Programação Paralela

Objetivo: *Apresentar conceitos básico de programação paralela com MPI.*

Apresentação

- Introdução;
- Arquiteturas paralelas;
- Análises de algoritmos;
- Comunicação em programas paralelos;
- Ambiente de desenvolvimento;
- Execução de programas paralelos: mpiexec
- MPI: conceitos básicos;
- Comunicação coletiva;
- Comunicação ponto a ponto;
- Paralelização de programas;
- Exemplos completos;
- Trilinos: introdução;

Bibliografia

- MPI: A Message Passing Interface Standard – http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
- Introduction to High Performance Scientific Computing, Victor Eijkhout – https://bitbucket.org/VictorEijkhout/hpc-book-and-course/src
- Parallel Computing for Science and Engineering, Victor Eijkhout – https://bitbucket.org/VictorEijkhout/parallel-computing-book/src
- Parallel and Distributed Computation, Dimitri P. Bertsekas, 1988;
- Manuais Trilinos – https://trilinos.org/about/documentation/
- Tutorial Python – https://docs.python.org/3/tutorial/index.html

Motivação

Porque computação paralela?

- Respostas em tempos de engenharia;
- Incerteza;
- Otimização;
- Múltiplas Físicas;
- Problemas 3D, transientes;
- Milhões, bilhões de células;

Alternativas

- Algoritmo melhor (programador mais inteligente!);
- Compilador melhor;
- Computador mais rápido;
- Usar vários processadores!

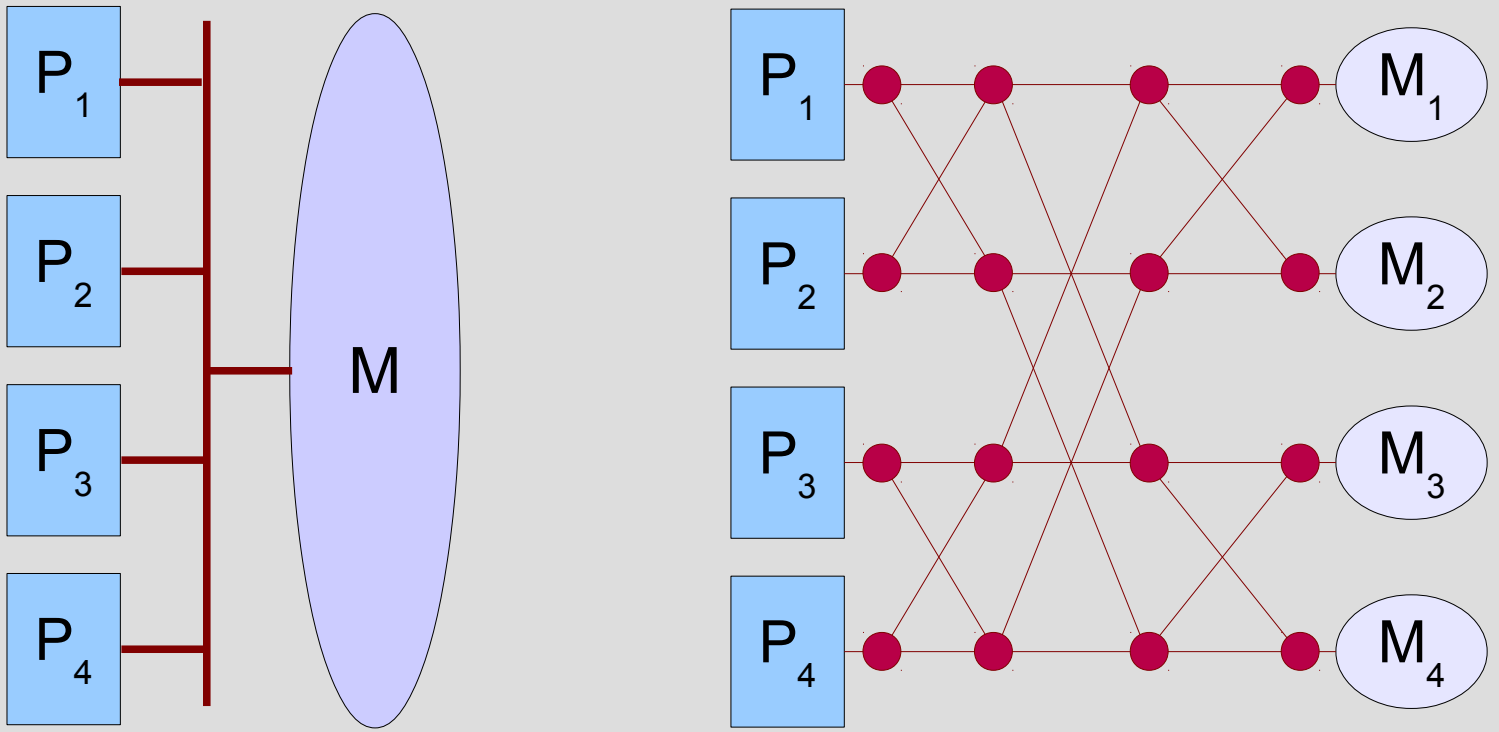
Porque Estudar Computação Paralela

- Fatores que não existem na computação serial:
  - Decomposição de problemas e alocação de tarefas;
  - Comunicação entre processadores;
  - Sincronização;
- Aprender um padrão “universal” para escrita de programas portáteis: MPI;

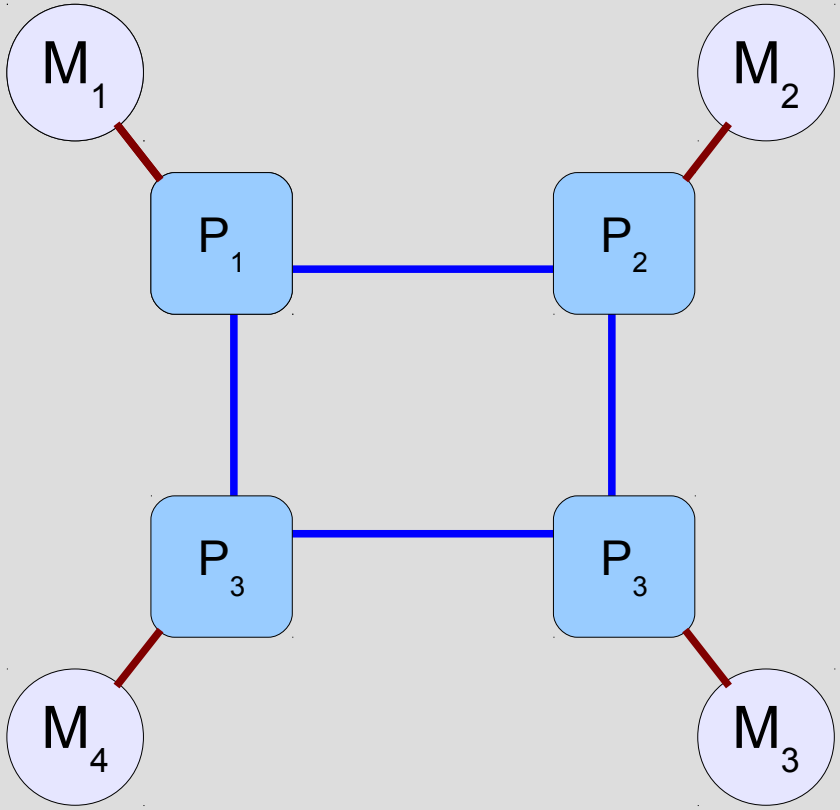
Classificação de Computadores Paralelos

- Tipo e número de processadores:
  - Massivamente paralelos x “Coarse Grained”
- Mecanismo global de controle:
  - Ausente ou presente
  - SIMD x MIMD
- Operação síncrona ou assíncrona
- Comunicação entre processadores
  - Memória compartilhada x distribuída (troca de mensagens)

Memória Compartilhada



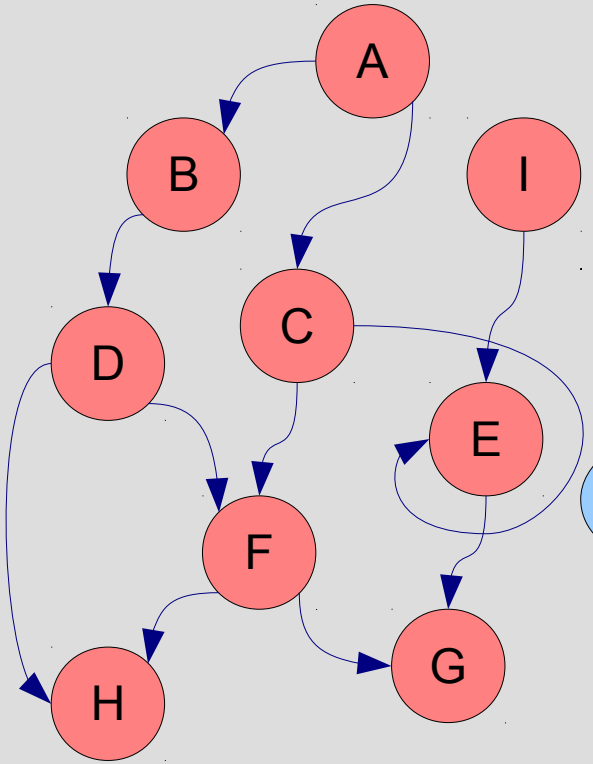
Memória Distribuída



Modelo para Computação Paralela

- Hipóteses Básicas:
  a) Cada processador é um computador. sequencial convencional: laços, controle, etc.
  b) Mecanismo para troca de informação.
  c) Todas as operações levam 1 unidade de tempo.

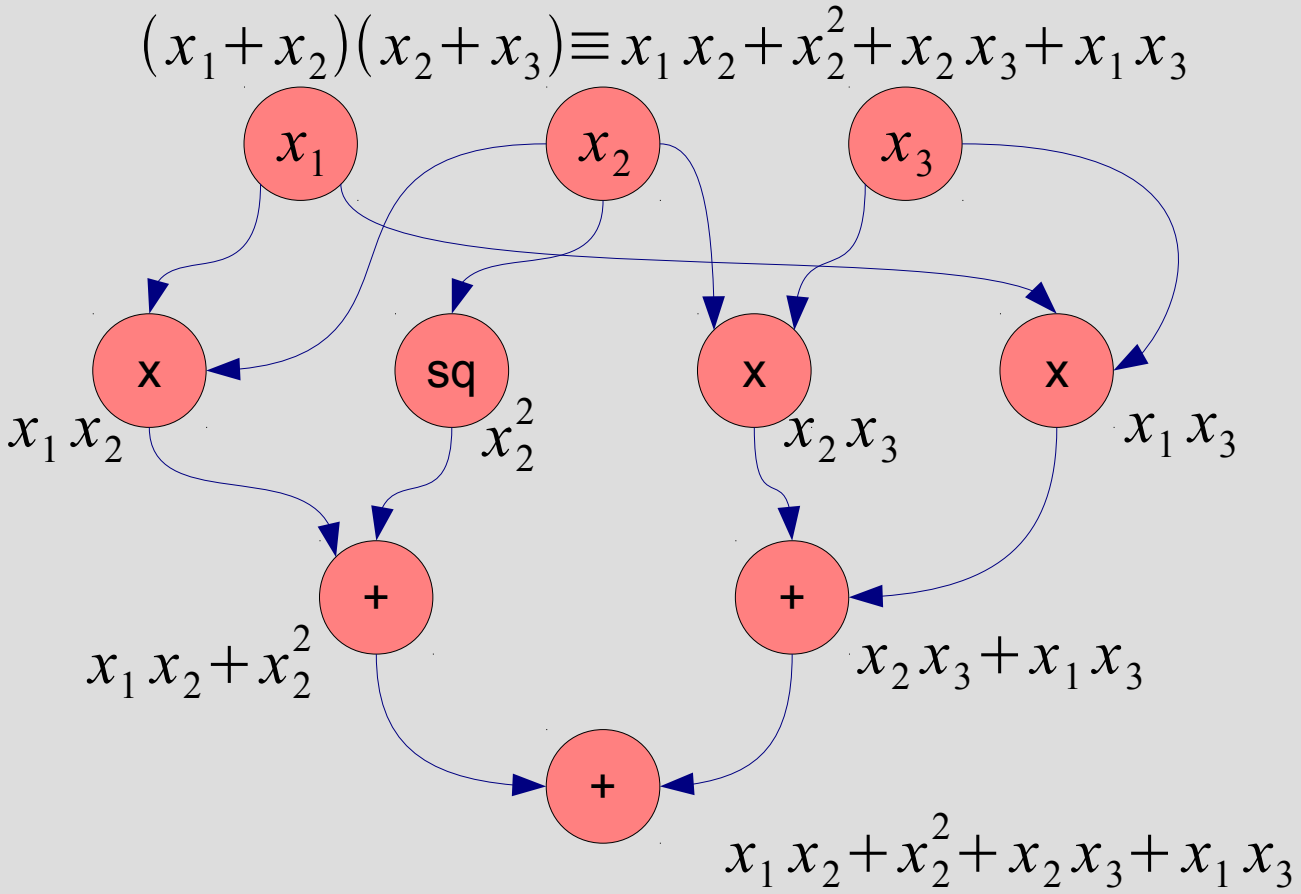
DAG – Directed Acyclic Graph



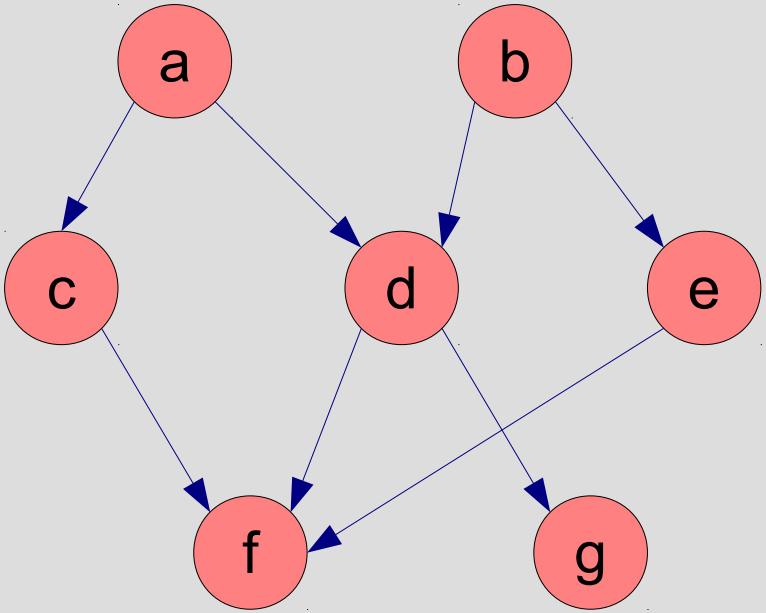
N = {A, B, ..., I}
A = { (A,B), (A,C), ...}
G = {N, A}

Idéia Fundamental:
Nós – Operações
Arcos – Dependência

Exemplo


$(x_1+x_2)(x_2+x_3)\equiv x_1x_2+x_2^2+x_2x_3+x_1x_3$
x_1
x_2
x_3
x
sq
x
x
x_1x_2
x_2^2
x_2x_3
x_1x_3
+
+
$x_1x_2+x_2^2$
$x_2x_3+x_1x_3$
+
$x_1x_2+x_2^2+x_2x_3+x_1x_3$

Terminologia



Predecessor:
i é predecessor de j se $(i, j) \in A$

Caminho Positivo:
$\{(i_0, \ldots, i_K) | (i_k, i_k + 1) \in A ; k = 0, \ldots, K - 1\}$

$x_i = f_i\ (\{x_j \mid j \text{ é predecessor de } i\})$

- Grau de entrada
- Grau de saída
- N_0
- Caminho Positivo
- Profundidade do DAG

Aspectos Importantes

- DAG *não* representa todos os aspectos da operação de um algoritmo paralelo
- Representa:
  - Operações necessárias
  - Precedência das operações
- Não representa:
  - Aonde? (em qual processador)
  - Quando? (em que tempo será executado)

Agendamento

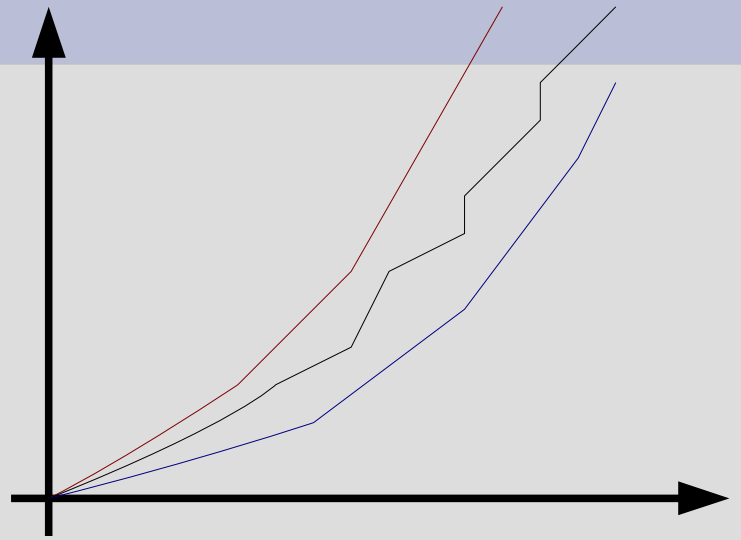
- Imaginando p processadores idênticos
- Para cada nó em N_0 vamos atribuir um P_i
- t_i é o tempo no qual a operação será executada
- $t_i = 0$ para nós de entrada
- Conseqüências:

$$\begin{cases} \text{Se } i \neq j, t_i = t_j \Rightarrow P_i \neq P_j \\ (i,j) \in A \Rightarrow t_j \geq t_i + 1 \end{cases}$$

