UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE QUIXADÁ

Modelos de Computação

Prof. João Marcelo Uchôa de Alencar

joao.marcelo@ufc.br

Introdução

- Vamos ver os três modelos básicos de computação paralela.
- As topologias de interconexão que tem impacto no desempenho de máquinas reais baseadas nos modelos apresentados.
- Entender como podemos analisar a complexidade de um algoritmo ou computação paralela.

História dos Sistemas Paralelos

- Seja Π um problema qualquer...
  - *N-body*
  - Ordenação de inteiros
  - Qualquer problema computacional.
- Existem infinitos algoritmos (potencialmente) para resolver Π, mas estamos interessados na divisão:
  - Algoritmos sequenciais: uma operação é realizada por etapa.
  - Algoritmos paralelos: várias operações podem ser realizadas por etapa.
- Em computação paralela, queremos projetar um algoritmo paralelo para Π.

História dos Sistemas Paralelos

- Seja *P* um algoritmo paralelo qualquer.
  - Dizemos que há paralelismo em *P*.
  - Pode ser explorado por vários tipos de computadores paralelos.
    - Várias operações de *P* podem ser executadas simultaneamente em várias unidades de processamento de um computador C_1.
    - Da mesma forma, podem ser distribuídas nas unidades funcionais replicadas de um pipeline de um computador C_2, com um único processador.
    - Na ausência de uma máquina paralela, *P* pode até ser executado de forma sequencial em computado serial C_3.
- Importante observar que *P* pode ser executado em uma máquina serial utilizando todo o sistema, mas um algoritmo serial *S* não fará proveito de uma máquina paralela.

História dos Sistemas Paralelos

- Seja *C(p)* um computador paralelo com *p* unidades de processamento.
  - O **desempenho** de *P* em *C(p)* depende do tipo de *C* e da quantidade *p*.
  - Não necessariamente *P* conseguirá fazer uso efetivo de *C(p)*
    - Paralelismo potencial.
    - Paralelismo efetivo.
- Mas o que seria o desempenho?
  - Tempo de execução do algoritmo serial em um computador serial *versus* tempo de execução do algoritmo paralelo em *C(p)*?
  - Uma vez que consigo um algoritmo *P* que executa mais rápido para um dado *p* em *C(p)*, faz sentido ficar comparando com o algoritmo serial?

História dos Sistemas Paralelos

- Na prática, queremos ter uma **base** para comparar *P* em uma máquina *C(p)* variando o *p*.
  - Nós vamos tomar como base a execução de *P* em *C(1)*.
  - O desempenho é dado pelo *speedup*: $S \stackrel{\text{def}}{=} \frac{T_{seq}}{T_{par}}$
  - T_{par} é o tempo de execução paralelo de *P* em *C(p)*.
  - T_{seq} é o T_{par} de *P* em *C(1)*.
  - A execução paralela de *P* e *C(p)* é *S* vezes mais rápida do que a execução sequencial de *P*.
- O *speedup* é definido dessa forma para mais na frente permitir avaliar se incrementar *p* faz sentido ou não.

História dos Sistemas Paralelos

- O quanto está sendo usando das *p* unidades de processamento de *C(p)* para obter um *speedup?* É chamada eficiência.
  - $E \stackrel{\text{def}}{=} \frac{S}{p}$, pode ser reescrita da forma $E \stackrel{\text{def}}{=} \frac{T_{seq}}{pT_{par}}$
  - Como $T_{par} \leq T_{seq} \leq pT_{par}$, temos que o *speedup* tem limite superior em *p* e a eficiência é menor ou igual a 1.
- Em geral, para qualquer *C* e *p*, a execução paralela de *P* em *C(p)* pode ser no máximo *p* vezes mais rápida do que a execução de *P* em uma única unidade de processamento.
  - Na prática, algumas mágicas da hierarquia de memória dão *speedups* mais vantajosos.

História dos Sistemas Paralelos

Como determinamos *Tpar* para um problema *P* em *C(p)*?

Qual relação em *Tpar* e *C* (a natureza do computador paralelo)?

Quais as propriedades de *C* que afetam o valor de *Tpar*?

Modelando Computação Paralela

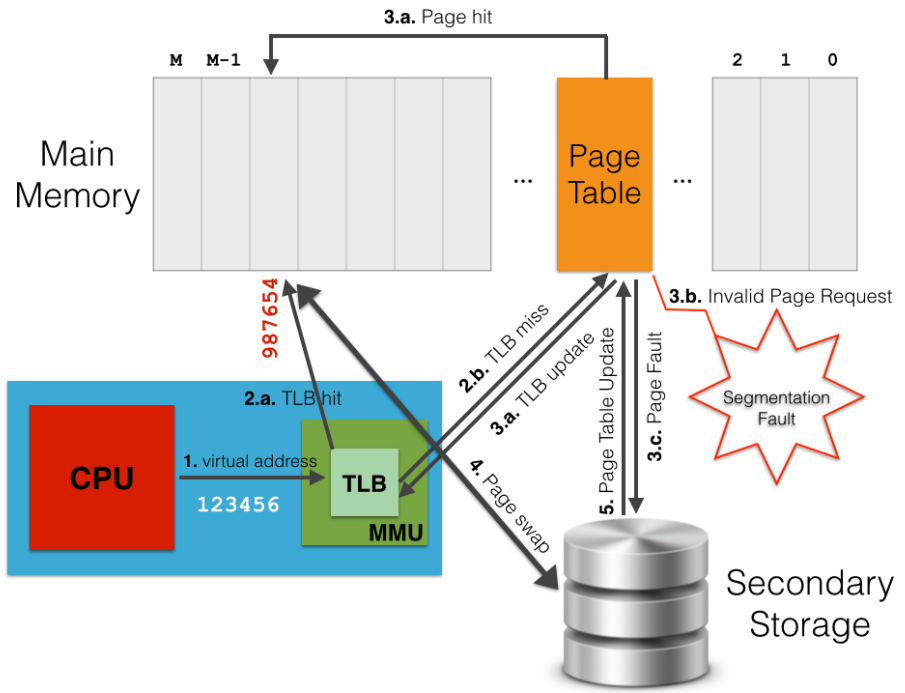
- Características do *hardware* que influenciam a modelagem de algoritmos paralelos:
  - Conexão entre as unidades de processamento.
  - Memória compartilhada ou local.
  - Técnica de sincronização entre as unidades.
- Quais características de computadores paralelos **devem ser consideradas** e quais **podem ser ignoradas** no projeto e análise de algoritmos paralelos?
  - Para a computação serial, vários modelos foram propostos (máquina de Turing, cálculo lambda, etc).
  - Cada modelo tem como objetivo abstrair propriedades irrelevantes do *hardware* para determinados tipos de computação.

Modelando Computação Paralela

- Modelo **Random Access Machine** (RAM)
  - Abstração para máquinas seriais.

