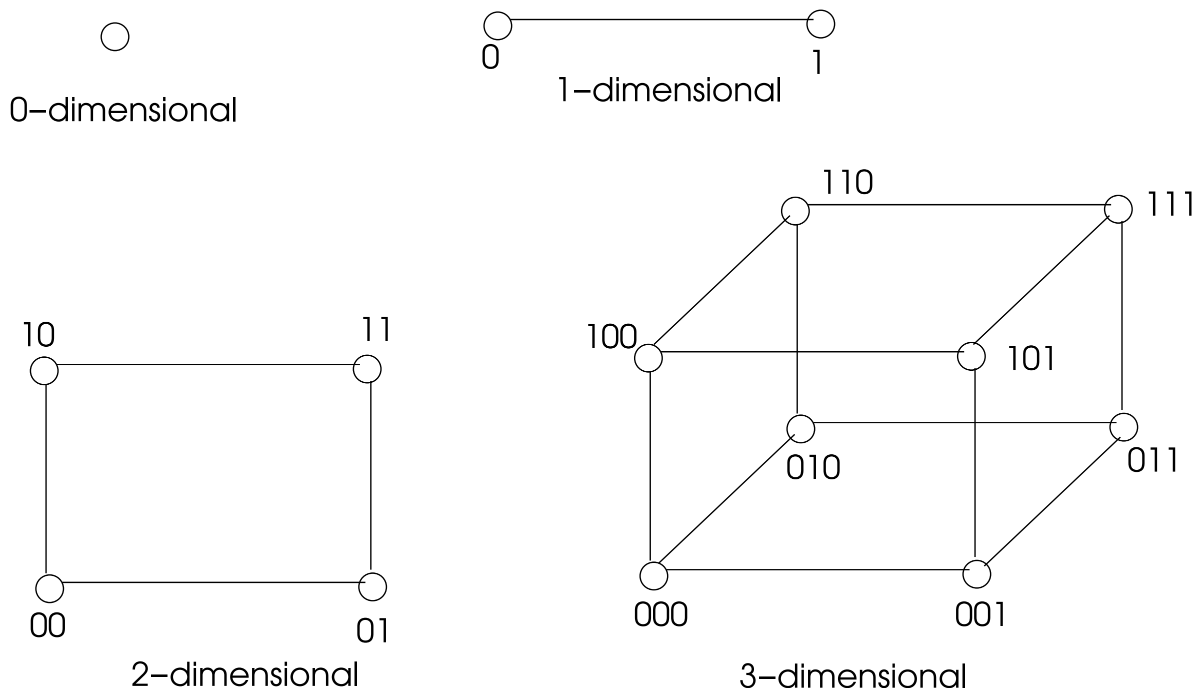
Hipercubo

Um hipercubo consiste de $p = 2^d$ processadores interconectados em um cubo d-dimensional que pode ser definido como segue.

$\triangleright$ Seja a representação binária de i sendo $i_{d-1}i_{d-2}\cdots i_0$, onde $0 \leq i \leq p-1$. Então o processador P_i está conectado ao processador $P_{i(j)}$, onde $i^{(j)} = i_{d-1}\cdots\overline{i_j}\cdots i_0$, e $\overline{i_j} = 1 - i_j$, para $0 \leq j \leq d-1$