  - Agendamento: $\{(i, P_i, t_i) | i \in N_0\}$

Medidas de Complexidade

$$\text{Supondo} \begin{cases} D \in \mathbb{R} \\ f, g : D \to \mathbb{R} \\ c \in \mathbb{R} \mid c > 0 \\ x_0 \in \mathbb{R} \end{cases}$$

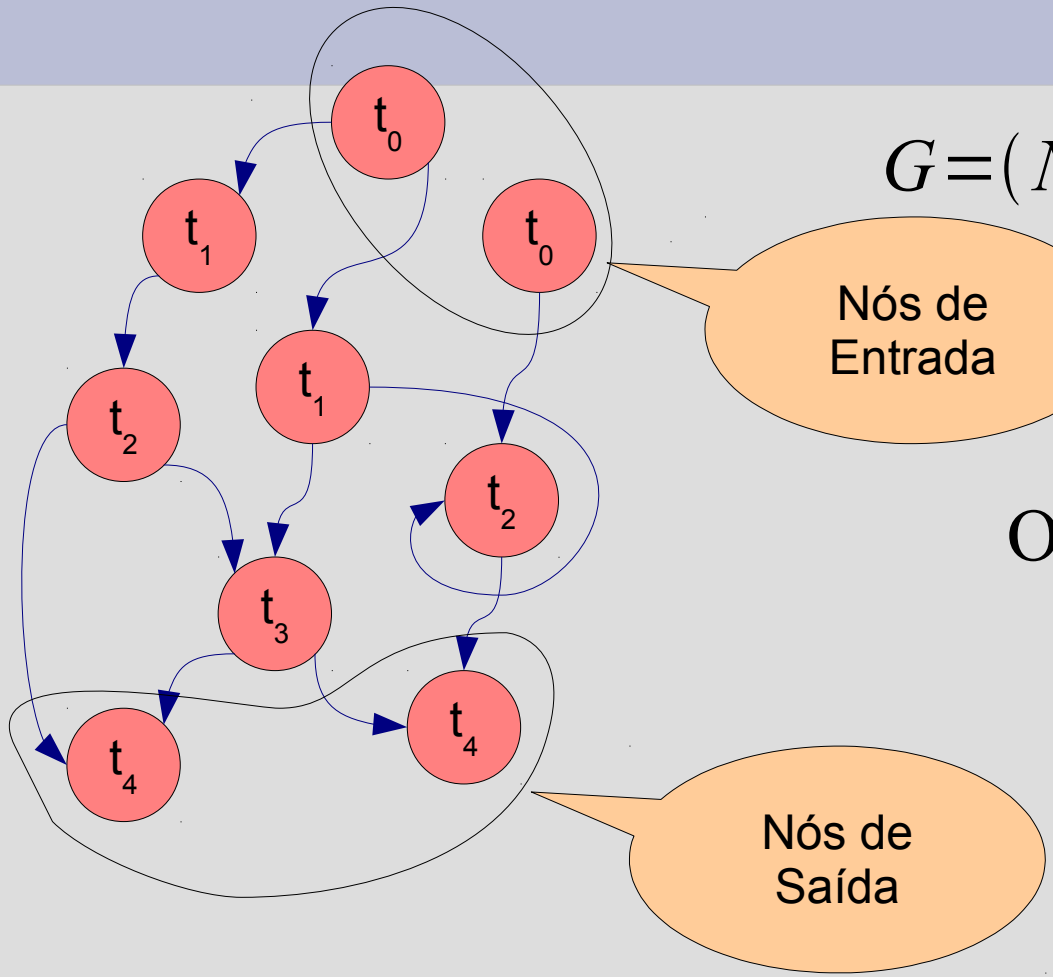


$$\text{Se } \forall x \geq x_0, |f(x)| \leq c\, g(x), \text{ então } f(x) = \mathrm{O}(g(x))$$

$$\text{Se } \forall x \geq x_0, |f(x)| \geq c\, g(x), \text{ então } f(x) = \Omega(g(x))$$

$$\text{Se } f(x) = \mathrm{O}(g(x)) \text{ e } f(x) = \Omega(g(x)), \text{ então } f(x) = \Theta(g(x))$$

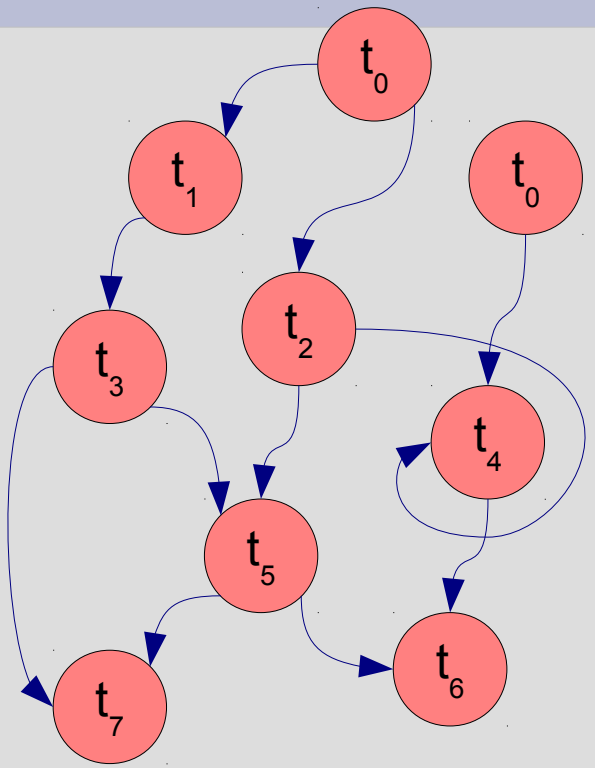
Complexidade Temporal



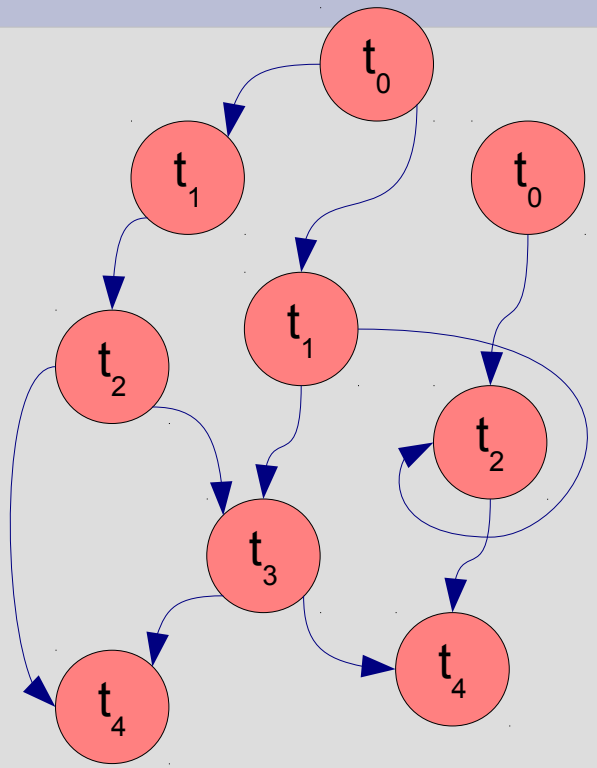
$$G=(N,A);\{(i,P_i,t_i)|i\in N_0\}$$

O tempo gasto é $\max_{i\in N} t_i$

Agendamentos Alternativos



$p=1;\max(t_i)=7$



$p=2;\max(t_i)=4$

Tempo de execução

- Considerando todos os possíveis schedules com *p* processadores, definimos:

$$T_p = \min\left(\max_{i \in N} t_i\right)$$

- Este é o menor tempo de execução possível com *p* processadores.
- Considerando uma disponibilidade infinita de processadores:

$$T_\infty = \min_{p \geq 1} T_p$$

Observações

$T_p \geq 0$

T_p não aumenta com p

$\exists\, p^* | T_p = T_\infty, p \geq p^*$, já que p é inteiro

T_∞ é a complexidade temporal do algoritmo

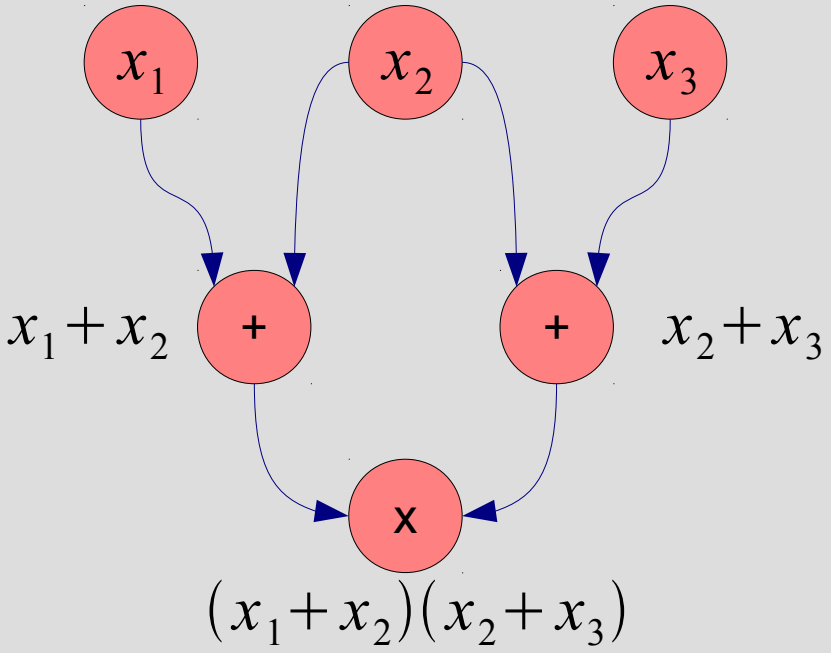
T_1 é o tempo para execução com 1 processador

$T_1 = |N_0|$

$T_\infty = D$

DAG Ótimo

- Não é nada trivial!



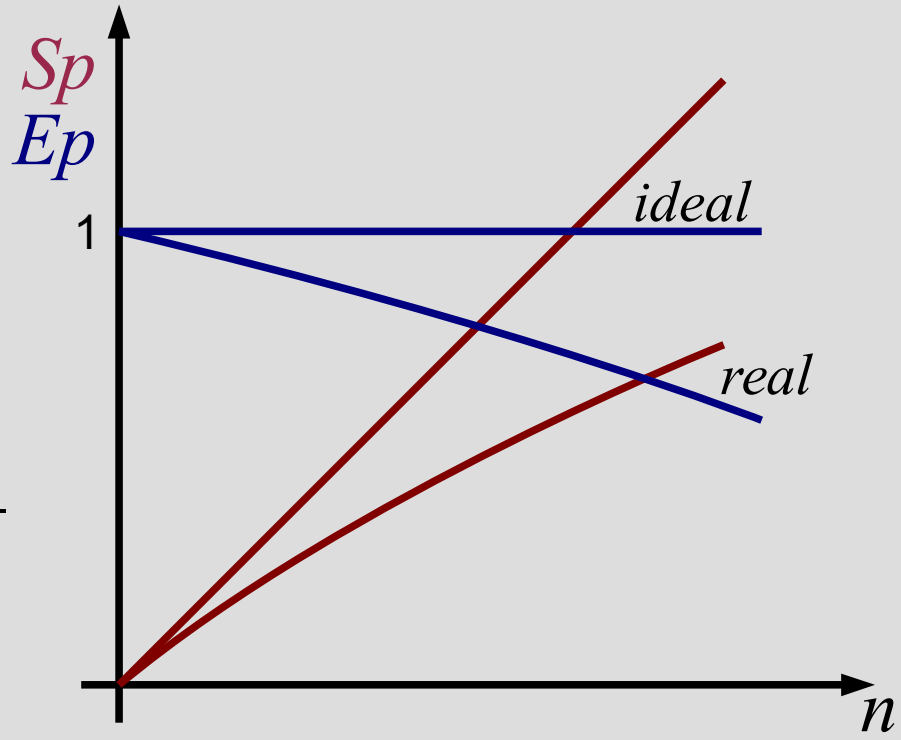
Definimos T_p^* como o menor T_p entre todos os DAG (algorimos) possíveis para um problema.

Speed-Up e Eficiência

T^* é definido como o tempo ótimo serial

$$Sp(n) = \frac{T^*(n)}{T_p}$$

$$Ep(n) = \frac{Sp(n)}{p} = \frac{T^*(n)}{pT_p(n)}$$



Lei de Amdahl

- A maior parte dos problemas tem seções que são seqüenciais;
- Supondo que *f* seja a porcentagem do tempo de execução que é estritamente seqüencial:

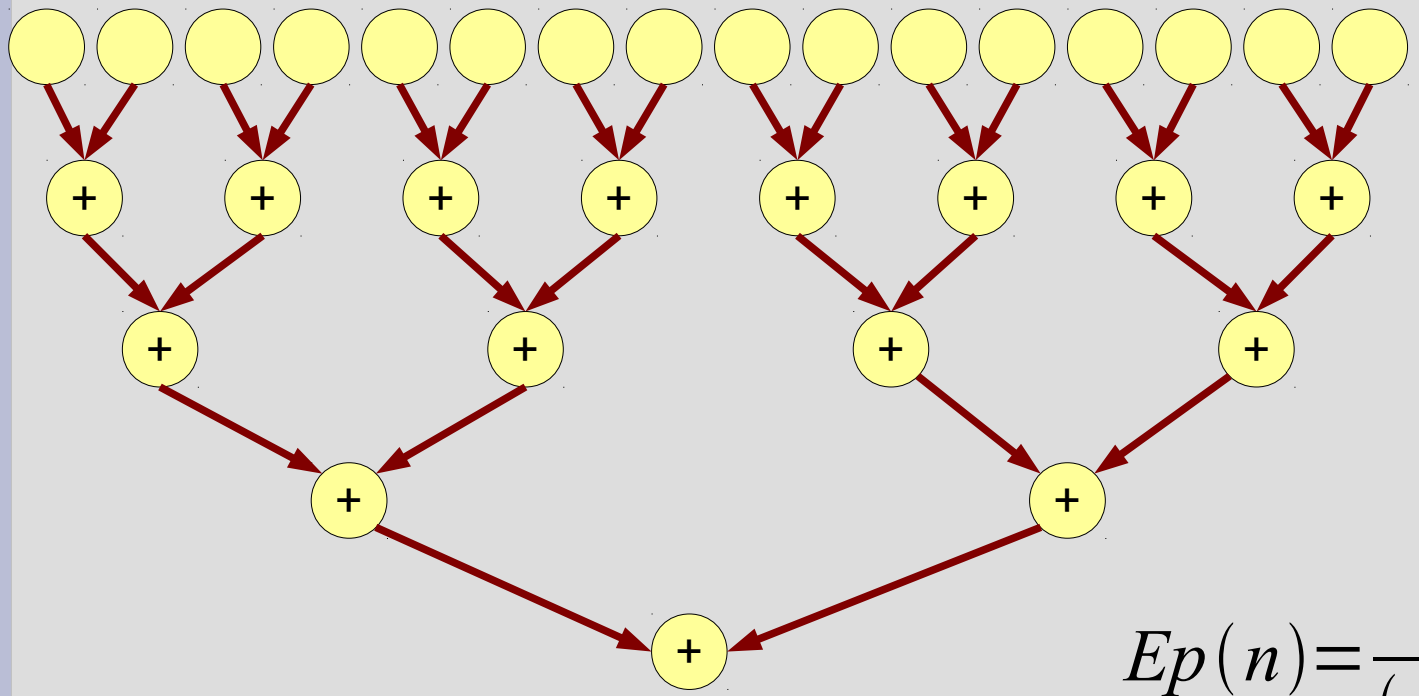
$$T^*(n) = (f + (1-f))T^*(n)$$

$$T_\infty(n) = f\,T^*(n) \text{ (paralelização perfeita!)}$$

$$Sp(n) = \frac{T^*(n)}{f\,T^*(n)} = \frac{1}{f}$$

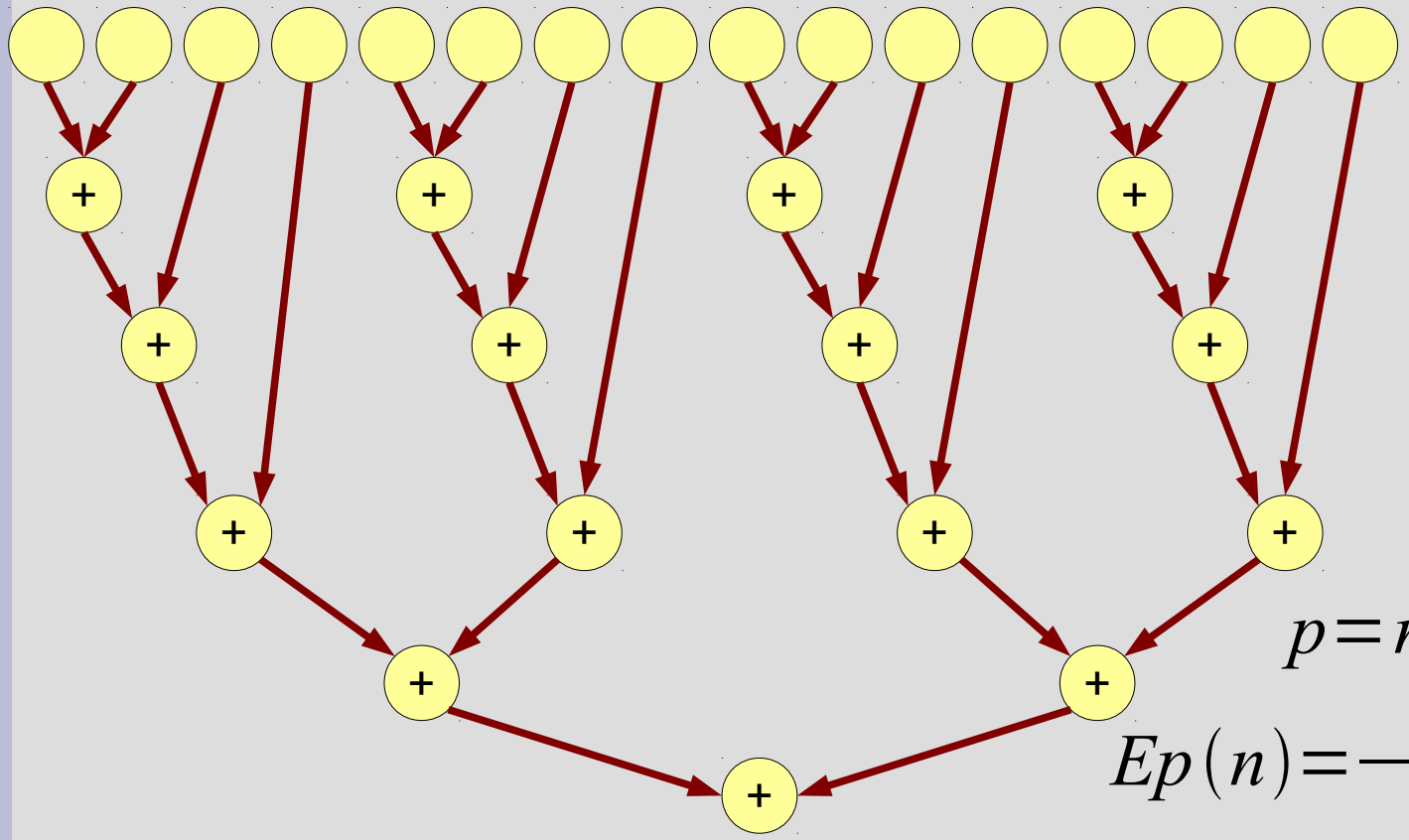
f(%)	Sp(n)
50,0	2
25,0	4
10,0	10
5,0	20
2,5	40
1,0	100

Adição de n Números



$$Ep(n) = \frac{n-1}{(n/2)\log(n)}$$

Adição de n Números

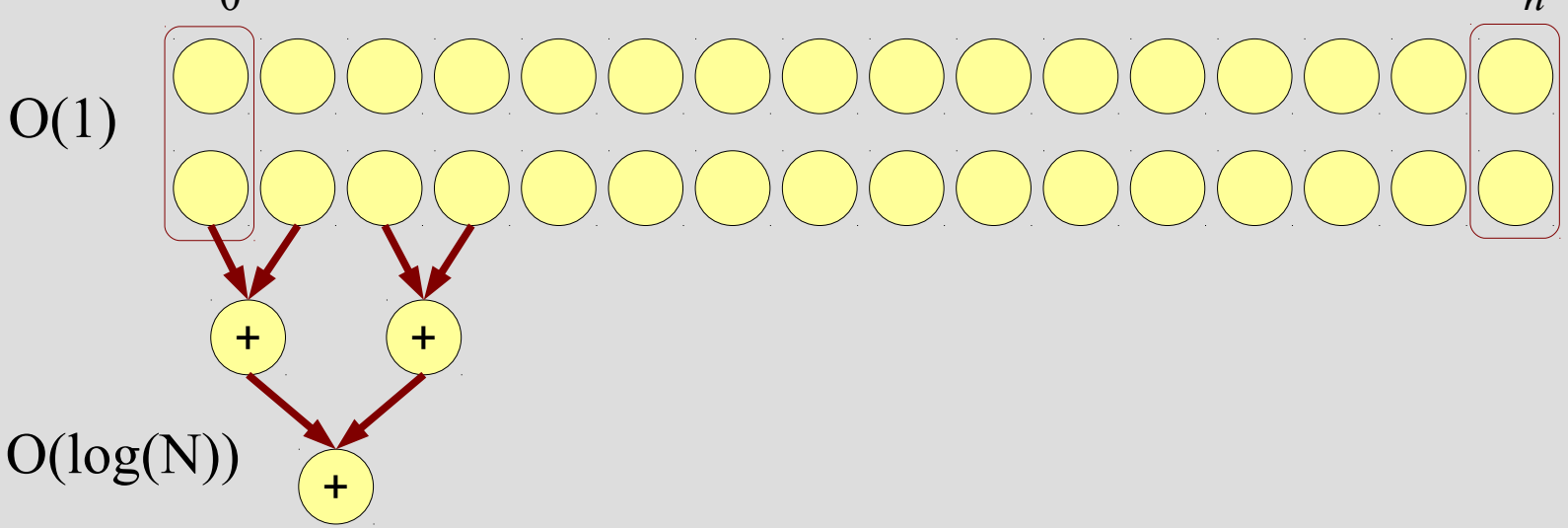


$$p = n/\log(n)$$

$$Ep(n) = \frac{n-1}{\frac{n}{\log(n)} 2\log(n)}$$

Produto Interno

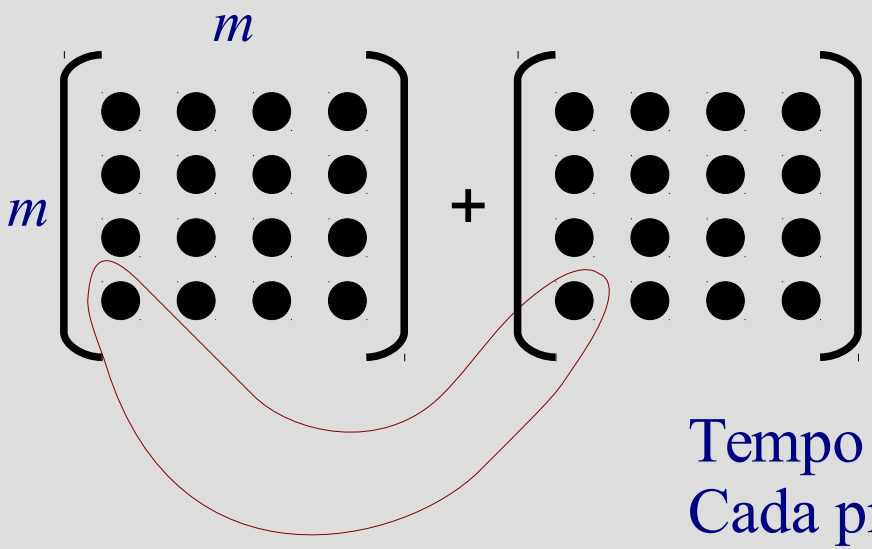
$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \, y_i$$



Adição de 2 Matrizes

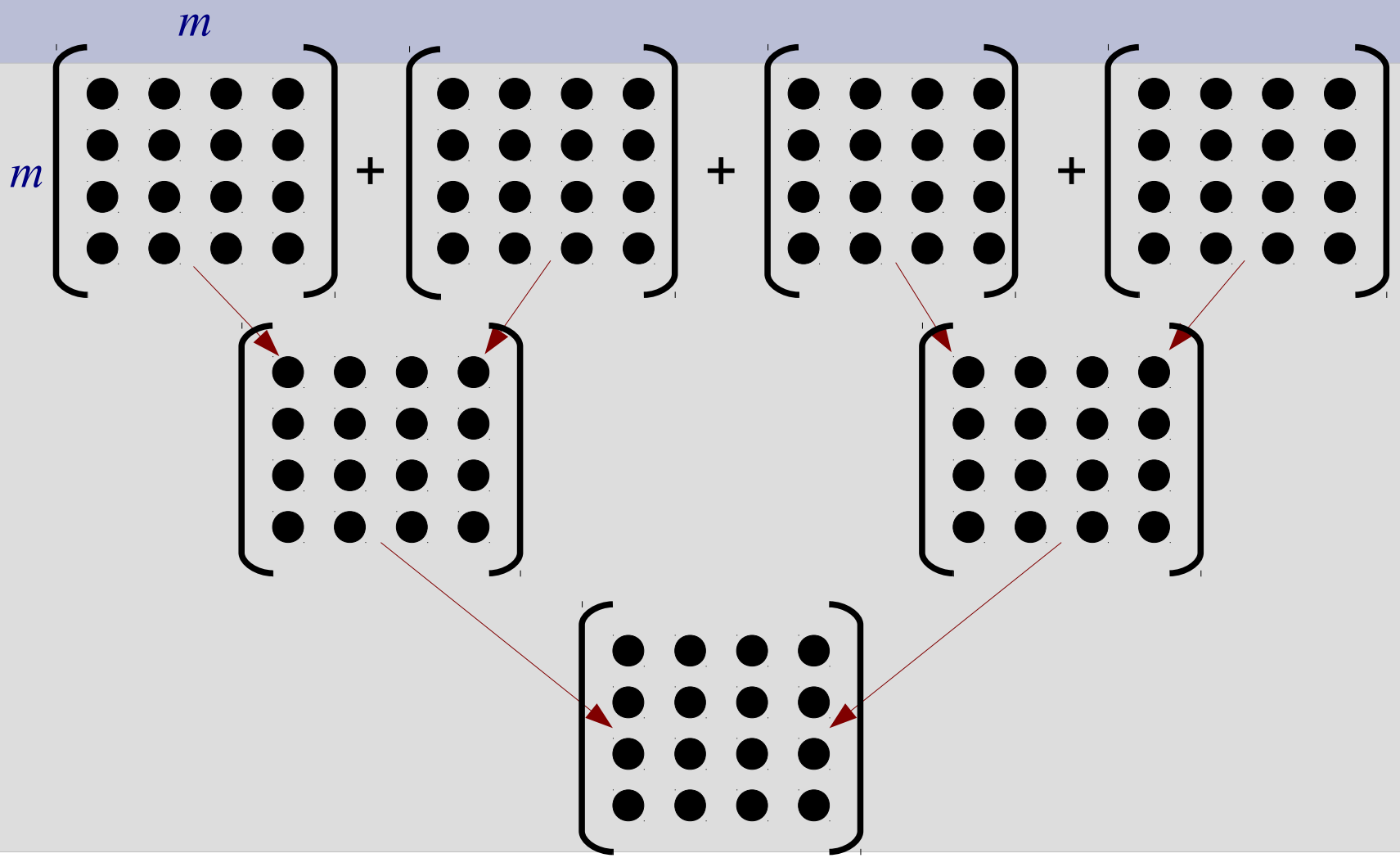
$$C_{m \times m} = A_{m \times m} + B_{m \times m}$$

$$c_{ij} = a_{ij} + b_{ij}$$



Tempo 1 com m^2 processadores!
Cada processador computa 1 soma!

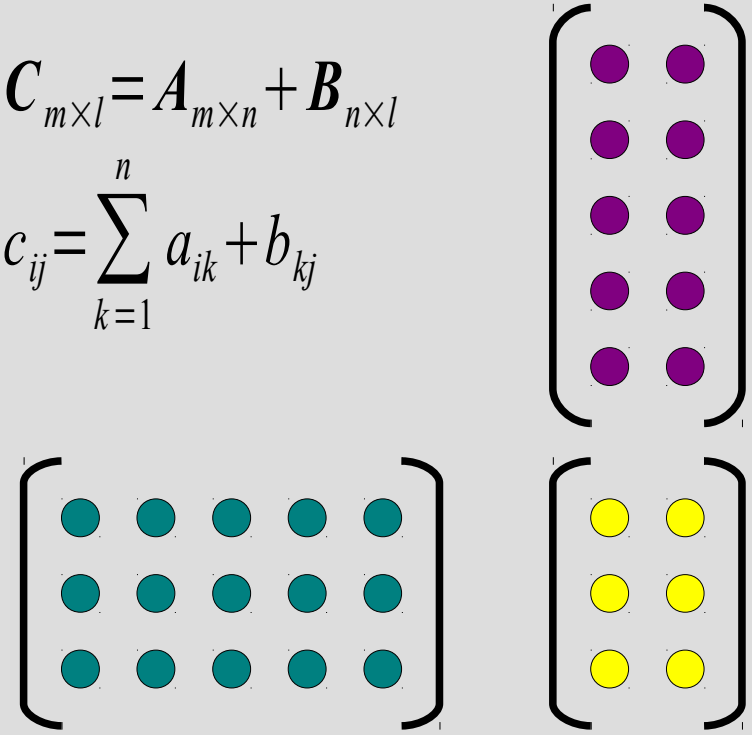
Adição de N Matrizes



Multiplicação de Matrizes

$$C_{m \times l} = A_{m \times n} + B_{n \times l}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} + b_{kj}$$



- *ml* produtos internos de comprimento *n* que podem ser calculados simultaneamente com *ml* processadores: tempo *n*
- com *ml(n/2)* processadores o tempo é O(log(*n*))

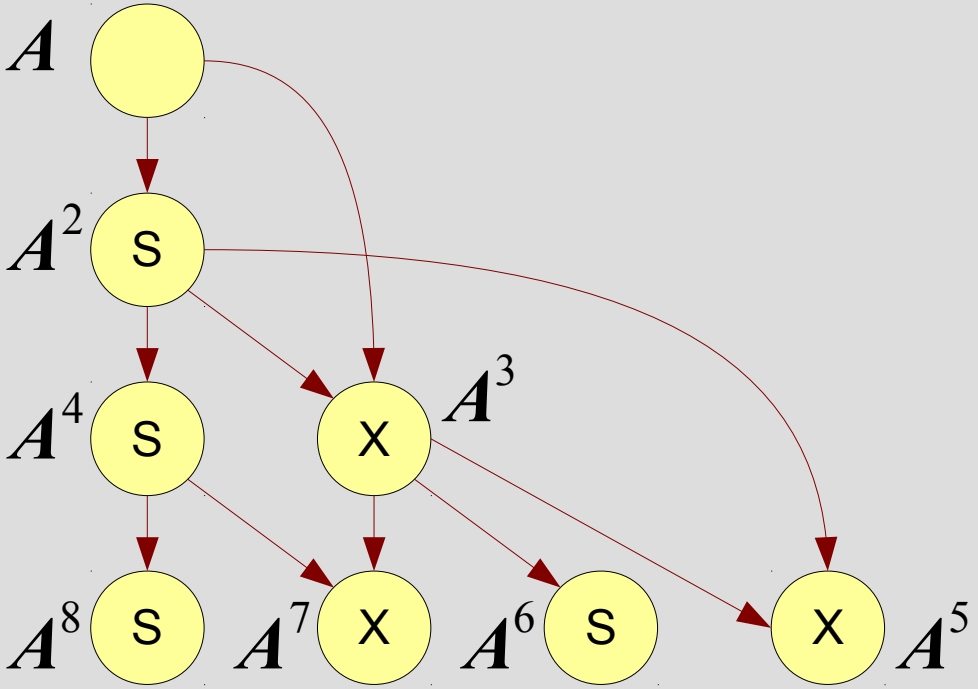
Potências de Matrizes

Dada $A_{n \times n}$

Calcular $A^k = \overbrace{A\,A \ldots A}^{k \text{ vezes}}$

Se $k = 2^l$

$A^k = \underbrace{((A^2)^2 \cdots)^2}_{l = \log(k) \text{ vezes}}$



No passo i, calculamos $A^{2^{i-1}+1}, \ldots, A^{2^i}$

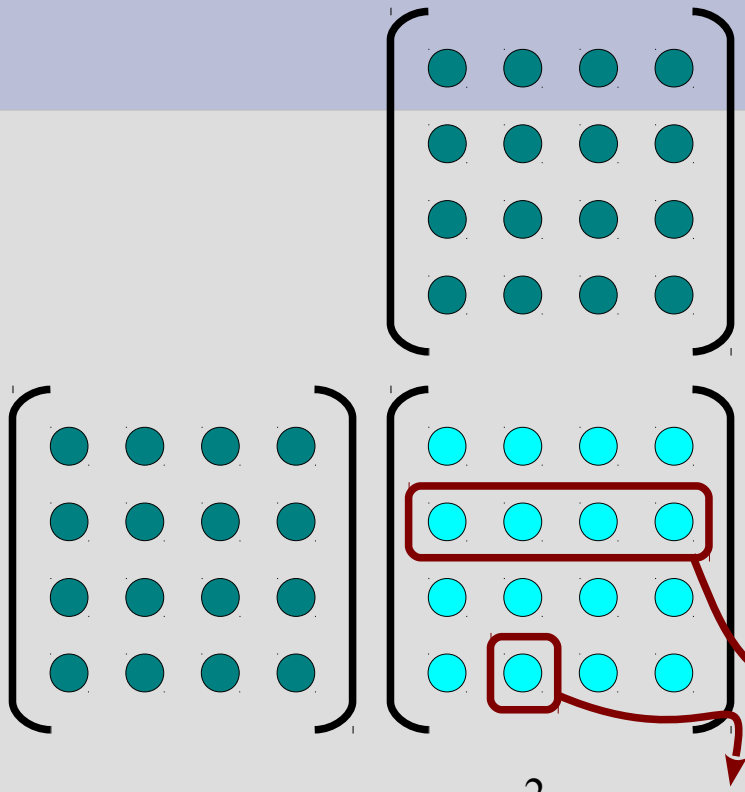
Observações Gerais

- Claramente, mostramos o algoritmo mais rápido possível (máximo número de processadores)
- Muitas vezes eficiência muito baixa, eg, multiplicação de matrizes:

$$E_p = \frac{\mathrm{O}(n^3)}{n^3 \mathrm{O}(\log n)}$$

- Podemos melhorar a eficiência usando menos processadores (o que é normal para problemas de grande porte!)