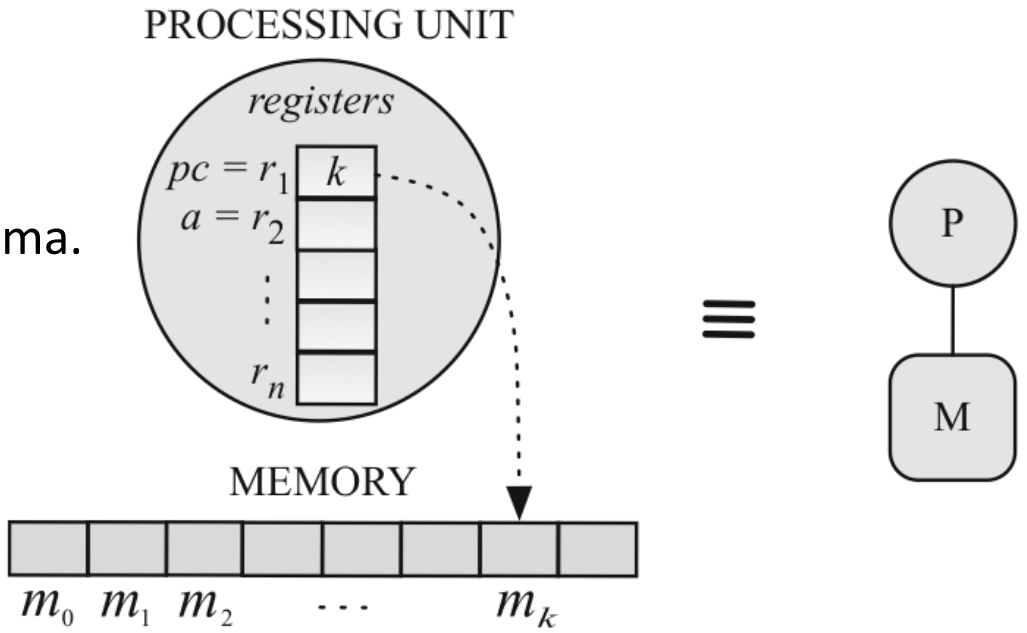
Para um computador real, o acesso a memória é complexo:

- Memórias cache
- Memória virtual
- Memória física



Modelo RAM:

- Unidade de processamento .
- Memória:
  - $m_0, m_1, m_2, ..., m_i, ..., m_n$
  - Dado um *i* arbitrário ler ou escrever m_i exige **tempo constante.**
- Registradores:
  - $r_1, r_2, r_3, ..., r_i, ... r_n$
  - Contador de programa ($pc=r_1$)
  - Acumulador ($a=r_2$)
- O programa é carregado no início da memória.
- Os dados ocupam as posições seguintes.
- A execução prossegue incrementando o contador de programa.



Qual modelo adequado para **computação paralela**?

Modelando Computação Paralela

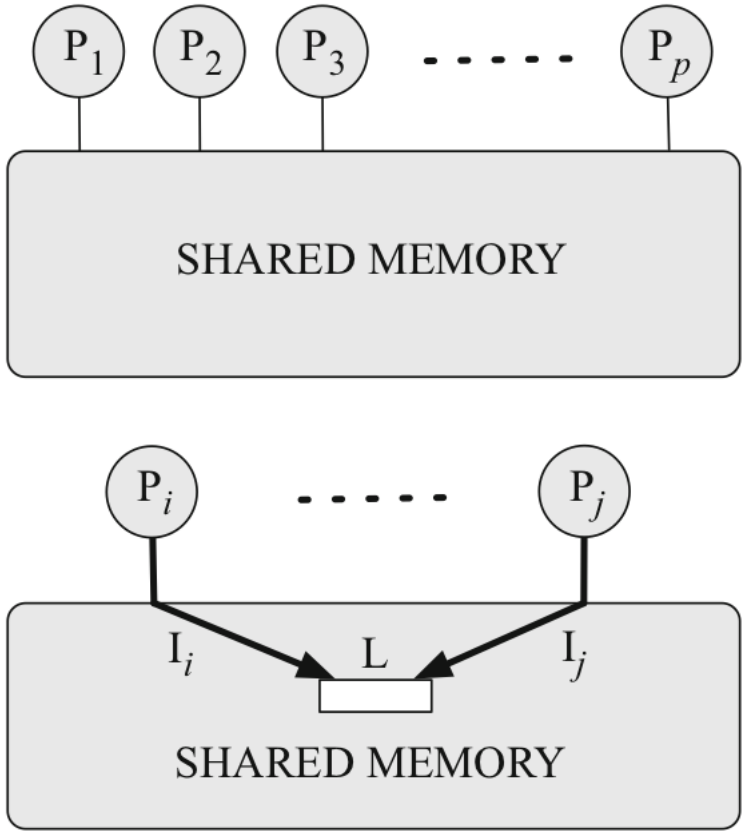
- Modelos Multiprocessados
  - Baseados no RAM.
  - Assume um computador *C(p)* com $p \geq 2$ unidades de processamento.
  - Diferem na organização da memória e como o acesso é feito
  - Três principais:
    - *Parallel Random Access Machine* (PRAM).
    - *Local Memory Machine* (LMM).
    - *Modular Memory Machine* (MMM).

Modelando Computação Paralela

- *Parallel Random Access Machine* (PRAM)
  - **Qualquer** unidade de processamento pode, em uma única etapa, acessar **qualquer** endereço da memória.
  - A requisição é feita diretamente à memória, sem intermediários.
  - Não há limite na quantidade de unidades de processamento *p*.
  - As palavras na memória possuem o mesmo tamanho finito.
- Qualquer pessoa com conhecimento de arquitetura e sistemas operacionais sabem que esse modelo é uma **idealização**.

Modelando Computação Paralela

- O acesso simultâneo de P_i e P_j em L, através das instruções I_i e I_j, não tem problema na leitura.
- Na escrita, qual o valor resultante?
  - O *hardware* pode serializar as escritas.
  - Mas qual a ordem? I_i, I_j ou I_j, I_i.
  - Imprevisível.
  - Uma leitura e escrita simultâneos também tem o mesmo problema.
- Seria o caso do algoritmo considerar soluções em *software* para sincronizar as escritas na memória.



Modelando Computação Paralela

- A limitação da escrita levou a definição de variações do PRAM
  - Modelos que permitem certos tipos de acesso ao mesmo endereço.
  - Modelos que evitam a imprevisibilidade do acesso simultâneo.
- Três principais variações, em ordem decrescente de realismo:
  - *Exclusive Read Exclusive Write* PRAM (EREW-PRAM)
  - *Concurrent Read Exclusive Exclusive* Write (CREW-PRAM)
  - *Concurrent Read Concurrent Write* (CRCW-PRAM)

Modelando Computação Paralela

- EREW-PRAM
  - Modelo mais realista.
  - Se qualquer acesso simultâneo for feito, o programa para.
  - É responsabilidade dos projetistas de algoritmos impedir o acesso simultâneo.
- CREW-PRAM
  - Permite leitura simultânea.
  - Proíbe escrita simultânea.
  - Novamente, cabe ao projetista de algoritmos a evitar a escrita simultânea.