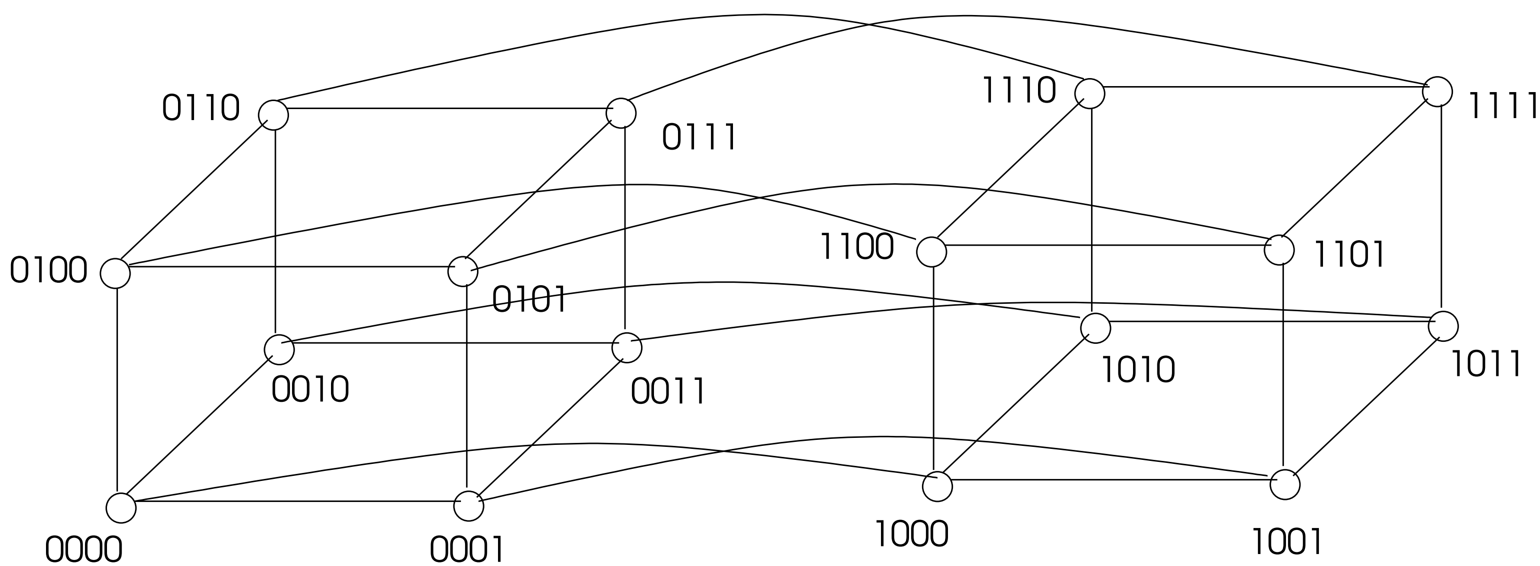
Hipercubo

Dois processadores estão conectados se, e somente se, a representação binária de seus índices diferem apenas na posição de um bit.



Hipercubo

O hipercubo possui uma estrutura recursiva. Podemos estender um cubo d-dimensional a um cubo $(d+1)$-dimensional através da conexão dos processadores dos dois cubos d-dimensional.



4–dimensional

Hipercubo

O diâmetro de um hipercubo d-dimensional é $d = \log p$. O grau máximo $d = \log p$.

Soma no Hipercubo

Cada entrada $A[i]$ de uma array A de tamanho n é armazenado inicialmente na memória local de um processador P_i de um hipercubo de $(n = 2^d)$ processadores síncronos. O objetivo é computar a soma

$$s = \sum_{i=0}^{n-1} A[i]$$

e armazená-la no processador P_0.

Soma no Hipercubo

O algoritmo consiste de iterações

A primeira computa a soma dos pares de elementos entre processadores cujos índices diferem na posição do bit mais significativo. As somas são armazenadas no subcubo $(d-1)$-dimensional cujo endereço do bit mais significativo é zero.

O restante do algoritmo funciona de forma análoga $i^{(l)}$ denota o índice que foi complementado.

Soma no Hipercubo

Entrada: Um vetor A de $n = 2^d$ elementos tal que $A(i)$ é armazenado na memória local do processador P_i, $0 \leq i \leq n-1$, de um hipercubo de n-processadores síncronos.

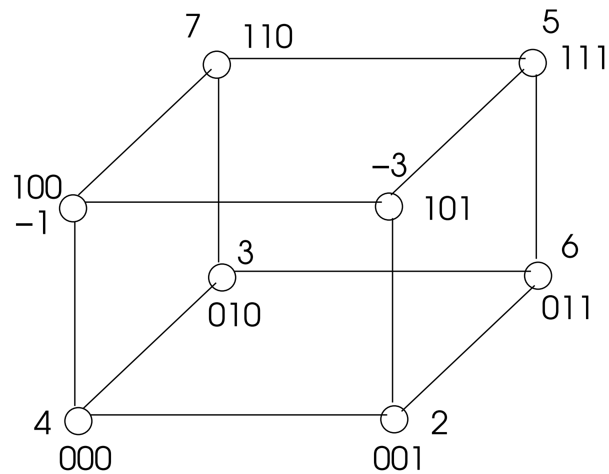
Saída: A soma $\sum_{i=0}^{n-1} A(i)$ aramazenada em P_0