Eficiência



Tempo $2\log n$ com $\frac{n^3}{\log n}$ procs.

$$E_p = \frac{n^3}{\frac{n^3}{\log n} 2\log n} \equiv \mathrm{O}(1)$$

Tempo n^2 com n processadores

$$E_p = \frac{n^3}{n\,n^2} \equiv \mathrm{O}(1)$$

Tempo n com n^2 processadores

$$E_p = \frac{n^3}{n^2\,n} \equiv \mathrm{O}(1)$$

Métodos Iterativos

$$\begin{cases} x(t+1) = f(x(t)); \quad t = 0, 1, \ldots, \\ x \in \mathbb{R}^n \\ f : \mathbb{R}^n \to \mathbb{R}^n \end{cases}$$

$$x_i(t+1) = f_i(x_1(t), x_2(t), \ldots, x_n(t)); \quad i = 1, \ldots, n$$

Se

$f(x) = \boldsymbol{A}x + b$

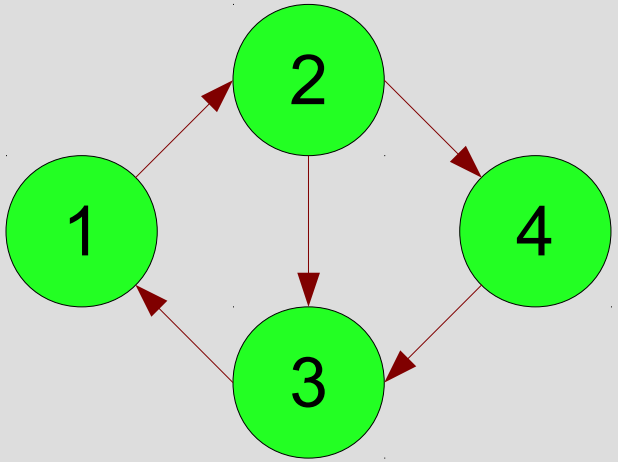
$\boldsymbol{A}$ é uma matriz $n \times n$ constante

$b \in \mathbb{R}^n$

Então a iteração é linear

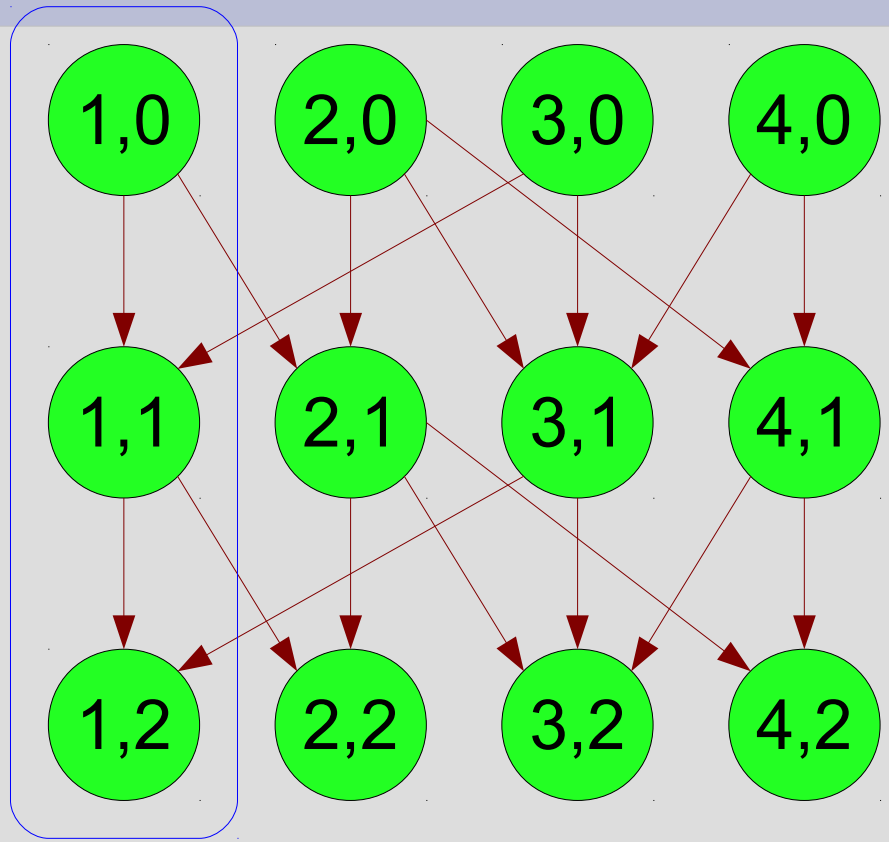
Iteração de Jacobi

$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_2(t+1) = f_2(x_1(t), x_2(t))$$
$$x_3(t+1) = f_3(x_2(t), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t), x_4(t))$$



O grafo de dependência
não é um DAG!

DAG Iteração de Jacobi



$t_4 = 1$

Paralelização em Blocos

$$\mathbb{R}^n = \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \times \mathbb{R}^{n_p}$$

$$\sum_{j=1}^{p} n_j = n$$

$$\vec{x} \in \mathbb{R}^n \equiv \vec{x} = (\vec{x}_1, \cdots, \vec{x}_j, \cdots, \vec{x}_p); \quad \vec{x}_j \in \mathbb{R}^{n_j}$$

$\vec{x}_j$ é um bloco de $\vec{x}$

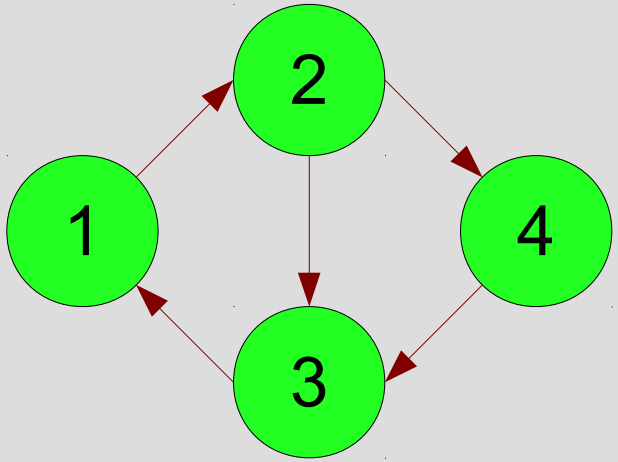
Iteração por blocos:

$$\vec{x}_j(t+1) = f_j(\vec{x}(t)); \quad j = 1, \ldots, p$$

Iteração de Gauss-Seidel

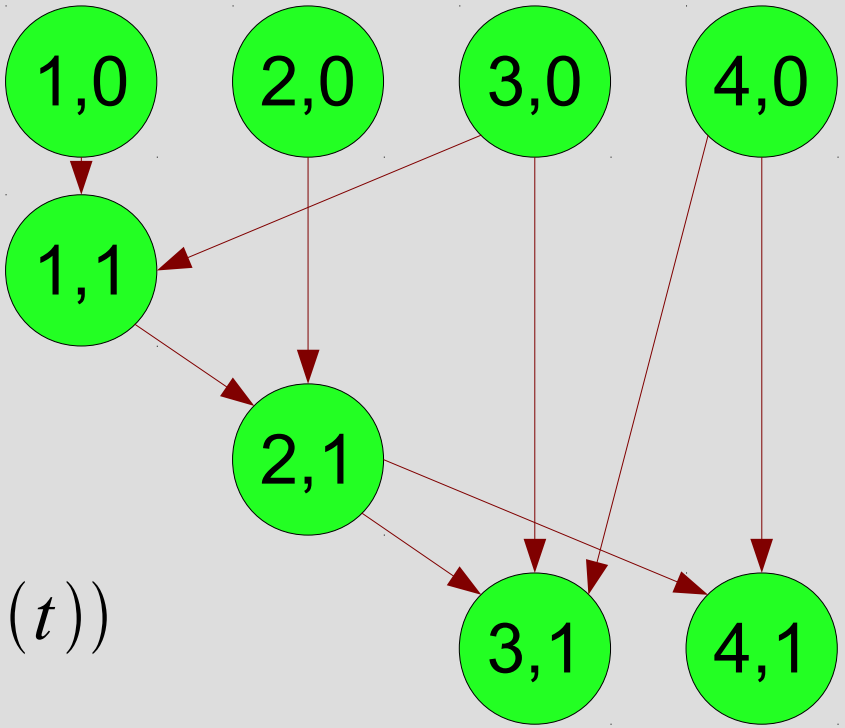
$$x_i(t+1)=$$
$$f_i(x_1(t+1), x_2(t+1), \ldots, x_{i-1}(t+1), x_i(t), \ldots, x_n(t))$$
$$i=1,\ldots,n$$

$$x_1(t+1)=f_1(x_1(t), x_3(t))$$
$$x_2(t+1)=f_2(x_1(t+1), x_2(t))$$
$$x_3(t+1)=f_3(x_2(t+1), x_3(t), x_4(t))$$
$$x_4(t+1)=f_4(x_2(t+1), x_4(t))$$

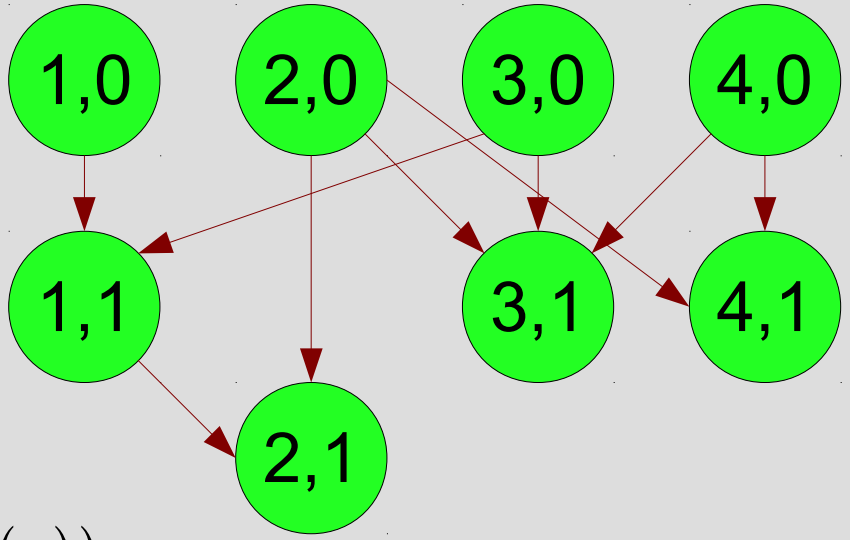


DAG Iteração de GS

$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_2(t+1) = f_2(x_1(t+1), x_2(t))$$
$$x_3(t+1) = f_3(x_2(t+1), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t+1), x_4(t))$$



Outra Ordem de Atualização



$$x_1(t+1) = f_1(x_1(t), x_3(t))$$
$$x_3(t+1) = f_3(x_2(t), x_3(t), x_4(t))$$
$$x_4(t+1) = f_4(x_2(t), x_4(t))$$
$$x_2(t+1) = f_2(x_1(t+1), x_2(t))$$

Problema!

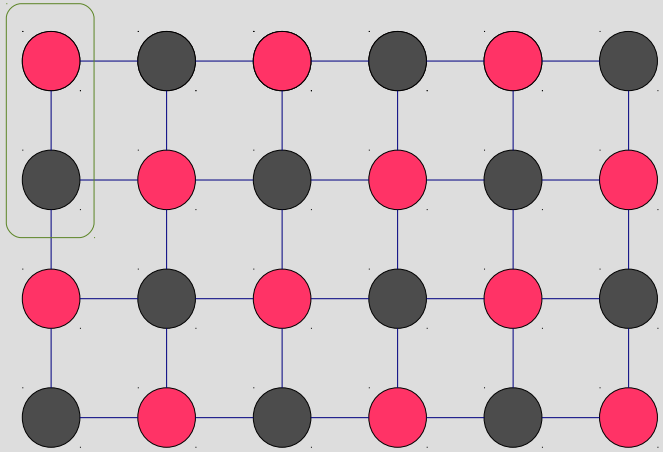
- Não é sempre possível paralelizar a iteração de Gauss-Seidel!
- Se cada variável depende de todos as outras (o grafo de dependência é “cheio”), então não há paralelismo;
- Nas iterações de métodos numéricos, (PDEs) no entanto, o grafo de dependência é em geral “esparso”, e é possível encontrar paralelismo.

Colorindo Grafos

- “Colorir” um grafo significa atribuir um número (uma “cor”) a cada nó do grafo, de forma que nós vizinhos tenham cores diferentes.
- Podemos mostrar que se um grafo de dependência simétrico pode ser colorido com K cores, então uma iteração de GS pode ser feita em K passos.
- A idéia é que nós com a mesma cor podem ser atualizados simultaneamente.

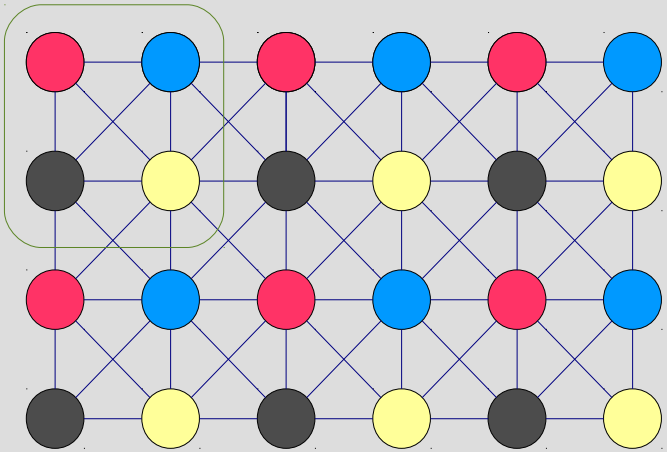
Exemplos – Diferenças finitas 2D

$$\nabla^2 u_{ij} \approx \frac{u_{i+1,\,j} + u_{i-1,\,j} + u_{i,\,j+1} + u_{i,\,j-1} - 4\,u_{i,\,j}}{\Delta h^2}$$