Modelando Computação Paralela

- CRCW-PRAM
  - Menos realista.
  - Permite leitura e escrita simultânea.
  - Restrições para a escrita:
    - **CONSISTENT**-CRCW-PRAM: escrita simultânea permitida, desde que todas as unidades de processamento estejam tentando escrever o mesmo valor em um endereço.
    - **ARBITRARY**-CRCW-PRAM: várias unidades de processamento tentam escrever em um mesmo endereço, mas apenas uma conseguirá. Não há como saber qual.
    - **PRIORITY**-CRCW-PRAM: cada unidade de processamento tem uma prioridade designada. Na escrita concorrente, apenas a unidade com maior prioridade conseguirá escrever.
    - **FUSION**-CRCW-PRAM: Uma operação associativa e comutativa é aplicada a todos os valores que as unidades estão tentando escrever em um mesmo endereço. Exemplo de operações: soma, produto, função máximo, função mínimo, etc.
  - Novamente, cabe ao projetista de algoritmos lidar com essas restrições ao desenvolver seus algoritmos paralelos.

Modelando Computação Paralela

- Os modelos EREW-PRAM, CREW-PREAM e CRCW-PRAM diferem no poder de expressão de algoritmos?
  - Sim, mas por pouco.
  - Um algoritmo projetado no EREW-PRAM é mais lento do que um algoritmo para o mesmo problema projetado no CRCW-PRAM por um fator de no máximo $O(\log p)$.
  - A diferença se dá pelas etapas extras que o EREW-PRAM teria que executar para atingir o mesmo resultado que o CRCW-PRAM.
- Portanto, quando projetando um algoritmo paralelo, podemos usar o modelo PRAM mais adequado e ainda ter um resultado válido para as outras variantes.

Modelando Computação Paralela

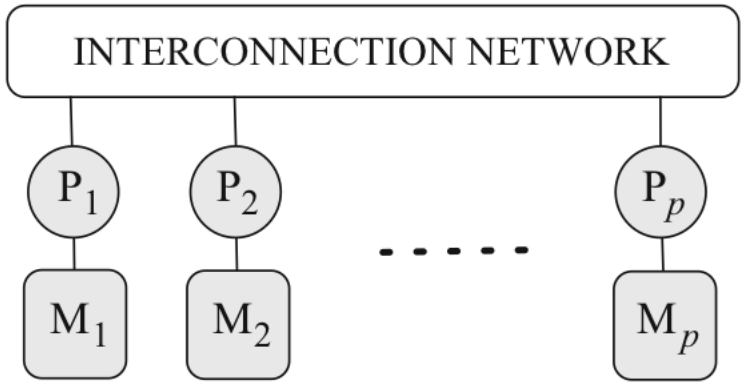
- Se mesmo o EREW-PRAM não é 100% realista, por que usamos o PRAM para projetar algoritmos?
  - Muitas vezes o algoritmo projetado no PRAM não é prático ou fácil de ser implementado em um computador real.
  - Porém, o simples fato dele existir prova que o problema Π é paralelizável, servindo como incentivo para pesquisar um algoritmo prático.
- Pensar em uma algoritmo no PRAM permite raciocinar sobre a decomposição em tarefas e no particionamento dos dados.
  - Onde as tarefas executarão, *threads*, processos, etc, pode ficar para o projeto do algoritmo específico para uma máquina real.
  - Da mesma forma, onde os dados residirão para cada tarefa acessa, memória local, distribuída, compartilhada, etc.

Modelando Computação Paralela

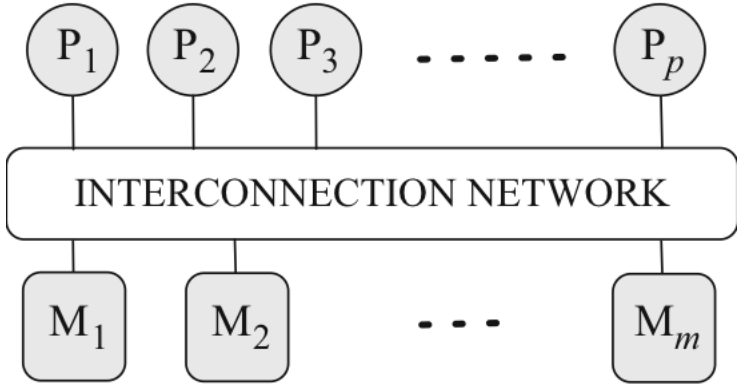
- O modelo *Local-Memory-Machine (LMM).*
  - *p* unidades de processamento, cada uma com sua memória local, em conexão direta.
  - As unidades são ligadas por uma rede de interconexão.
  - Uma unidade pode requisitar um endereço de memória à outra unidade.
  - Todas as operações locais executam em uma única etapa.
  - O tempo para acessar memória remota depende de:
    - A capacidade de banda da rede.
    - O padrão de acesso às memórias remotas.

Modelando Computação Paralela

- O modelo *Memory Module Machine* (MMM)
  - *p* unidades de processamento, sem memória local.
  - *m* módulos de memória.
  - A rede de interconexão conecta as unidades aos módulos.



LMM



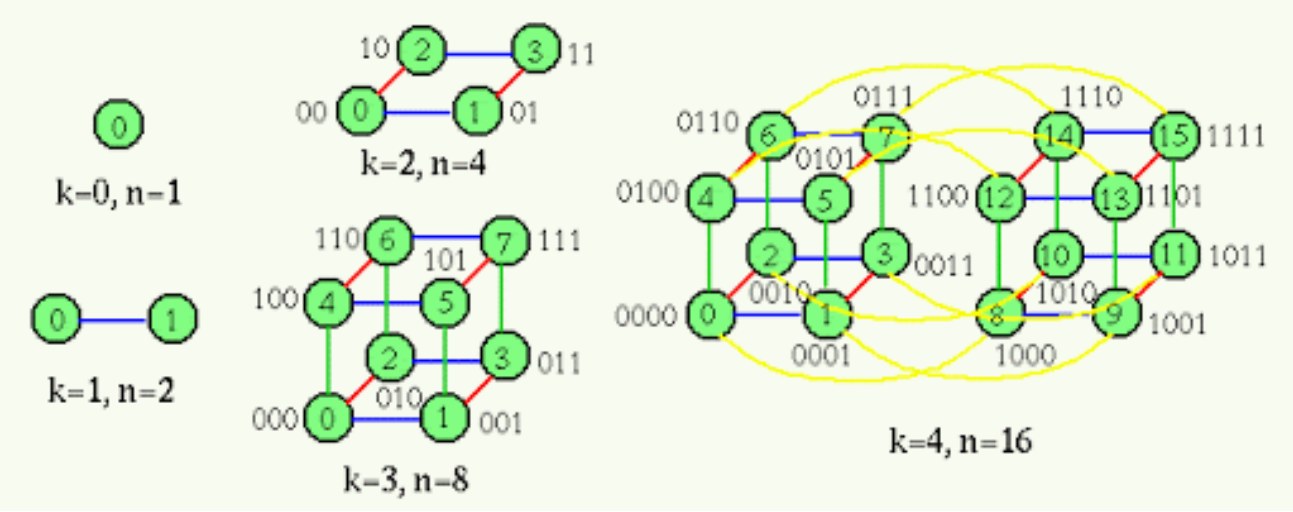
MMM

O Impacto da Comunicação

- Características principais de *C(p)*:
  - Organização da unidade central de processamento (CPU).
  - Tipo de rede de interconexão.
- Cada vez as aplicações paralelas dependem mais do tempo de comunicação do que do tempo de computação.
  - Escalabilidade.
  - Eficiência.
- Fatores de desempenho de uma rede de interconexão:
  1. Roteamento.
  2. Controle de Fluxo.
  3. Topologia da rede.
- Fatores 1 e 2 possuem técnicas estabelecidas. O fator 3 ainda é objeto de estudo.
- Existem sistemas paralelos que permitem adaptar a topologia de acordo com a aplicação em execução.