Algoritmo

para $l = d-1$ **até** 0 **faça**
  se $0 \leq i \leq 2^l - 1$ **então**
    $A[i] := A[i] + A[i^{(l)}]$

Soma no Hipercubo

Exemplo: $n = 8$, então $d = 3$



	0	1	2	3	4	5	6	7
A	4	2	3	6	–1	–3	7	5

Soma no Hipercubo

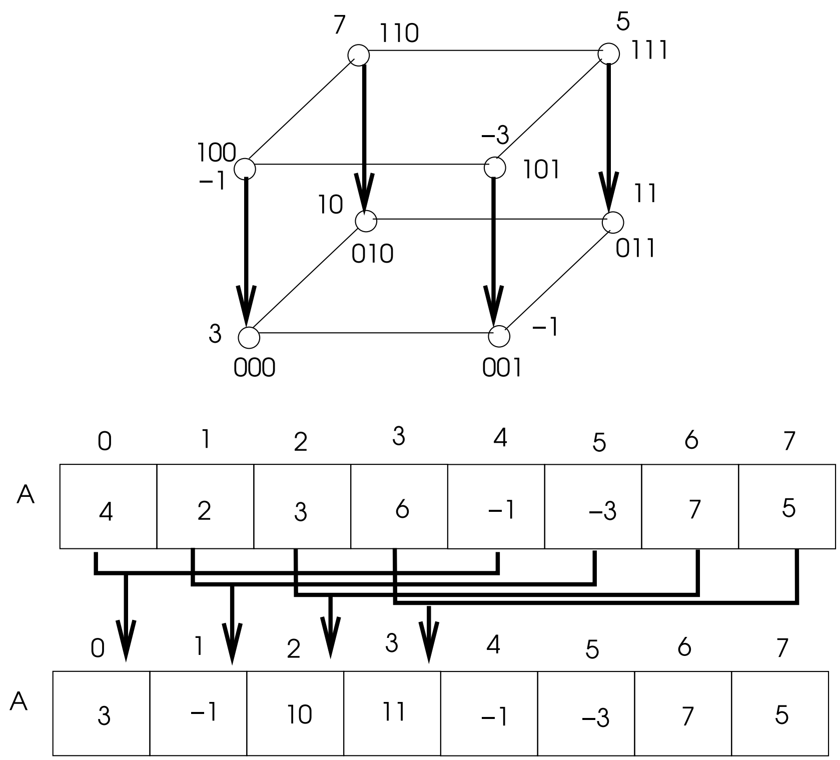
Dimensão do cubo $d = 3$

Para $l = d - 1 = 2$, executar a soma $A(i) = A(i) + A(i^{(l)})$, para $0 \leq i \leq 3$

$$A(0) = A(0) + A(0^{(2)}) = A(0) + A(4) = 4 + (-1) = 3$$
$$A(1) = A(1) + A(1^{(2)}) = A(1) + A(5) = 2 + (-3) = -1$$
$$A(2) = A(2) + A(2^{(2)}) = A(2) + A(6) = 3 + 7 = 10$$
$$A(3) = A(3) + A(3^{(2)}) = A(3) + A(7) = 6 + 5 = 11$$