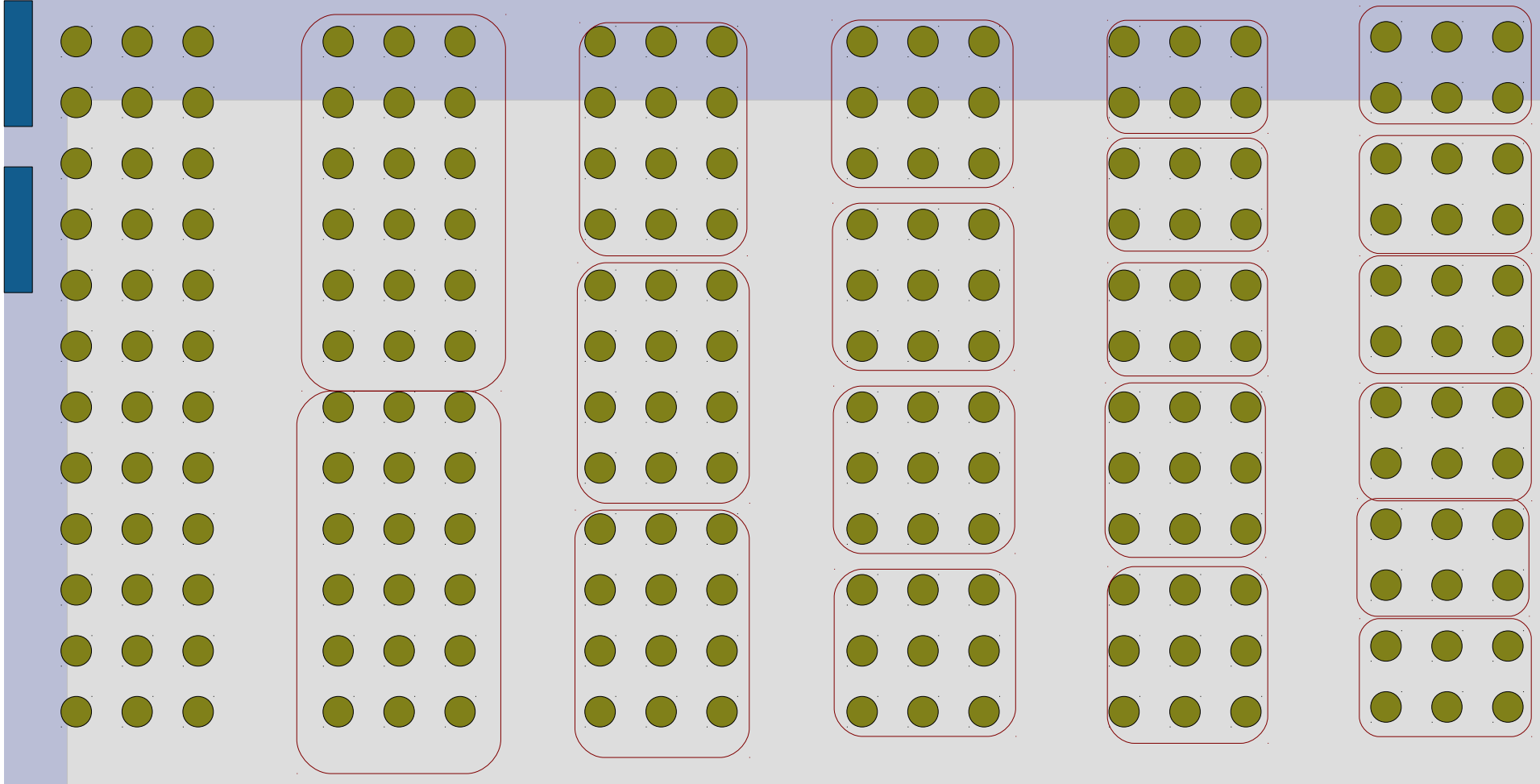


Exemplo – Diferenças Finitas 2D

$$\nabla^2 u_{ij} \approx \frac{1}{\Delta h^2}\left(4u_{i+1,j}+4u_{i-1,j}+4u_{i,j+1}+4u_{i,j-1}-20u_{i,j}+\right.$$
$$\left. u_{i+1,j+1}+u_{i-1,j-1}+u_{i+1,j-1}+u_{i-1,j-1}\right)$$

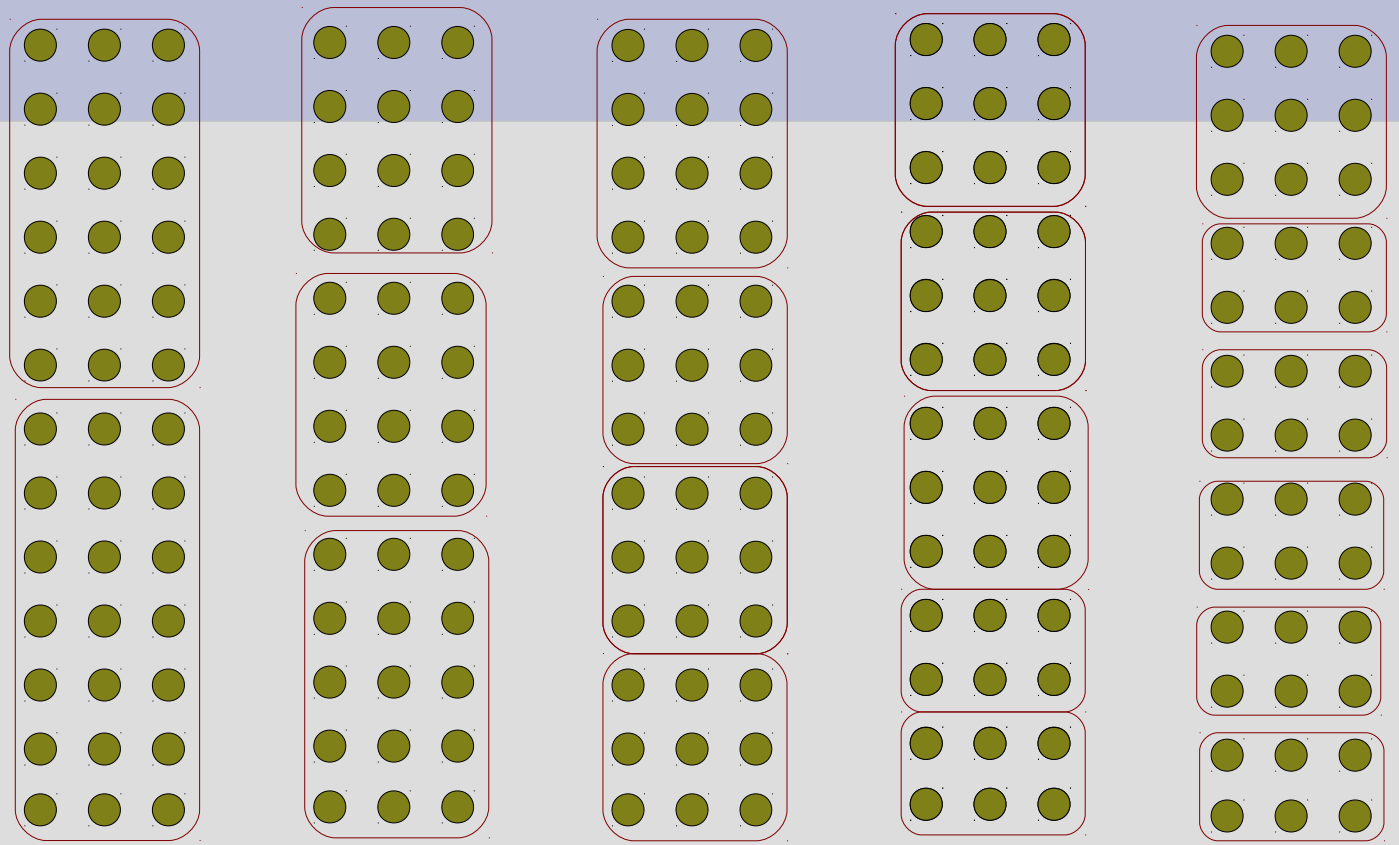


Balanço de Carga



$t_1 = 36$ $t_2 = 18$ $t_3 = 12$ $t_4 = 9$ $t_5 = 9$ $t_5 = 6$

Balanço de Carga



T_2=21; T_3=15; T_4=12; T_5=9; T_6=9;

E_2=85% E_3=80% E_4=75% E_5=80% E_6=66%

Comunicação em Programas Paralelos

- Comunicação não é instantânea!
- O tempo de comunicação pode ser uma parcela importante do tempo total de um algoritmo paralelo;
- Custo de comunicação ("communication penalty"):

$$CP = \frac{T_{total}}{T_{comp}}$$

$$E_p = \frac{T_1}{pT_p} = \frac{T_1}{pCPT_{comp}}$$

Comunicação em redes

- Modelo:
  - Comunicação através de uma rede.
  - Mensagens trocadas através de pacotes de tamanho variável.
  - Armazena e encaminha (“store & forward”) em cada nó.
- Modelo simples, apenas para mostrar alguns dos problemas envolvidos.

Atrasos devidos à comunicação

$$D = P + R\,L + Q$$

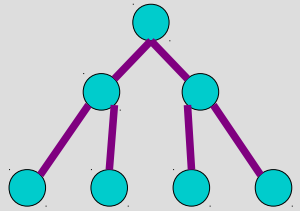
D: tempo total
P: processamento + propagação
Q: enfileiramento
R: tempo para 1 bit
L: tamanho da mensagem em bits

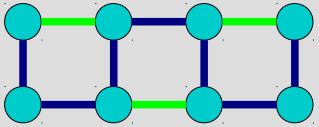
Topologia de Conexão

- Topologia: G = (N, A), onde cada arco representa uma conexão (“link”) bidirecional, direta, sem erros, entre dois processadores;
- A comunicação pode ser simultânea em todas as conexões de um nó;
- Pacotes de mesmo tamanho atrasam o mesmo tempo em todas as conexões;
- Para simplificar, cada pacote leva uma unidade de tempo para cruzar cada conexão;

Características Importantes

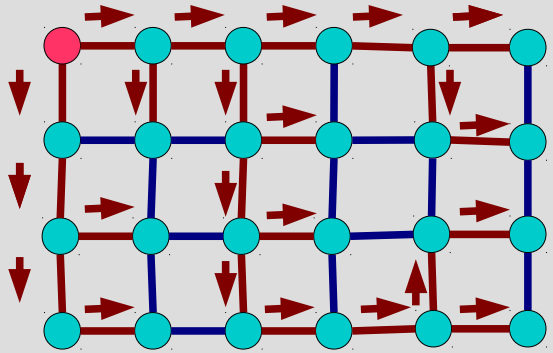
- *Diâmetro:* maior distância entre um par de nós;
- *Conectividade:* é uma medida do número de caminhos independentes entre dois nós;
- *Flexibilidade:* facilidade de executar eficientemente vários algoritmos;
- Custo para algoritmos padrão;



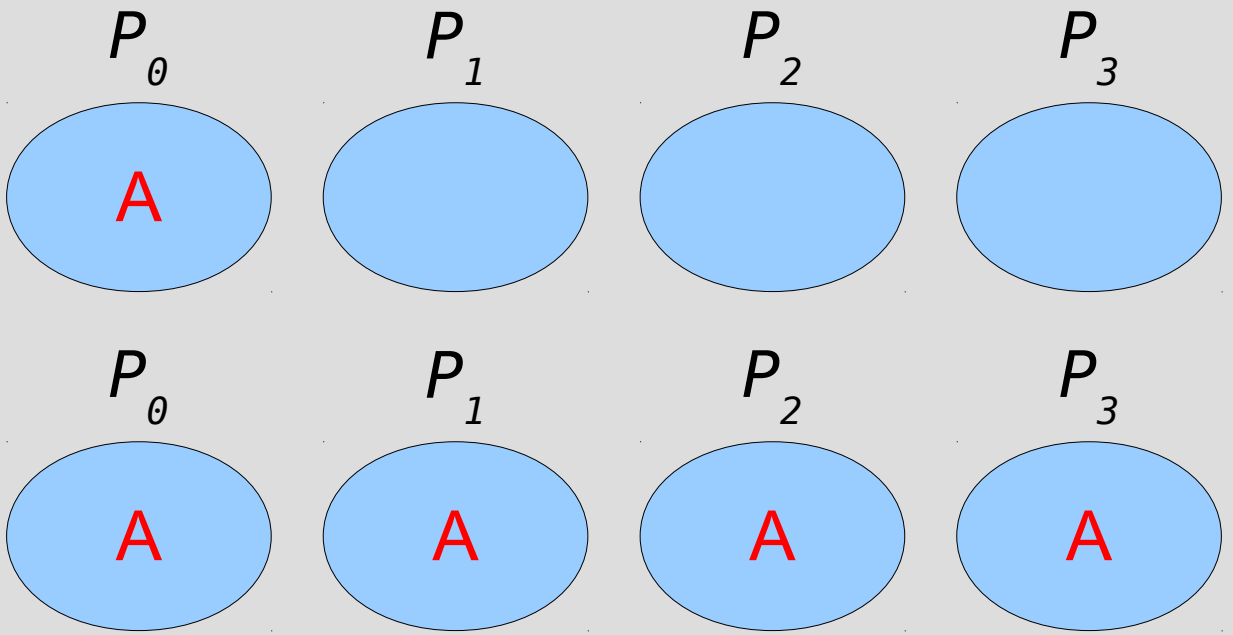


Algoritmos Padrão: Broadcast

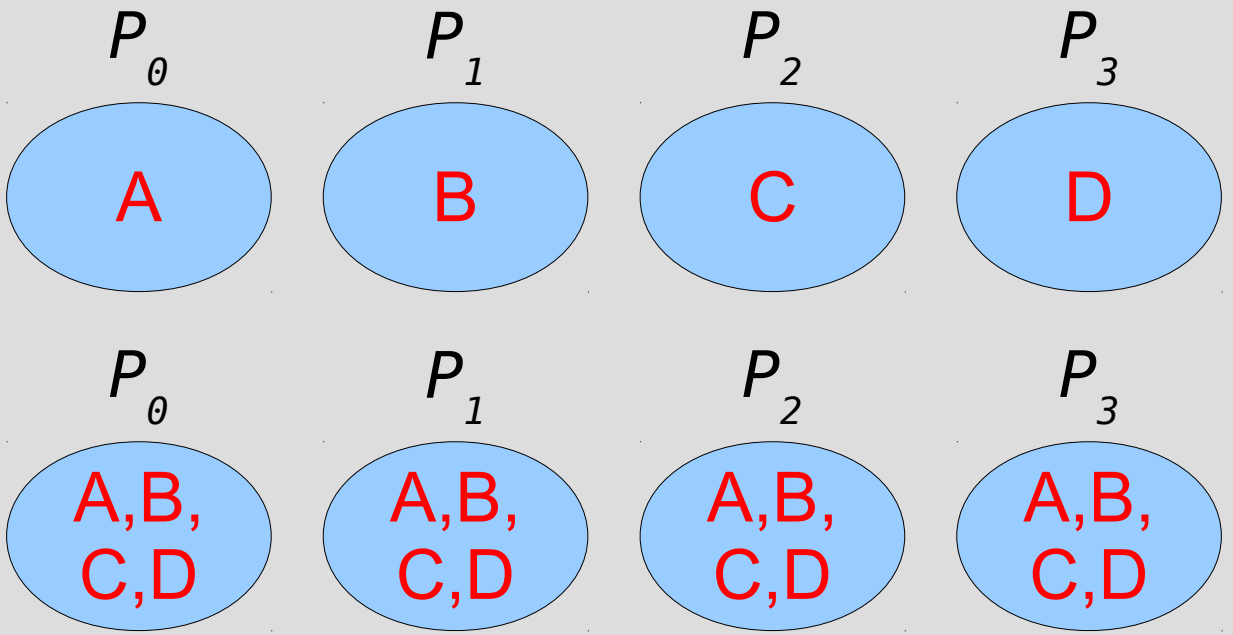
- "Broadcast" (Difusão???)
  - *Única ("single node")*: um nó envia o mesmo pacote para todos os outros nós;
  - *Múltipla ("multinode")*: *cada* nó envia um pacote para todos os outros nós;
- "Spanning tree" com raiz no nó de origem;