Propriedades Básicas das Redes de Interconexão

- Usamos fundamentos de Teoria dos Grafos.
- A rede é modelada como:
  - Um grafo *G(N, C).*
  - *N* é um conjunto de nós comunicantes.
  - *C* é um conjunto de links de comunicação (ou canais).



Propriedades de Topologias

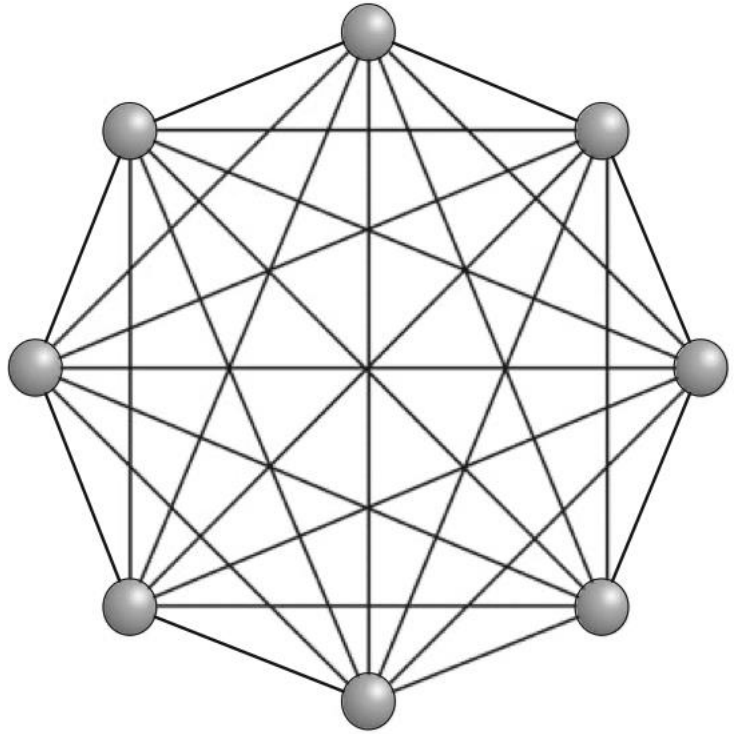
- **Grau de um nó**: o número de canais conectados a um nó de comunicação.
  - Uma rede é regular se todos os nós tiverem o mesmo grau.
- Um **caminho** em um grafo:
  - Conjunto de arestas que permitem a comunicação entre dois nós.
  - Número de saltos.
  - Podem existir vários caminhos entre dois nós.
    - Máximo.
    - Mínimo.
    - Caminho médio.
- **Diâmetro** da rede:
  - Enumera-se todos os caminhos mínimos entre dois nós.
  - O maior valor entre os caminhos mínimos é o diâmetro da rede.

Propriedades de Desempenho das Redes

- Largura de banda do **canal**:
  - A quantidade de *bits* que pode fluir em uma aresta por unidade de tempo.
  - $t_{comm} = t_s + t_d$, onde $t_d = m\ t_w$.
  - A largura de banda fica sendo então $1 / t_w$.
- Largura de banda de **bissecção**:
  - Dada uma divisão da rede em duas partes de tamanhos iguais (ou quase iguais), a largura do corte é a soma da largura de banda de todos os canais que conectam as duas novas redes.
  - A menor largura de corte possível é a largura de banda de bisseção.
- Latência:
  - Tempo necessário para um pacote viajar da origem até o destino.

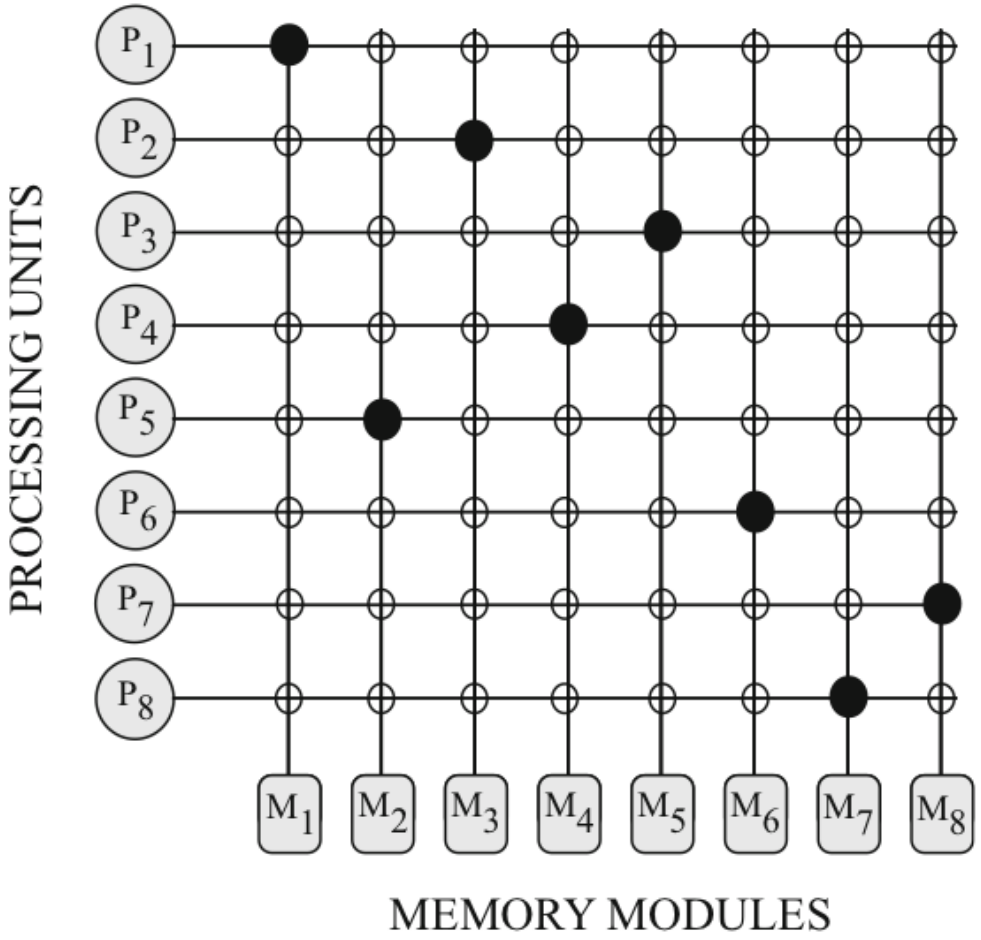
Classificação das Redes de Interconexão

- Redes Diretas
  - As conexões são feitas diretamente entre os nós.
  - Em uma rede direta completamente conectada, cada nó é conectado a todos os outros.
  - Em uma rede direta não completamente conectada, as mesmas podem precisar de roteamento através de nós intermediários.