Soma no Hipercubo

Exemplo: $n = 8$, então $d = 3$



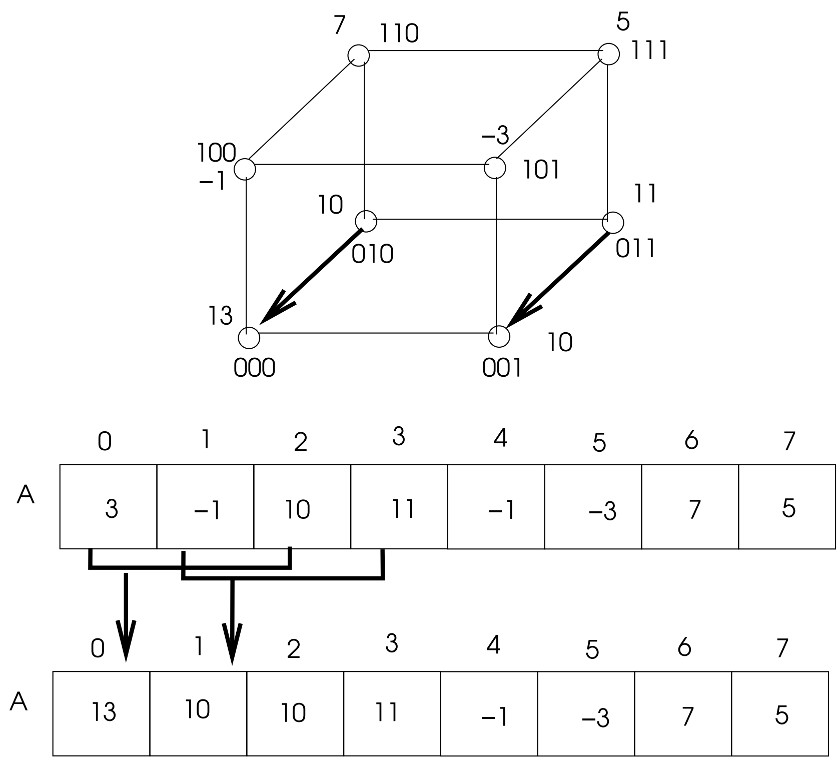
Soma no Hipercubo

Iteração

Para $l = 1$, executar a soma $A(i) = A(i) + A(i^{(l)})$, para $0 \leq i \leq 1$

$$A(0) = A(0) + A(0^{(1)}) = A(0) + A(2) = 3 + 10 = 13$$
$$A(1) = A(1) + A(1^{(1)}) = A(1) + A(3) = -1 + 11 = 10$$

Soma no Hipercubo



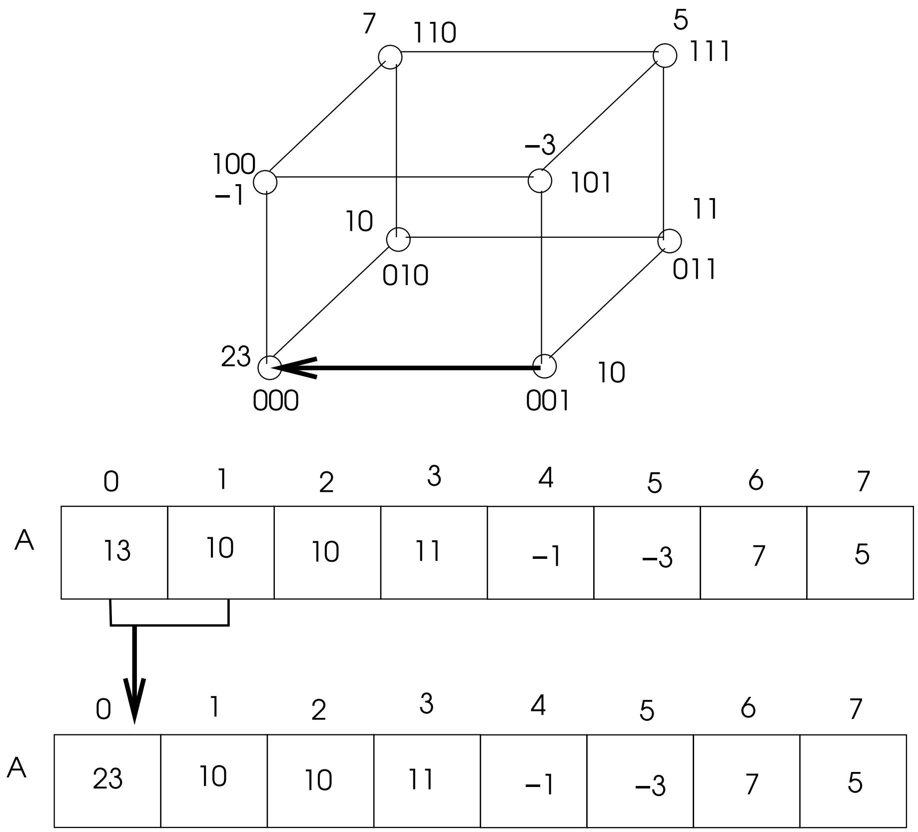
Soma no Hipercubo

Iteração

Para $l = 0$, executar a soma $A(i) = A(i) + A(i^{(l)})$, para $0 \leq i \leq 0$

$A(0) = A(0) + A(0^{(0)}) = A(0) + A(1) = 13 + 10 = 23$

Soma no Hipercubo



Broadcast de um processador no Hipercubo

Broadcast de um processador no Hipercubo

Considere o problema de broadcast de um item X contido no registrador $D[0]$ de P_0 para todos os processadores P_i de um hipercubo de p-processdores, onde $p = 2^d$

Broadcast de um processador no Hipercubo

P_0 envia uma cópia de X para P_1

P_0 e P_1 enviam uma cópia para P_2 e P_3, e assim por diante.