Single Node Broadcast

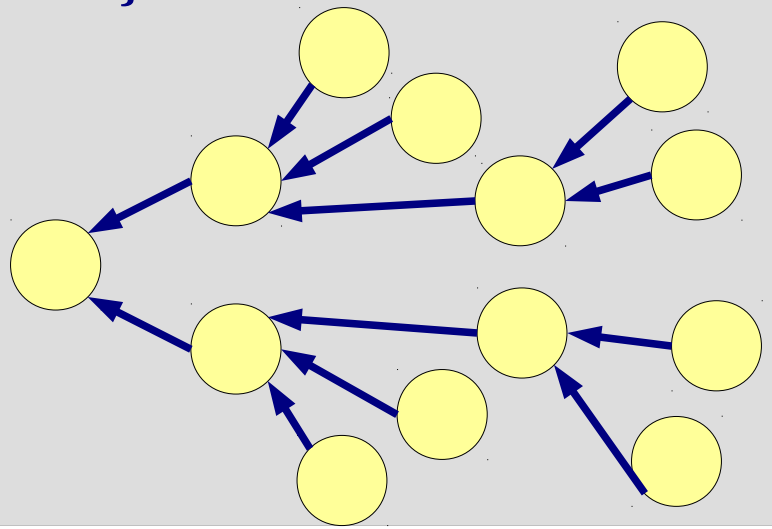


Multi Node Broadcast

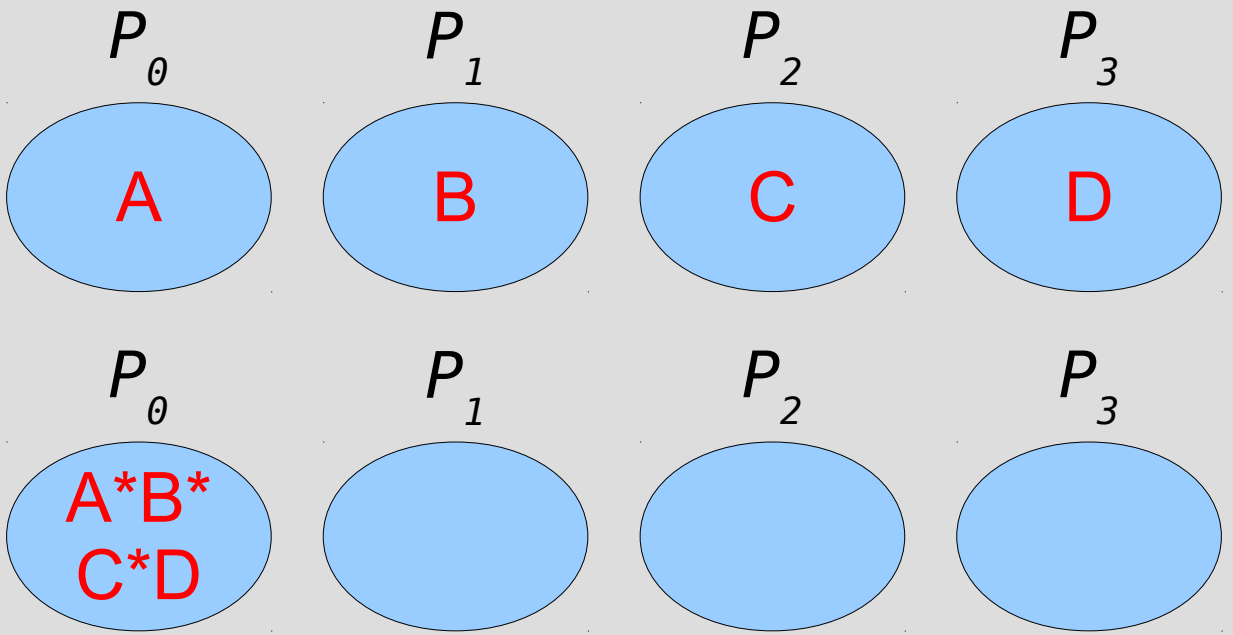


Algoritmos Padrão: Acumulação

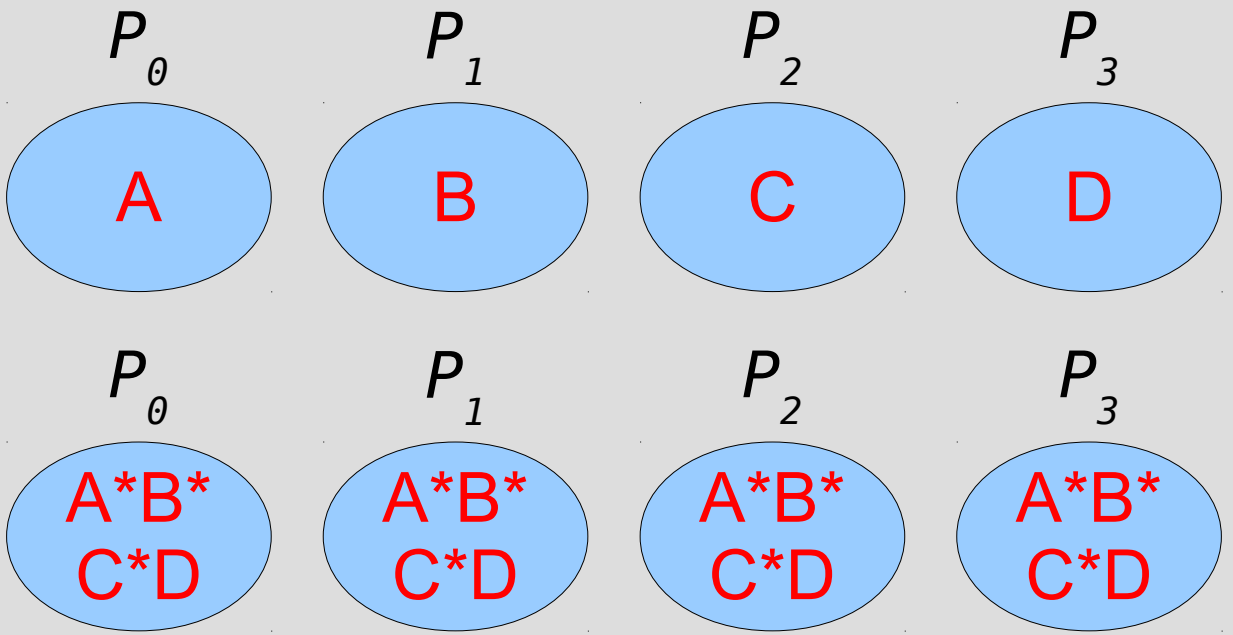
- *Única:* cada nó envia para um único nó um pacote com um componente de um resultado global – Tipicamente acumulados no caminho!
- *Múltipla:* acumulações únicas simultâneas em todos os nós.



Single Node Accumulation



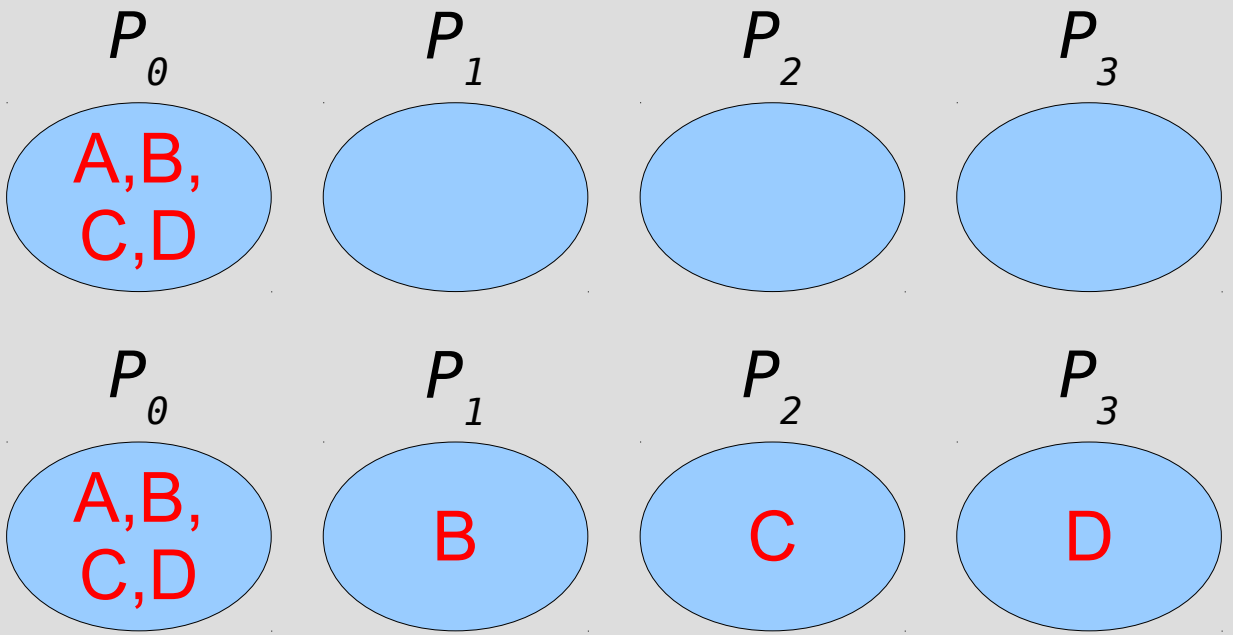
Multi Node Accumulation



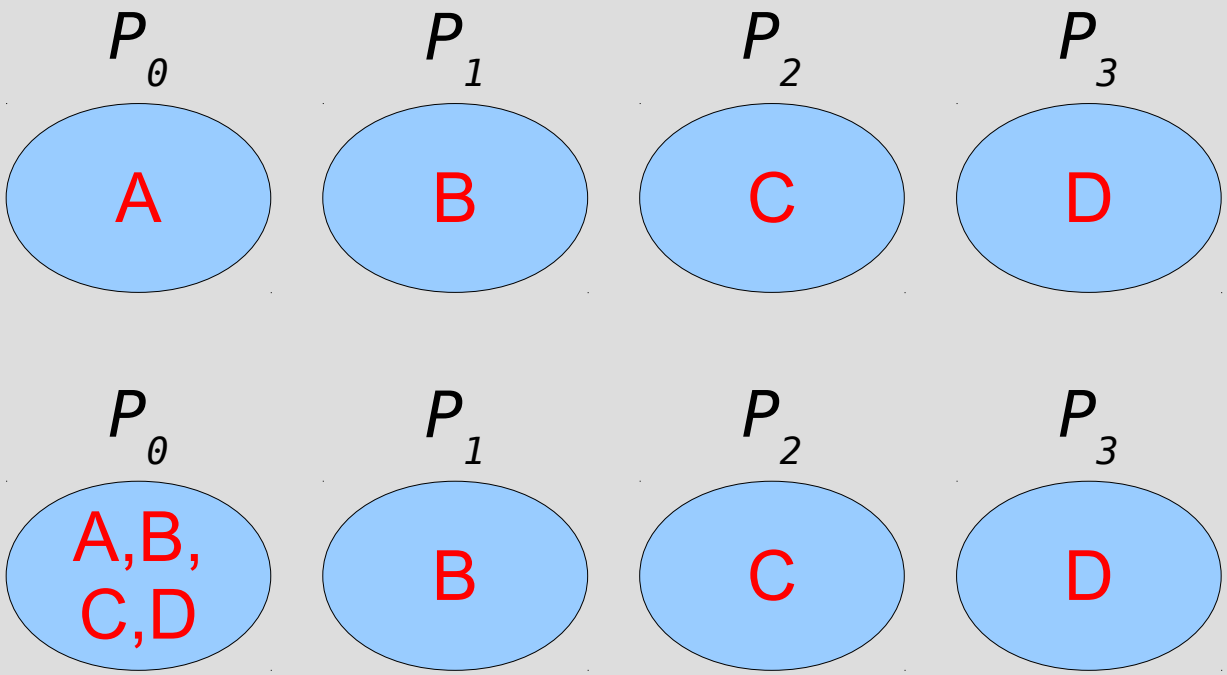
Algoritmos Padrão: Troca Total

- “Single node Scatter” - Espalhamento
  - Um nó manda um pacote diferente para cada outro nó;
- “Single node Gather” - Coleta
  - Um nó recebe um pacote diferente de cada outro nó;
- “Total Exchange”
  - Cada nó manda um pacote diferente para cada outro nó;

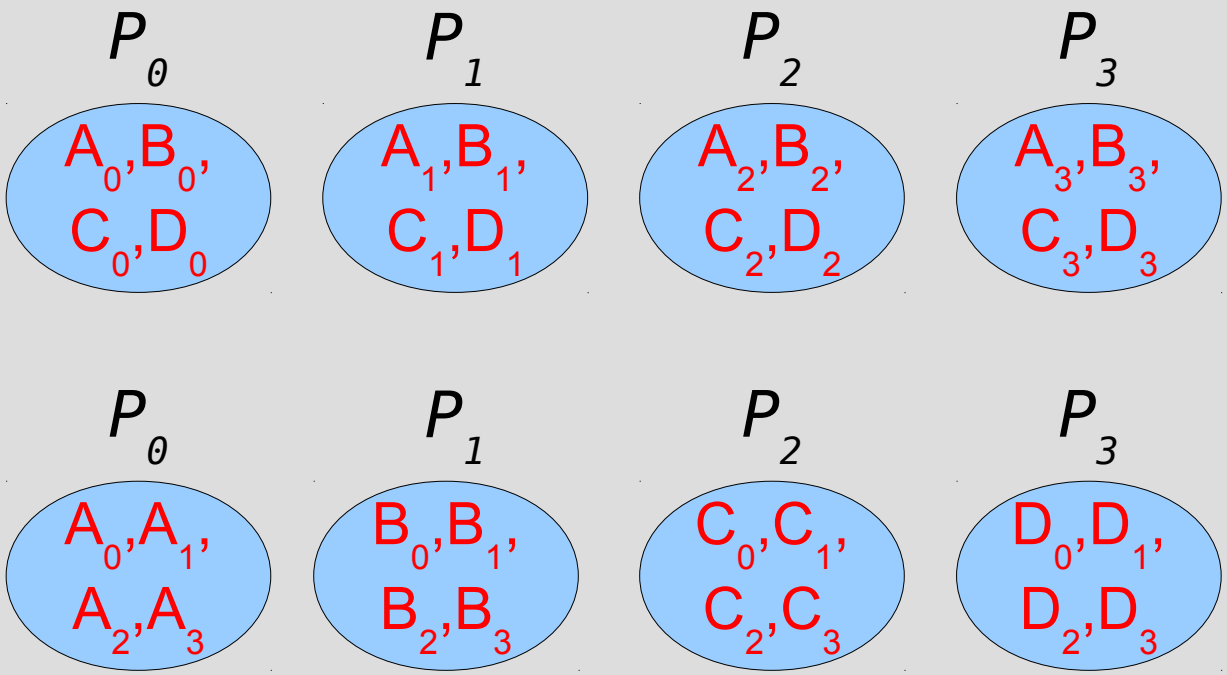
Single Node Scatter



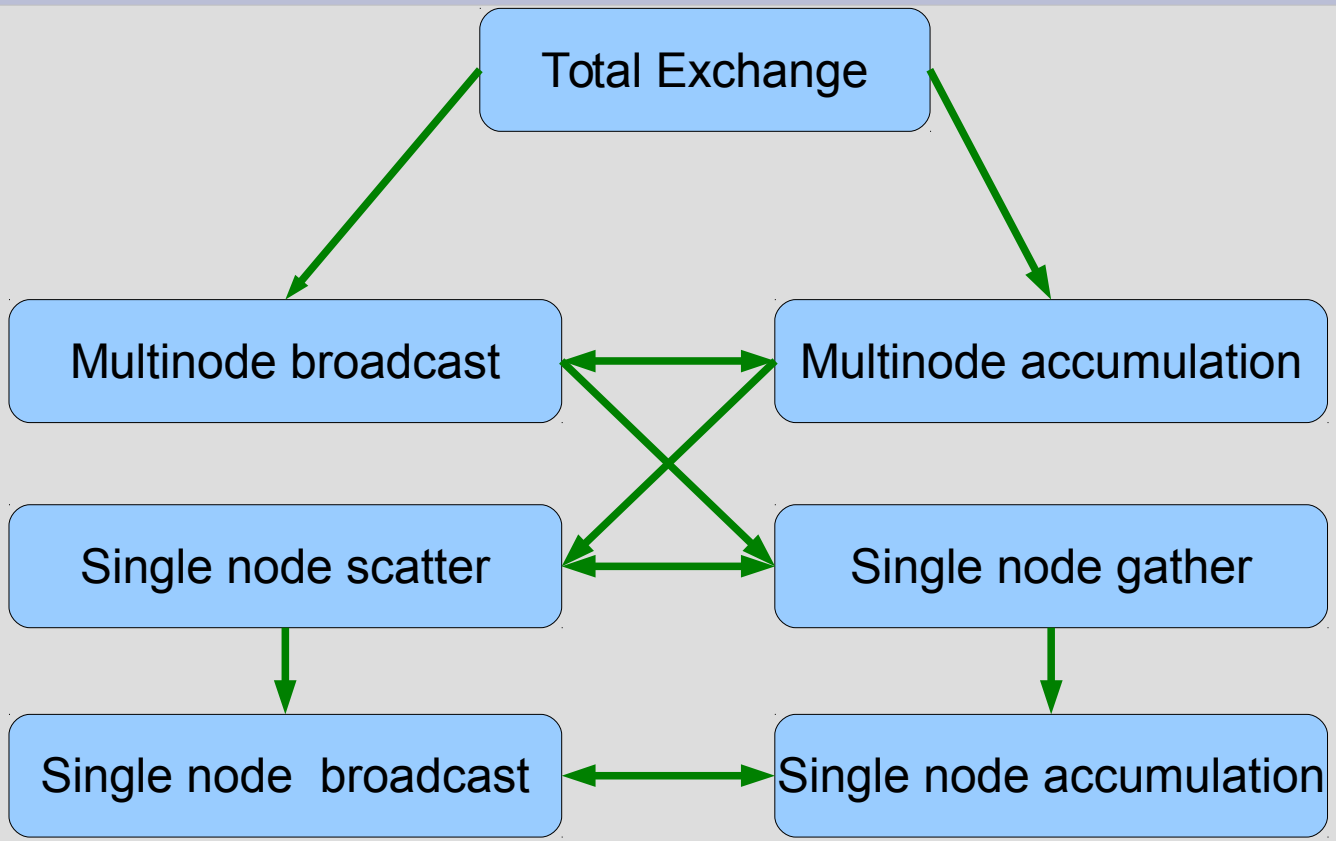
Single Node Gather



Total Exchange (MNS, MNG)