Classificação das Redes de Interconexão

- Redes Indiretas
  - Os nós de computação são conectados através de *switches*.
  - Em geral, cada *switch* conecta um nó de computação a um módulo de memória.
  - ***Switch Crossbar* Totalmente Conectado**.
  - Tipos de bloqueio:
    - Sem bloqueio.
    - Bloqueio reconfigurável.
    - Bloqueante.

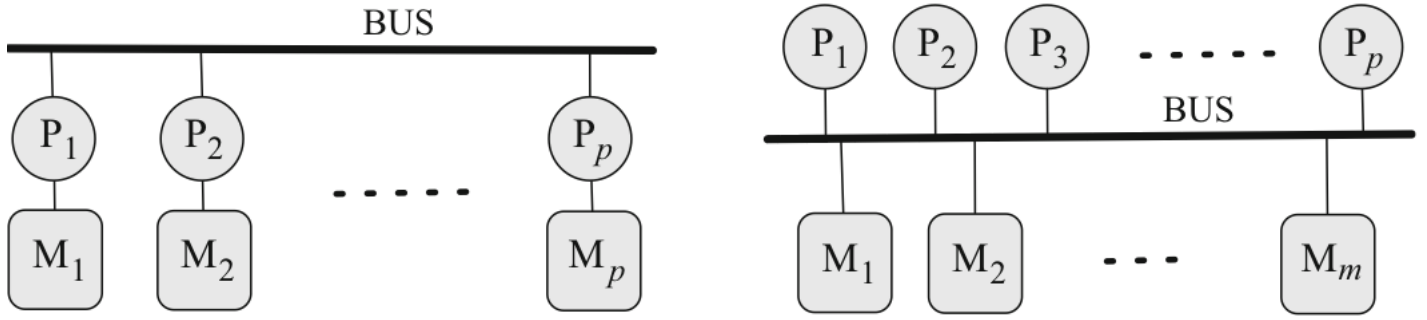


Topologias de Redes de Interconexão

- A classificação entre redes diretas e indiretas é limitante.
- Existem várias outras maneiras de conectar *p* unidades de processamento com *m* módulos de memória.
- Cada topologia tem suas características:
  - Grau do nó.
  - Largura de bissecção.
  - Diâmetro.
  - etc...
- Vamos apresentar uma visão geral das mais utilizadas em computadores reais.

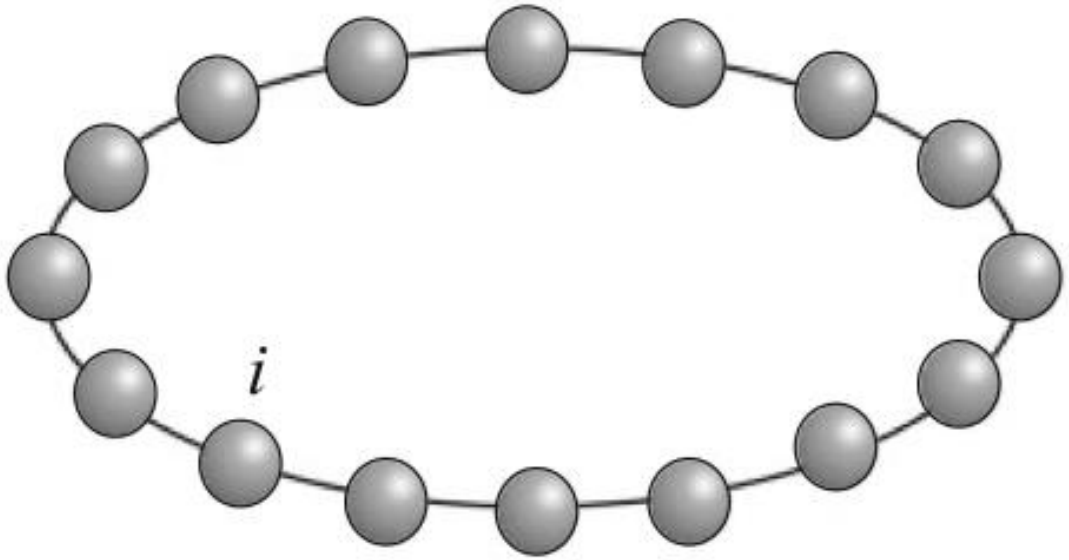
Barramento

- Canal de comunicação compartilhado.
- Só permite uma requisição por vez.
- Não tem muita escalabilidade.
- Um sistema de *caches* melhora o desempenho.



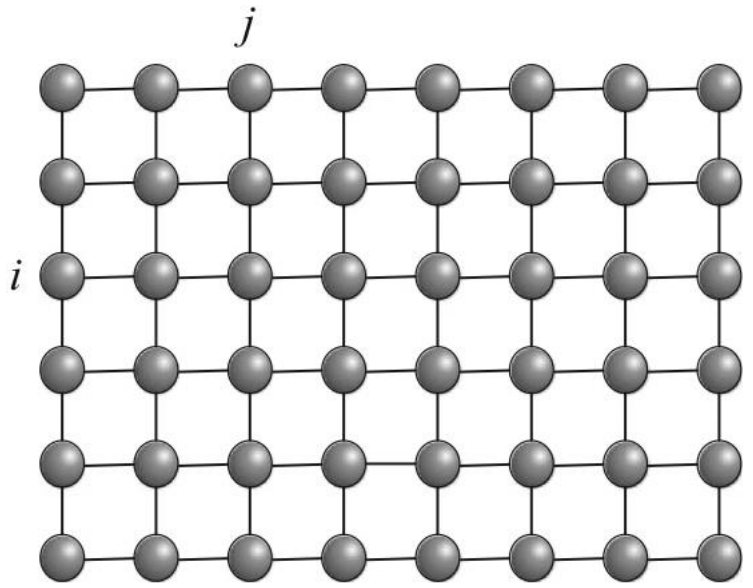
Anel

- Organização em que cada nó é conectado aos vizinhos.