Broadcast de um processador no Hipercubo

$D[i]$ é enviado do processador Pi ao processador $P_i^{(l)}$ através do link existente entre os dois processadores.

No segundo subpasso, $P_i^{(l)}$ recebe a cópia e armazena no registrador D.

Broadcast de um processador no Hipercubo

Algoritmo para o processador P_i

para $l := 0$ **até** $d - 1$ **faça**
  se $0 \leq i \leq 2^l - 1$ **então**
    $D[i^{(l)}] = D[i]$

Exercícios

1) Dados dois vetores A e B de n elementos, escreva um algoritmo distribuído na topologia em anel para calcular o produto escalar em A e B. O produto escalar é dado por $A[0] \cdot B[0] + A[1] \cdot B[1] + \cdots + A[n-1] \cdot B[n-1]$.

2) Projete um algoritmo para computar a multiplicação de duas matrizes A e B de $n \times n$ em um hipercubo.

Propriedades de Algoritmos Distribuídos

Topologia: O algoritmo utiliza alguma topologia específica ou é genérico?

Capacidade de comunicação: o algoritmo precisa que exista bufferização na comunicação? Quanto?

Comunicação **direta** ou **indireta**?

Propriedades de Algoritmos Distribuídos

Preservação da ordem de envio das mensagens: o algoritmo exige que os canais sejam FIFO em relação as mensagens?

Distribuição de Controle: Todos os processos executam tarefas idênticas (simetria) ou existe algum processo que realiza alguma tarefa específica de maneira centralizada.

Propriedades de Algoritmos Distribuídos

Estados Globais: Em uma execução de um algoritmo distribuído um processo não tem conhecimento do estado global do sistema em um dado instante. Cada processo sabe apenas o seu estado local.

Terminação de um algoritmo distribuído: A execução de um algoritmo distribuído termina quando todos os processos terminaram localmente e não há mais nenhuma mensagem em trânsito (mensagens enviadas e ainda não recebidas).

Propriedades de Algoritmos Distribuídos

Algoritmo para cálculo de produto escalar

- ▷ Específico para o cálculo do produto escalar
- ▷ Não exige capacidade de comunicação
- ▷ Comunicação direta
- ▷ Não exige canais FIFO
- ▷ Existe simetria
- ▷ Processo inicial é o único que sabe da terminação global

Algoritmo para Propagação de Informação

Um processo possui uma informação inicialmente e deseja difundí-la para todos os processos.

Cada processo conhece apenas quais são os processos vizinhos a ele.

Algoritmo para Propagação de Informação

Estrutura de dados de cada processo:

▷ $NVizinhos$: número de vizinhos

▷ id: id do processo

▷ $informacao$: a informação a ser transmitida

▷ $IdVizinho$: vetor com a identificação dos processos vizinhos

Algoritmo para Propagação de Informação

Idéia do algoritmo:

Para o processo que possui a informação inicialmente:

- ▷ Envia informação, para todos os vizinhos, aguarda e;
- ▷ Recebe informação de todos os vizinhos

Algoritmo para Propagação de Informação

Idéia do algoritmo:

Para os demais processos:

- ▷ Recebe informação de um vizinho (inicial)
- ▷ Envia informação para todos os vizinhos (inclusive para o inicial)
- ▷ Recebe informação de todos os vizinhos, exceto do vizinho inicial

Algoritmo para Propagação de Informação

Para o processo que possui a informação inicialmente:

$msg := informacao$
para $i := 0$ **até** $NVizinhos - 1$ **faça**
  $envia(msg, IdVizinho[i])$

para $i := 0$ **até** $NVizinhos - 1$ **faça**
  $recebe(msg, id)$

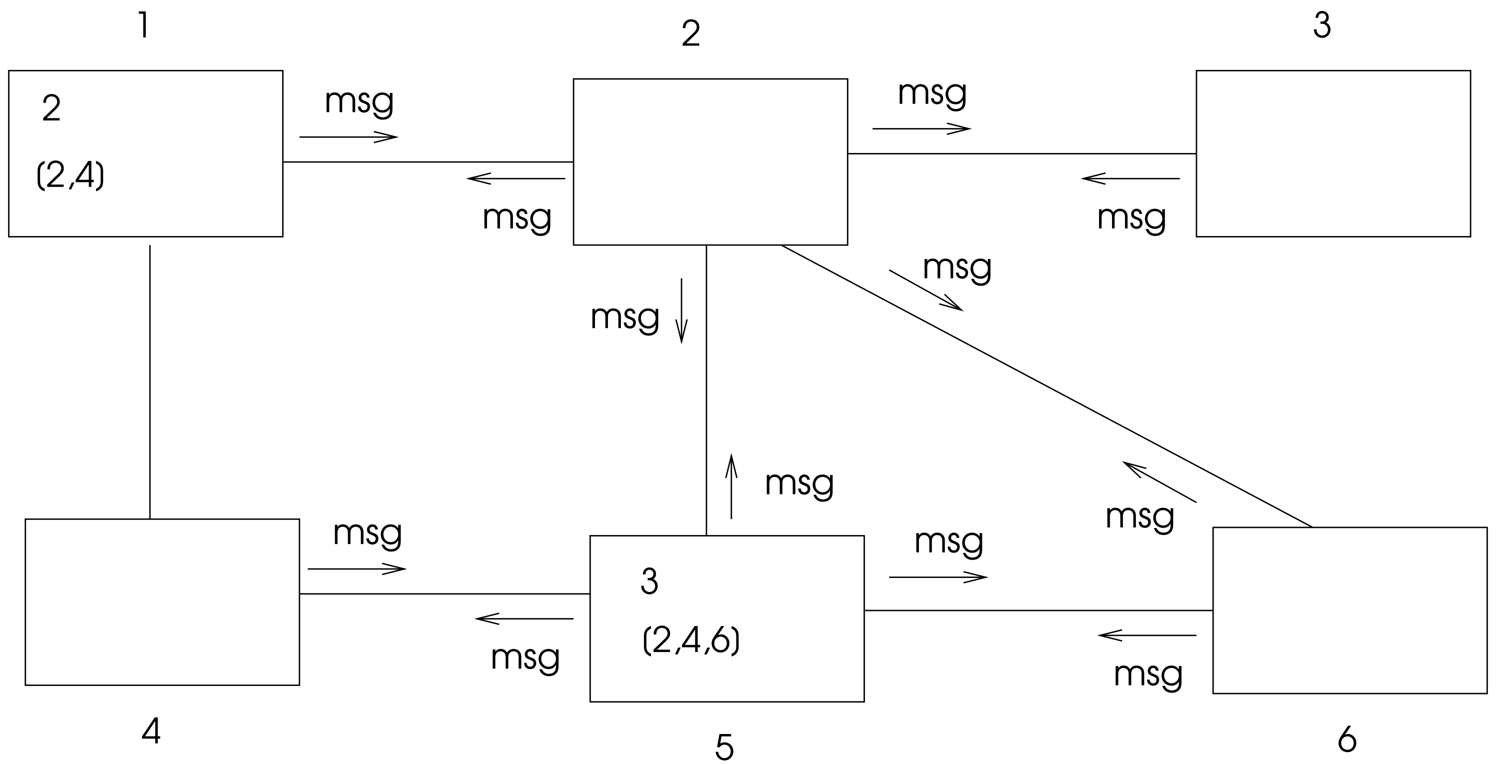
Algoritmo para Propagação de Informação

Para os demais processos:

$recebe(msg, id)$
$informacao := msg$
para $i := 0$ **até** $NVizinhos - 1$ **faça**
    $envia(msg, IdVizinho[i])$

para $i := 0$ **até** $NVizinhos - 2$ **faça**
    $recebe(msg, id)$

Processo com informação inicialmente: 1



Algoritmo para Propagação de Informação

Cada processo precisa enviar a informação para todos os vizinhos pois ele não sabe se algum destes vizinhos vai receber ou já recebeu a informação de outro processo.

Quando um processo enviar informação para seus vizinhos ele não sabe se a informação será a primeira recebida de algum destes vizinhos

Algoritmo para Propagação de Informação

Um processo que não possuir a informação inicialmente pode receber a informação pela primeira vez, vinda de qualquer um de seus vizinhos (não determinismo)

Um processo que não possui a informação inicialmente precisa enviá-la para todos os seus vizinhos, inclusive o vizinho inicial. Se não for feito assim, cada processo não saberá quantas mensagens deve esperar.