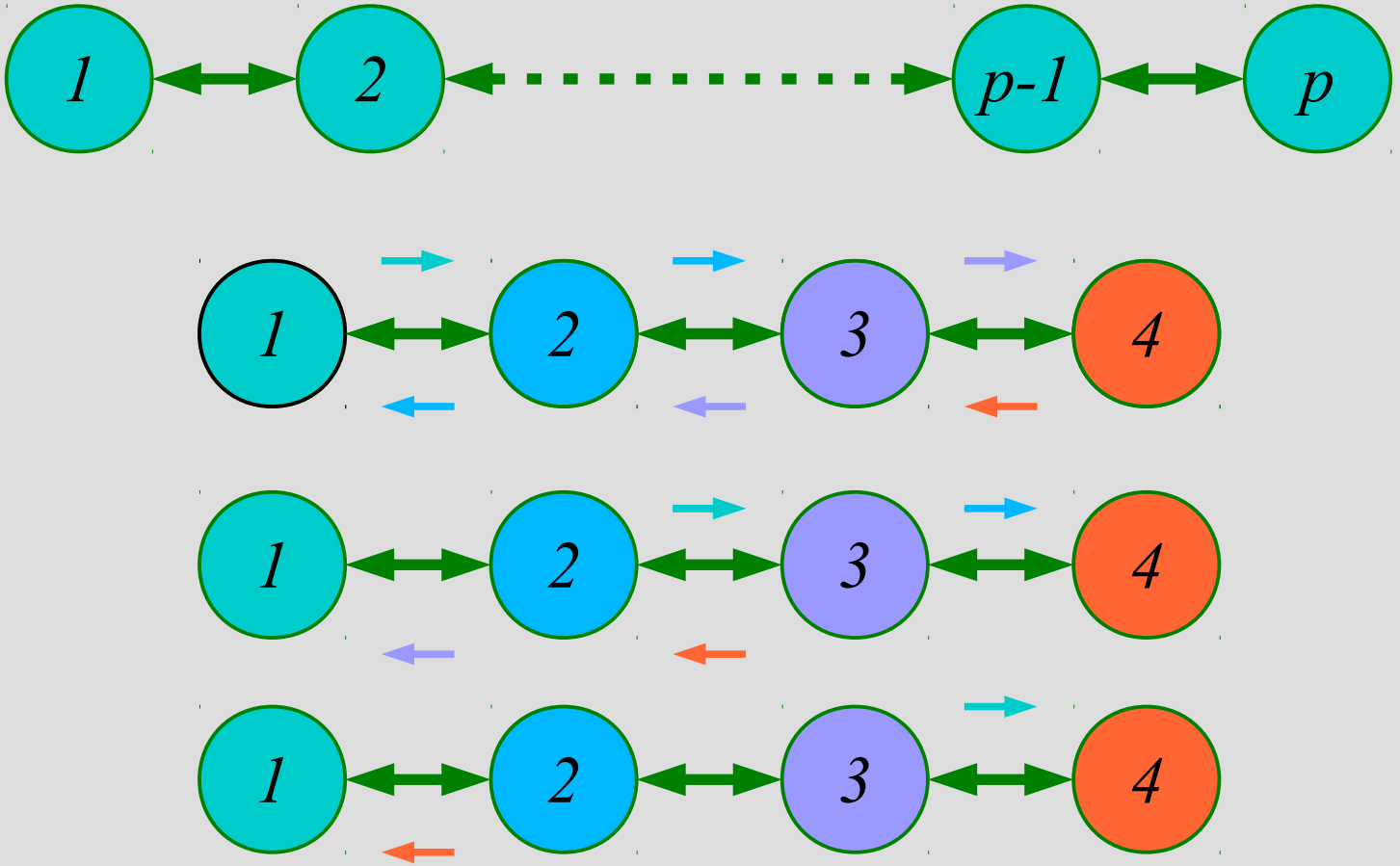
Algoritmos Padrão: Hierarquia



Topologias Padrão: Grafo Completo

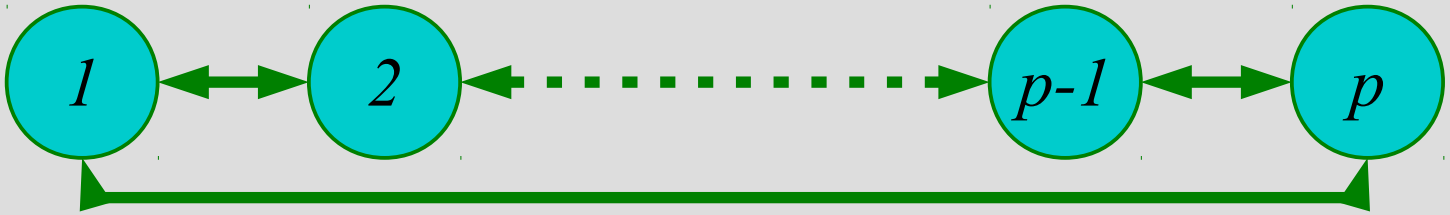
- Cada nó conectado a todos os outros nós;
- Certamente o mais rápido e mais flexível;
- “Bus” - redes pequenas;
- “Switched networks” - redes pequenas, médias e grandes;
- É esquema padrão atual;

Topologia Padrão: Arranjo Linear

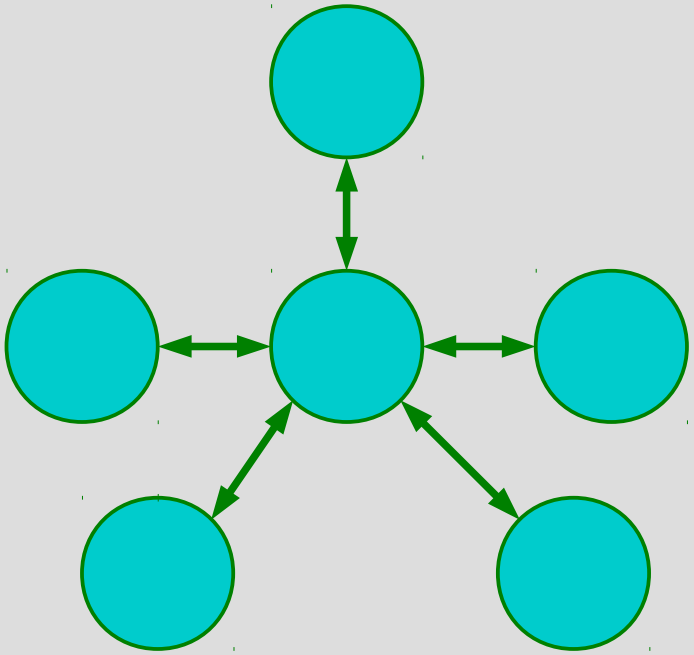


Topologia Padrão: Anel

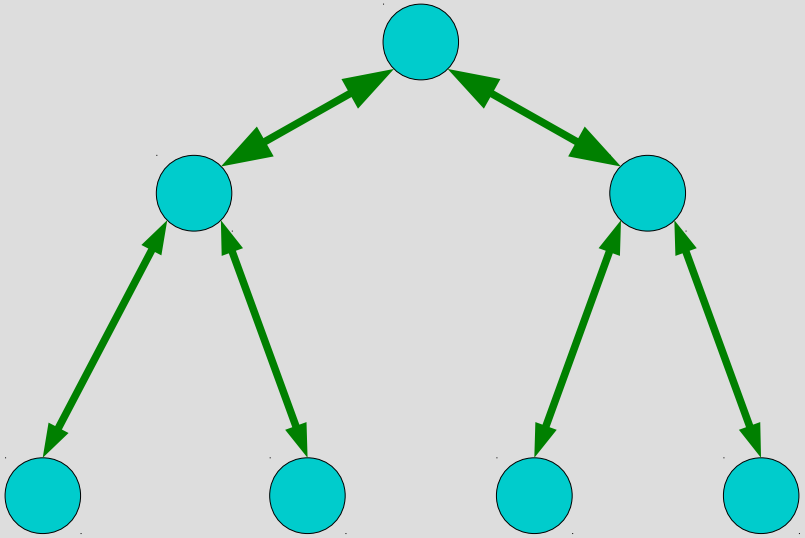
- Comportamento de mesma ordem que o arranjo linear! Valem todos os mesmos comentários.



Topologia Padrão: Árvore



Estrela

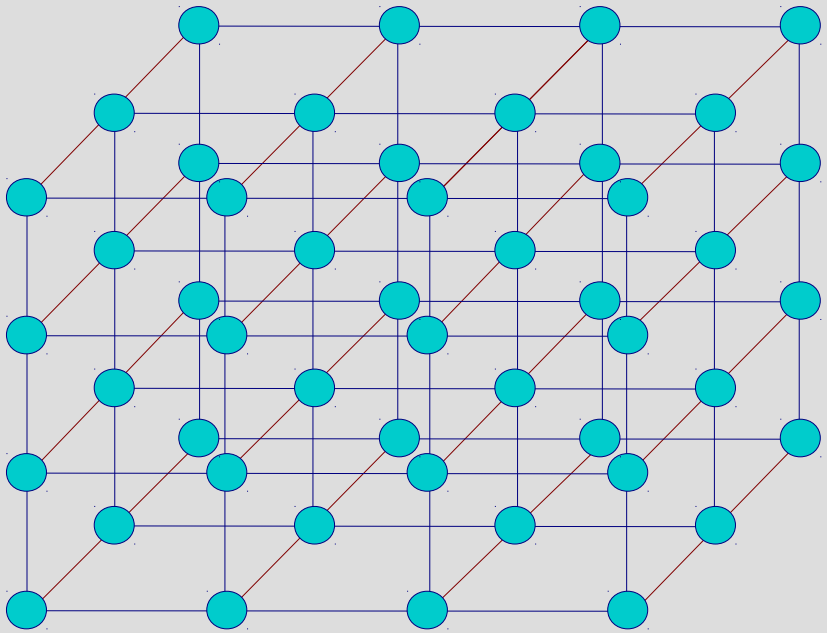


Árvore Binária Balanceada

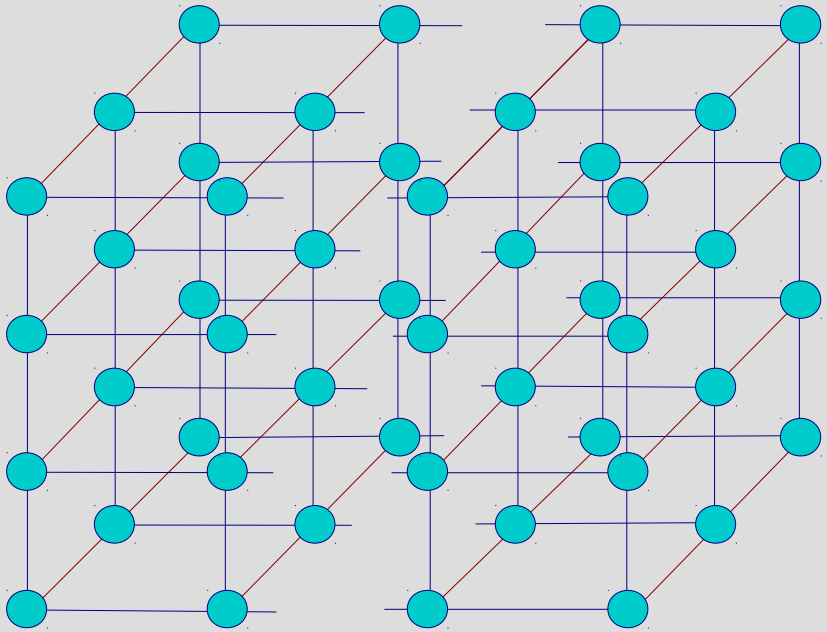
Topologia Padrão: Matrizes

- 2, 3, *d* dimensões
- Normalmente toroidal

$4 \times 4 \times 3 = 48$ processadores

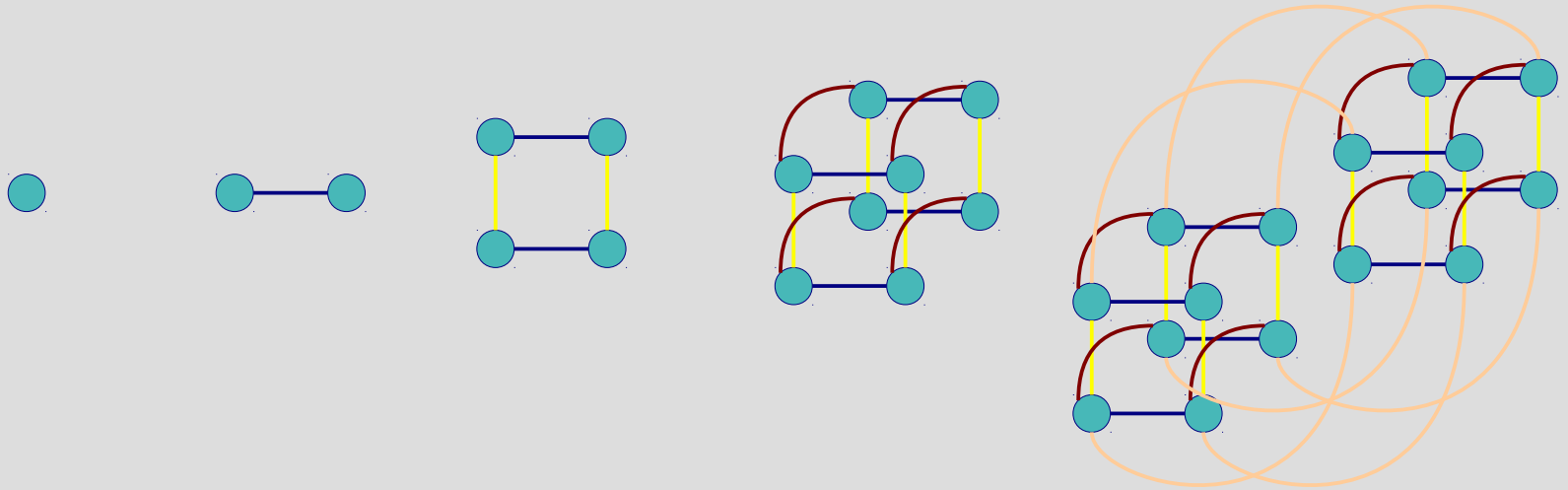


Topologia Padrão: Matrizes - TE



- Há $p^{(d-1)/d}$ links cortados pelo plano separador.
- Há $p/2$ processadores em cada bloco.
- Cada bloco recebe $(p/2)^2$ mensagens.
- Cada link deve transmitir $O(p^{(d+1)/d})$ mensagens.
- 2d – $O(p^{3/2})$; 3d – $O(p^{4/3})$

Topologia Padrão: Hipercubos



d=0 *d=1* *d=2* *d=3* *d=4*

Comparação Entre Topologias

Problema	*Anel*	*Árvore*	*Malha*	*Hipercubo*
Single node Broadcast (ou Single Node Accumulation)	$O(p)$	$O(\log p)$	$O(p^{1/d})$	$O(\log p)$
Single Node Scatter (ou Single Node Gather)	$O(p)$	$O(p)$	$O(p)$	$O(p/\log p)$
Multinode Broadcast (ou Multinode Accumulation)	$O(p)$	$O(p)$	$O(p)$	$O(p/\log p)$
Total Exchange	$O(p^2)$	$O(p^2)$	$O(p^{(d+1)/d})$	$O(p)$

Aspectos de Comunicação e Computação

- Muitos (todos!) os sistemas modernos permitem que mensagens sejam transmitidas enquanto o processador faz alguma outra coisa.
- Desta forma, o custo de comunicação pode ser reduzido. Exemplo:

Iteração por blocos:

$$\vec{x}_j(t+1) = f_j(\vec{x}(t)); \quad j = 1, \ldots, p$$

- Cada bloco pode ser enviado para todos os outros processadores assim que é atualizado

Aspectos de Comunicação e Computação

- O custo de comunicação tende a diminuir com o tamanho do problema!
- Para a iteração por blocos anterior:

$$\frac{T_{\text{comm}}}{T_{\text{comp}}} = \frac{\mathrm{O}(n)}{\mathrm{O}(n\,k)} = \mathrm{O}\left(\frac{1}{k}\right)$$

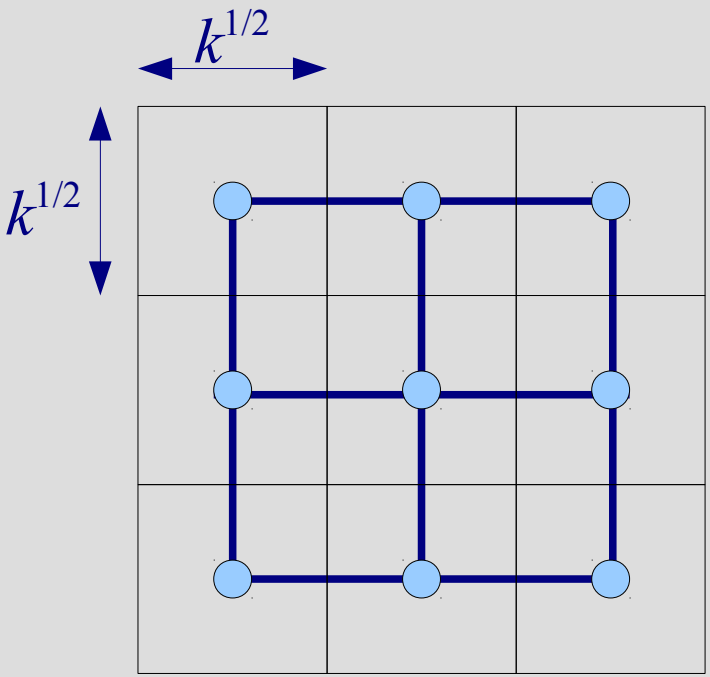
$$S_p = \frac{\mathrm{O}(n^2)}{\mathrm{O}(n) + \mathrm{O}(n\,k)} = \mathrm{O}(p)$$

Aspectos de Comunicação e Computação

- Claramente, a idéia fundamental é sempre ter um problema "grande" em cada processador;
- A topologia da rede determina o menor tamanho do problema que pode ser resolvido eficientemente;
- Exemplo anterior em um hipercubo:

$$\frac{T_{\text{comm}}}{T_{\text{comp}}} = \frac{\mathrm{O}\left(\frac{k\,p}{\log p}\right)}{\mathrm{O}(n\,k)} = \mathrm{O}\left(\frac{1}{k \log p}\right)$$