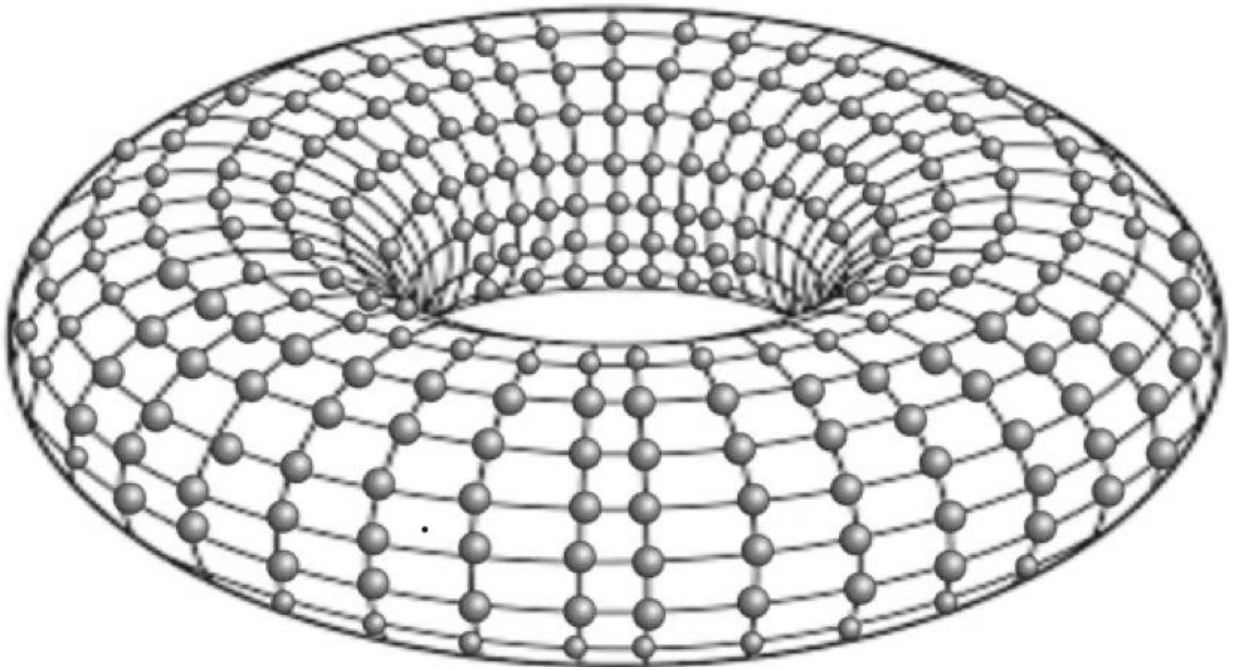
Mesh de 2 Dimensões

- *Switches* são organizados de forma retangular.
- Rótulo *(i, j)* nas dimensões X e Y.
- Cada *switch*, exceto os das bordas, tem 4 vizinhos.
- Conectados diretamente a cada *switch*, temos um banco de memória e uma unidade de processamento.

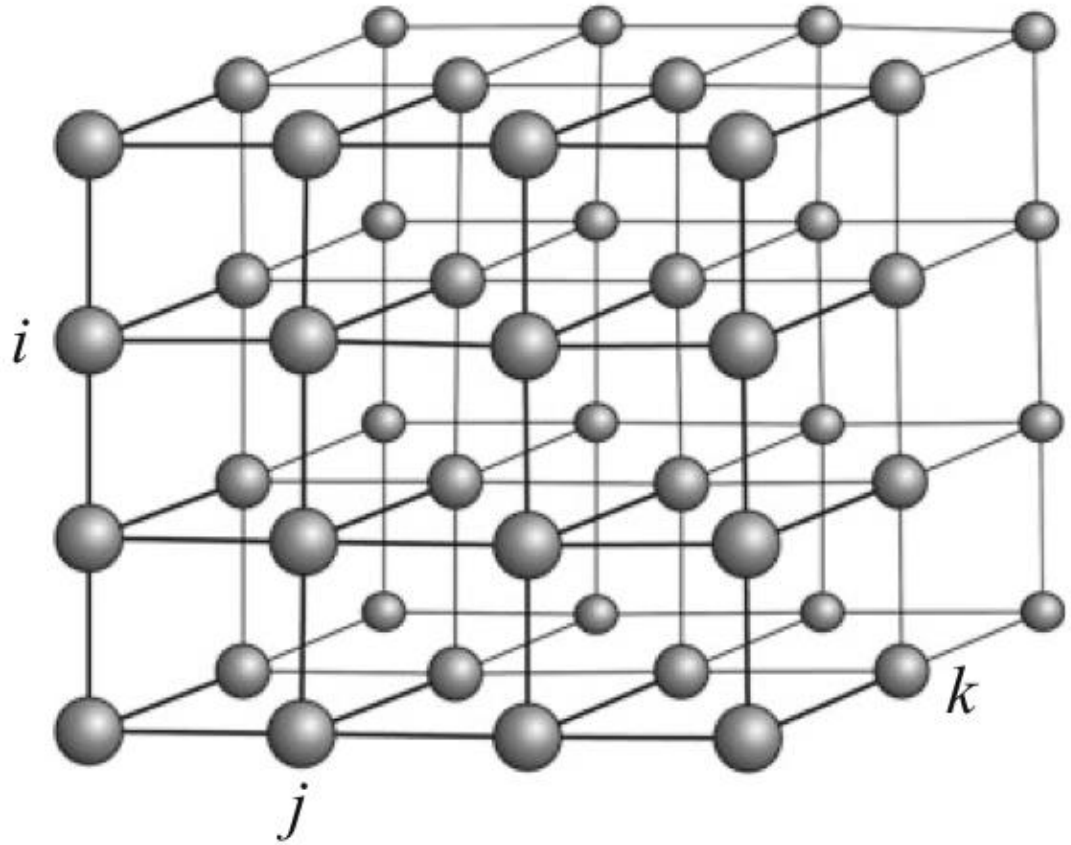


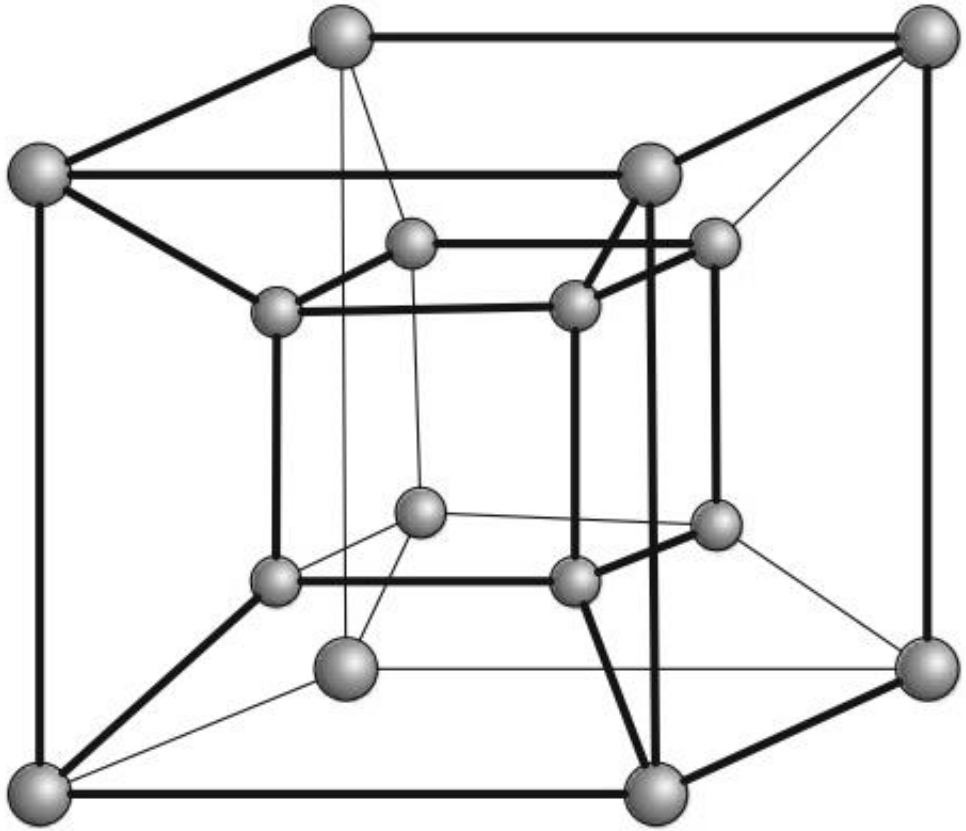
Torus de 2 Dimensões

- Igual ao *mesh*, mas os *switchs* das bordas são conectados.