Algoritmo para Propagação de Informação

Quando um processo recebe as mensagens dos vizinhos não importa a ordem em que recebe e sim que ele receba de todos.

Este algoritmo também permite que vários processos possuam a informação inicialmente.

Complexidades

- de mensagens: $O(n^2)$
  - Cada processo envia 1 mensagem para cada vizinho. Em cada canal passam 2 mensagens, uma em cada direção. Número de mensagens é igual a $2 \cdot |E|$, onde $|E|$ é igual ao número de canais
- Tamanho das mensagens: Tamanho da informação
- Tempo local: $O(n)$
  - loop em $NVizinhos$

Complexidades

▷ Tempo assíncrono: $O(n)$

- Cadeia causal
- Processo inicialmente envia informação para seus vizinhos espontaneamente
- Um dos processo vizinho recebe a informação pela primeira vez, e a envia para seus vizinho
- Um dos processo vizinho recebe a informação pela primeira vez, e a envia para seus vizinho
- $\cdots$ e assim por diante

No pior caso a cadeia causal é o comprimento do maior caminho do grafo da rede.

Algoritmo para Propagação de Informação

Propriedades

- ▷ Topologia ponto a ponto qualquer
- ▷ Comunicação direta assimétrica
- ▷ Capacidade de comunicação
- ▷ Não exige canais FIFO
- ▷ Controle distribuído
- ▷ Nenhum processo tem conhecimento de terminação global

Algoritmo para Propagação de Informação com Realimentação

Algoritmo para Propagação de Informação com Realimentação

Para uma topologia ponto a ponto qualquer

Um processo possui uma informação e precisa difundí-la para todos os demais processos

O processo inicial precisa saber quando todos os processos já receberam a informação

Cada processo sobe apenas quais são os processos vizinhos a ele.

Algoritmo para Propagação de Informação com Realimentação

Estrutura de dados de cada processo:

$\triangleright$ $NVizinhos$: número de vizinhos

$\triangleright$ id: id do processo

$\triangleright$ $idVizInicial$: id do processo inicial

$\triangleright$ $informacao$: a informação a ser transmitida

$\triangleright$ $IdVizinho$: vetor com a identificação dos processos vizinhos

Algoritmo para Propagação de Informação com Realimentação

Idéia do algoritmo:

Para o processo que possui a informação inicialmente:

- ▷ Envia informação, para todos os vizinhos, aguarda e;
- ▷ Recebe informação de todos os vizinhos

Algoritmo para Propagação de Informação

Idéia do algoritmo:

Para os demais processos:

- ▷ Recebe informação de um vizinho (inicial)
- ▷ Envia informação para todos os vizinhos, exceto para o inicial
- ▷ Recebe informação de todos os vizinhos, exceto do vizinho inicial
- ▷ Envia informação para o vizinho inicial

Algoritmo para Propagação de Informação

Para o processo que possui a informação inicialmente:

$msg := informacao$
para $i := 0$ **até** $NVizinhos - 1$ **faça**
    $envia(msg, IdVizinho[i])$

para $i := 0$ **até** $NVizinhos - 1$ **faça**
    $recebe(msg, id)$

Algoritmo para Propagação de Informação

Para os demais processos:

$recebe(msg, idVizInicial)$
$informacao := msg$
para $i := 0$ **até** $NVizinhos - 1$ **faça**
  se $IdVizinho[i] \neq idVizInicial$ **então**
    $envia(msg, IdVizinho[i])$

para $i := 0$ **até** $NVizinhos - 2$ **faça**
  $recebe(msg, id)$

$envia(msg, idVizInicial)$

Complexidades

- ▷ de mensagens: $O(n^2)$
- ▷ Tamanho das mensagens: Tamanho da informação
- ▷ Tempo local: $O(n)$
- ▷ Tempo assíncrono: $O(n)$

Propriedades

▷ Topologia ponto a ponto qualquer

▷ Comunicação direta assíncrona

▷ Capacidade de comunicação: 0 (quanto menor melhor o algoritmo)

▷ Não exige que canais sejam FIFO

▷ Distribuição de controle

▷ Terminação: apenas o processo que possui a informação inicialmente tem o conhecimento da terminação global.

Fim