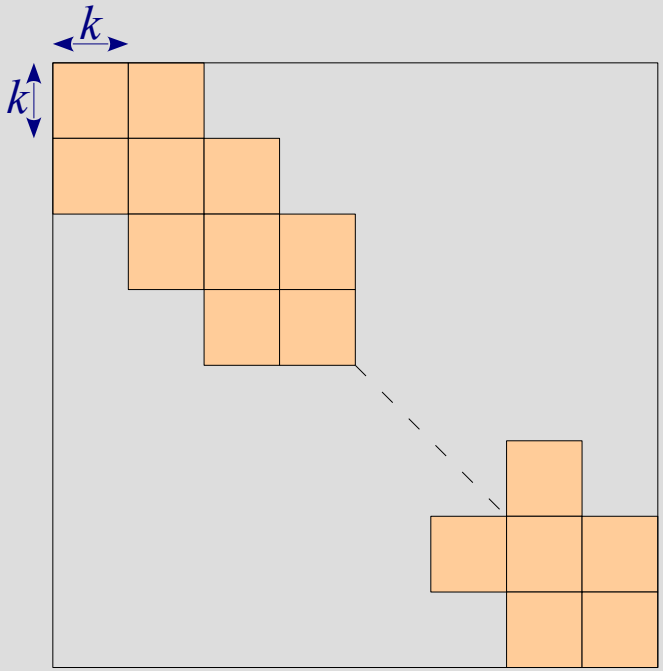
Efeito da Esparsidade



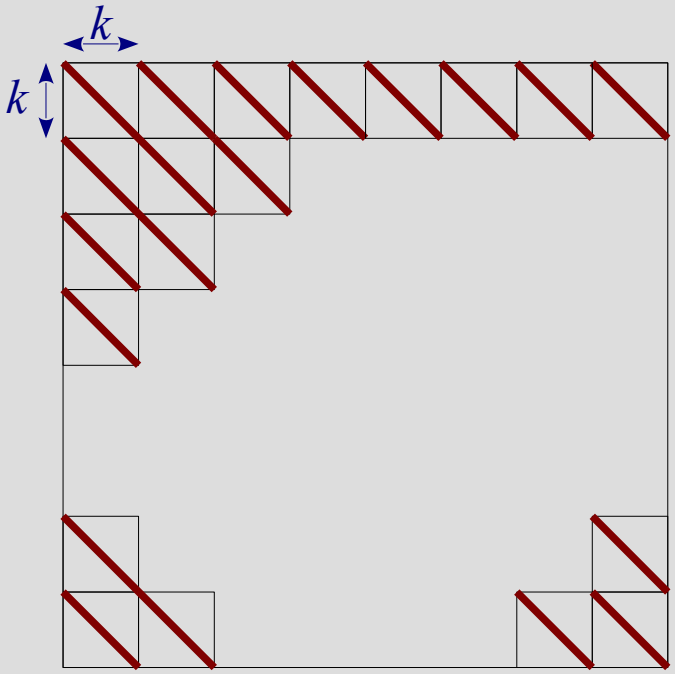
- Discretização PDE 2D.
- Efeito Área-Perímetro.
- k variáveis por bloco.

$$T_{\text{comm}} = \text{O}(\sqrt{k}) \text{ em uma malha}$$
$$T_{\text{comp}} = \text{O}(k)$$
$$\frac{T_{\text{comm}}}{T_{\text{comp}}} = \text{O}\left(\frac{1}{\sqrt{k}}\right)$$

Efeito da Esparsidade



$$T_{\text{comm}} = \mathrm{O}(k)$$
$$T_{\text{comp}} = \mathrm{O}(k^2)$$



$$T_{\text{comm}} = \mathrm{O}(n) = \mathrm{O}(pk)$$
$$T_{\text{comp}} = \mathrm{O}(n) = \mathrm{O}(pk)$$

Exemplos: Produto Interno

$$a \cdot b = \sum_{i}^{n} a_i \; b_i \quad \text{onde} \begin{cases} a, b \in \mathbb{R}^n \\ n = p\,k \end{cases}$$

Cada processador armazena

a_i e b_i, para $i = (i-1)k+1, \ldots, i\,k$

calcula

$$c_i = \sum_{j=(i-1)k+1}^{ik} a_j b_j$$

e acumula em um nó.

Produto Interno: Processadores

- Considerando um arranjo linear:

Acumulação em arranjo linear

O tempo para calcular o produto interno é:

$$k\alpha + (\alpha+\beta)\left\langle \frac{p}{2} \right\rangle = \frac{n\alpha}{p} + (\alpha+\beta)\left\langle \frac{p}{2} \right\rangle$$

Produtos parciais em paralelo

otimizando:

$$p \approx \left(\frac{2\alpha n}{\alpha+\beta} \right)^{\frac{1}{2}}$$

Produto Interno: Processadores

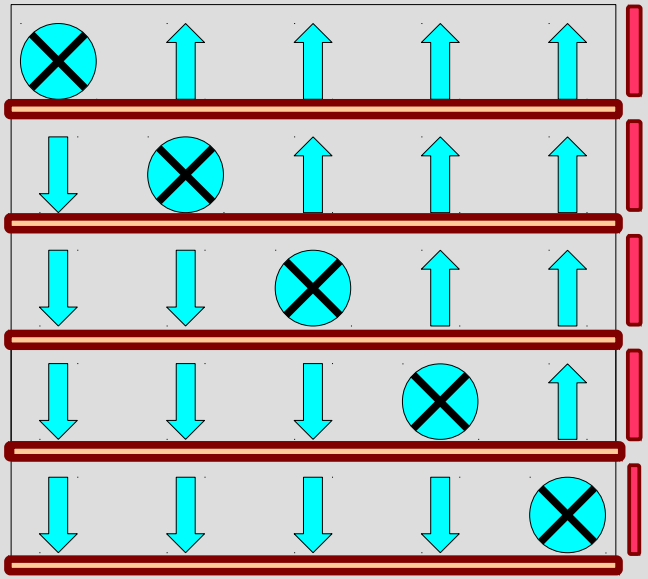
- Caso crítico:

  Se $\beta > (2n-1)\alpha$, então $p < 1$.

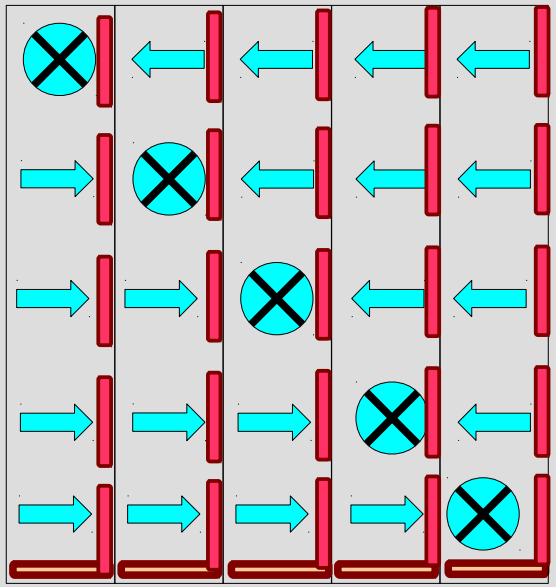
- Hipercubo:

  $$\frac{n\alpha}{p} + (\alpha+\beta)\log p, \text{ otimizando: } p \approx \left(\frac{\alpha n}{\alpha+\beta} \right)$$

Exemplo: Matriz x Vetor



Armazenamento
por linhas (MNB)



Armazenamento
por colunas (MNA)

Matriz Vetor: Processadores

- Considerando um arranjo linear:

Produtos parciais em paralelo

Multinode broadcast

O tempo para a multiplicação é:

$$\frac{\alpha n^2}{p} + (p-1)\left(\beta + \frac{n\gamma}{p}\right)$$

otimizando para p:

$$p \approx n\left(\frac{\alpha - \gamma/n}{\beta}\right)^{\frac{1}{2}}$$

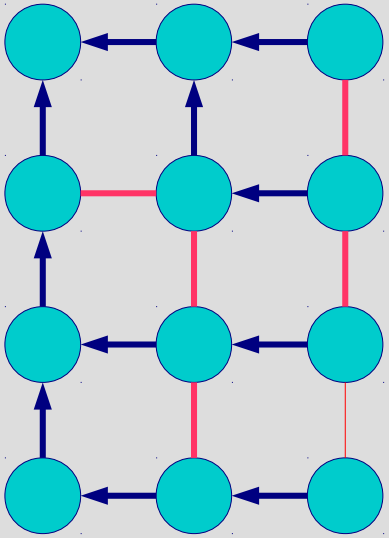
Sincronização: Algoritmos Síncronos

- O problema é composto de fases sequenciais, onde cada fase é composta por componentes atribuídos a processadores diferentes.
- O tempo no qual a computação de cada componente é realizada é independente dos outros componentes.
- Dentro de cada fase os processadores não interagem.
- Ao final de cada fase, os processadores interagem, trocando os dados necessário para a próxima fase.

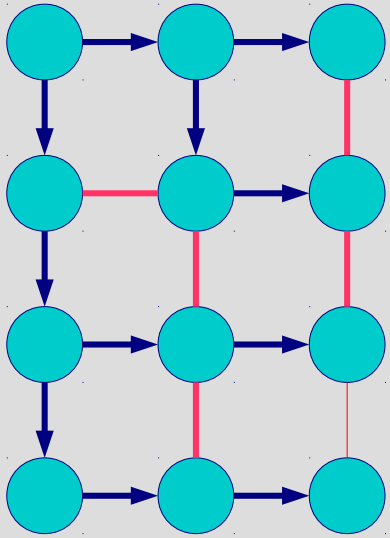
Sincronização Global

- Cada processador aguarda a mensagem de terminação de seus filhos, e assim que todas as mensagens de terminação chegam e sua computação termina, manda uma mensagem de terminação para seu “pai” e fica parado esperando a mensagem de início de fase.
- O processador raiz, após receber as mensagens de terminação de seus filhos, e terminar sua computação, envia mensagens de início de fase para seus filhos.
- Cada processador recebe a mensagem de início de fase, inicia seu processamento e envia a mensagem para seus filhos.

Sincronização Global



Mensagens de término de fase

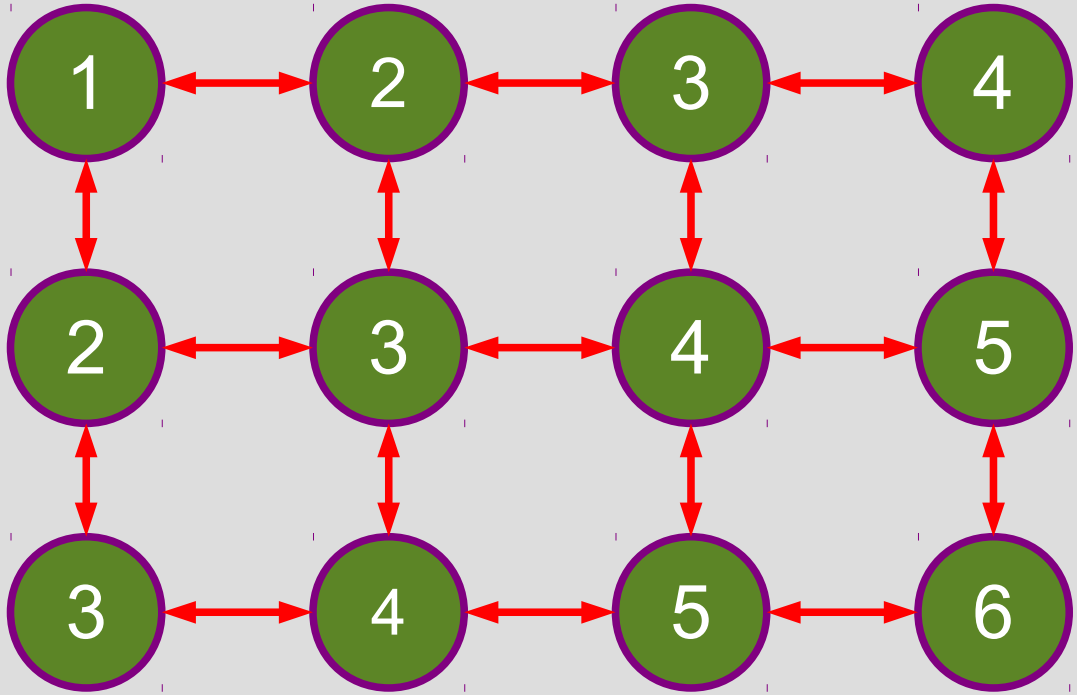


Mensagens de início de fase

Sincronização Local

- Cada processador sabe quais mensagens deve receber em cada fase (de quem e o que!)
- Cada processador envia todas as mensagens que deve assim que possível, preferencialmente ao mesmo tempo em que computa.
- Quando termina sua computação, espera por todas as mensagens que deve receber.
- Assim que recebe todas as mensagens, inicia a nova fase.

Sincronização Local

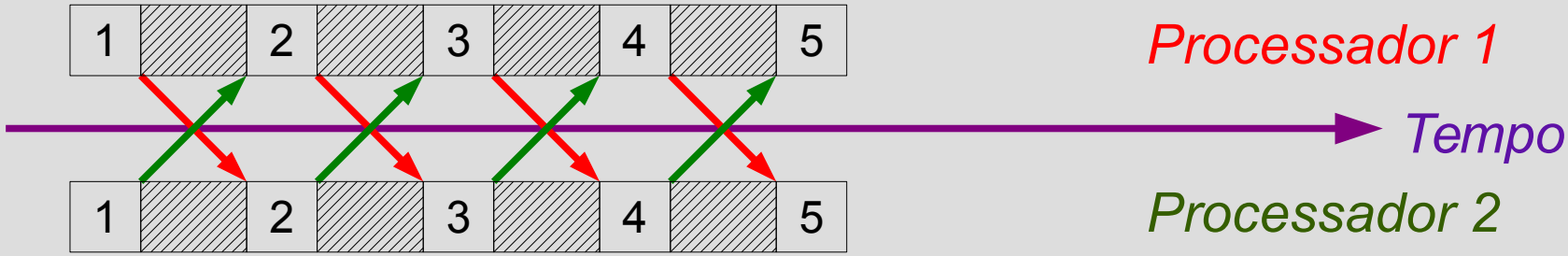


Iteração de Jacobi

Custo da Sincronização

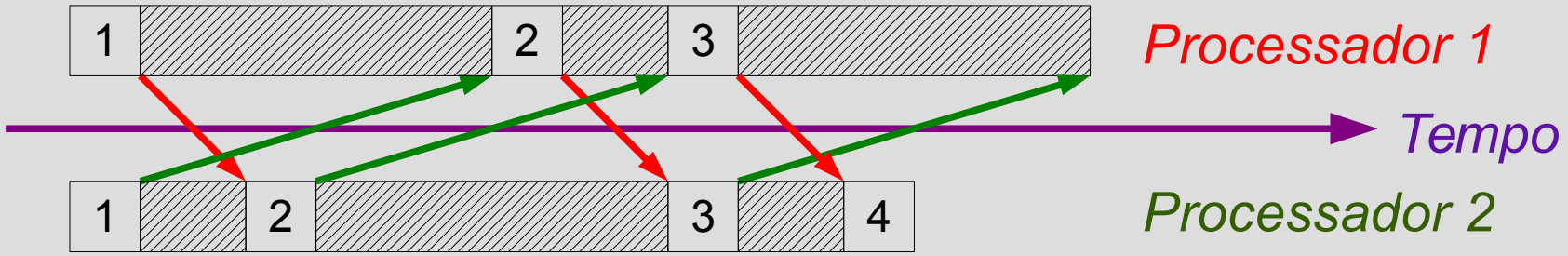
- Comunicação instantânea

- Comunicação não instantânea

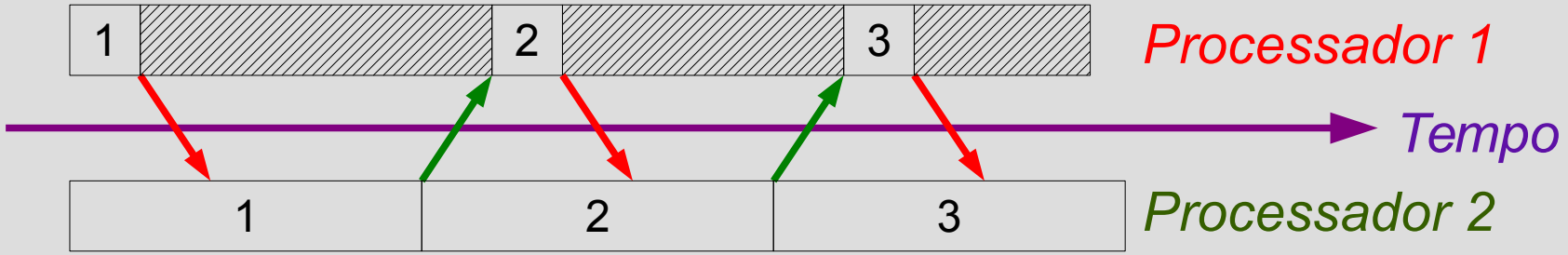


Custo da Sincronização

- Um canal lento



- Um processador lento



Algoritmo Assíncrono

- Comunicação não instantânea