Mesh e *Torus* de 3 Dimensões

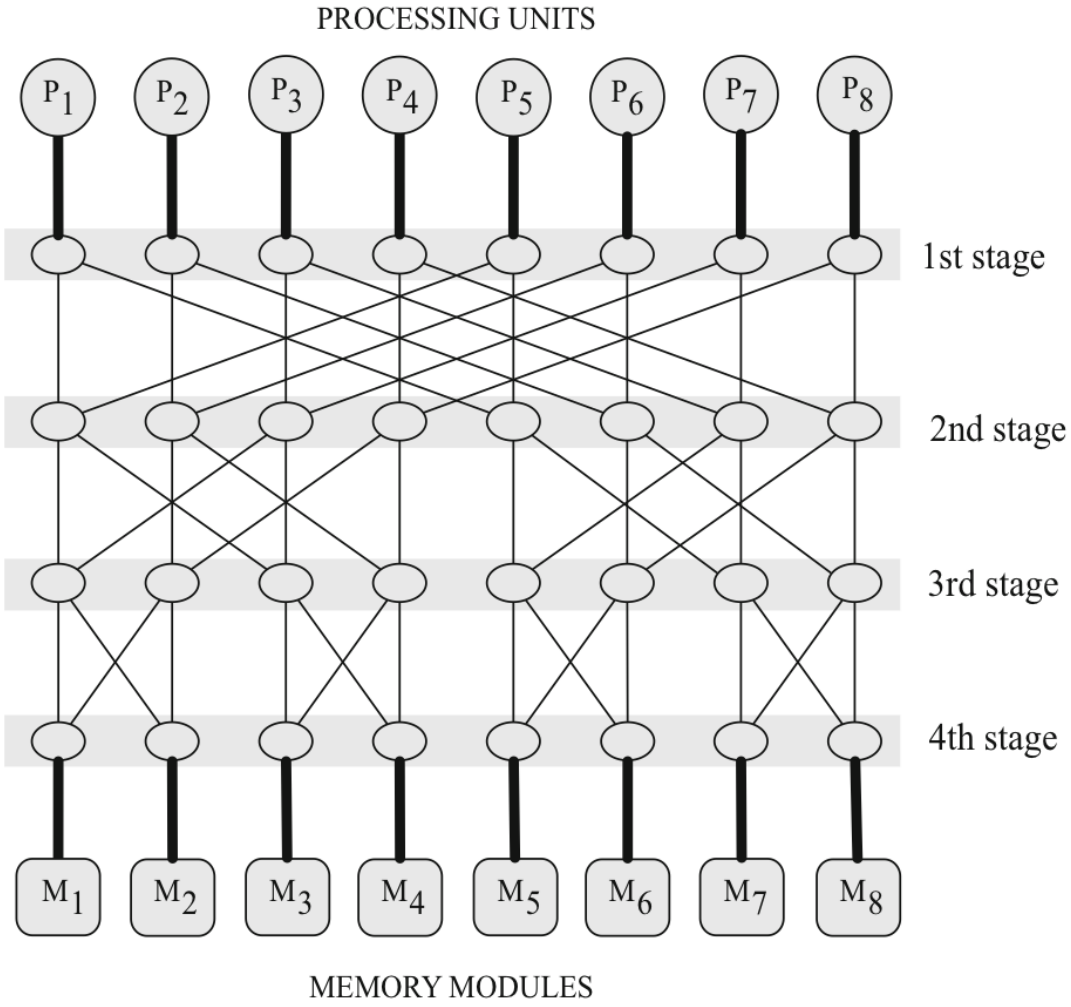


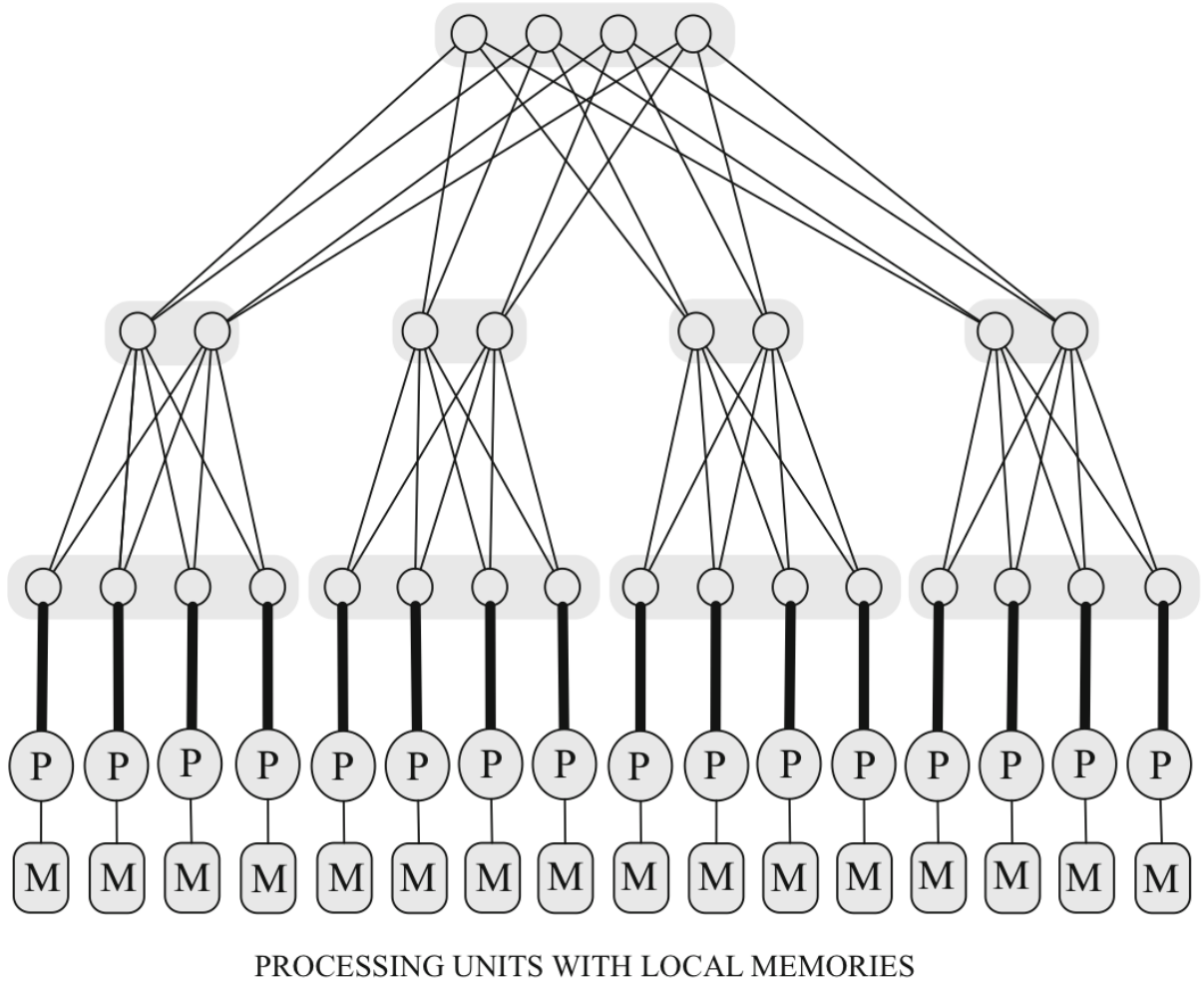


Hypercube

- Uma rede que tem $n = 2^b$ nós, para $b \geq 0$.
- Cada nó tem um rótulo distinto de *b bits*.
- Dois nós estão conectados se seus rótulos diferem entre si por um *bit*.
- Cada nó tem $b = log_2 n$ vizinhos.
- Os primeiros valores de *b* são fáceis, agora a medida que *b* cresce, a visualização se torna complexa.

Redes Multi-Estágio e *Fat Trees*





Complexidade de Computações Paralelas

- Novamente, seja Π um problema computacional.
  - Uma instância de Π é obtida pela “substituição” de suas variáveis de entrada por valores específicos de dados.
  - Podemos associar para cada instância π de Π um tamanho: *size*(π).
  - De maneira informal, temos *size*(π) como o espaço necessário na memória em um determinado computador.
- Por exemplo, considere Π = “ordenar uma sequência finita de números.”
  - π = “ordenar 0 9 2 7 4 5 6 3” é uma instância de Π com *size*(π) = 8.
- Se Π = “n é primo?”, e π = “17 é primo?”, qual *size*(π)?

Complexidade de Computações Paralelas

- Quando falamos em tempo de execução, consideramos as instâncias π do problema.
  - T(n): o tempo de execução de um algoritmo com entrada de tamanho *n*.
  - É interessante entender a taxa de crescimento de T(n).
  - Como o algoritmo se comporta para resolver problemas maiores.
    - T(n) = n, crescimento linear, instâncias duas vezes maiores levam duas vezes mais tempo.
    - T(n) = n^{const}, onde $const \geq 1$, temos crescimento polinomial.
    - T(n) = $2^{const,}$ crescimento exponencial.
- Ok, mas será que faz sentido aumentar o problema sem aumentar o número de unidades de processamento?

Complexidade de Computações Paralelas

- Temos que atualizar as definições de eficiência e speedup para o tamanho da instância.

$$S(n) \stackrel{\text{def}}{=} \frac{T_{seq}(n)}{T_{par}(n)} \qquad E(n) \stackrel{\text{def}}{=} \frac{S(n)}{p}$$

Quantas unidades de processamento *p* um computador *C(p)* deve ter para que em todas as instâncias de Π de tamanho *n*, o *speedup* do algoritmo P seja o máximo?

Complexidade de Computações Paralelas

- É natural imaginar que se desejamos aumentar *n*, devemos incrementar *p*.
  - $p = f(n)$ com $f(n) \leq f(n+1)$ para todo *n*.
  - Se $f(n)$ tiver crescimento exponencial:
    - Adicionar mais nós nas redes interconexões começa a ficar complicado.
    - O algoritmo terá limite de usabilidade.
  - Se $f(n)$ for polinomial:
    - Se o expoente não atingir valores absurdos (n^{100}), é uma alternativa viável.
    - O cálculo nos prova que funções polinomiais são dominadas por funções exponenciais.
- Devemos buscar soluções para Π de modo que o número de unidades de processamento necessárias para o *speedup* máximo cresça de maneira polinomial em relação ao tamanho do problema.

Definições

- **Definição 1**: Uma função é polilogaritmíca em *n* se é polinomial em log_n, ou seja, se é do formato $a_k(logn)^k + a_{k-1}(log_n)^{k-1} + \ \ldots + a_1(log_n)^1 + \ a_0$, para algum $k \ \geq 1$.
- Lembrando que não importa o modelo PRAM que escolhermos, o tempo de execução varia na ordem de $O(logp)$.
- **Definição 2**: NC é a classe de problemas computacionais solucionáveis em tempo polilogarítmico em uma máquina PRAM com número polinomial de unidades de processamento.
- NC é a classe de problemas eficientemente paralelizáveis.

Exemplo de Problema

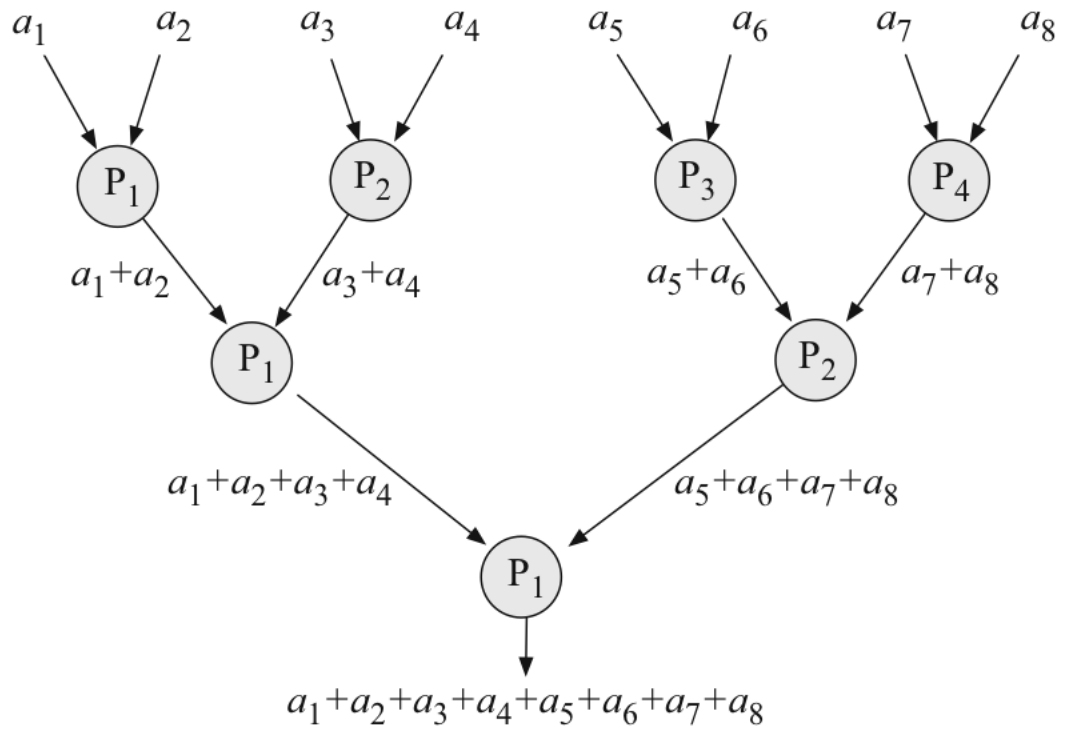
- Seja Π = “adicionar *n* números”.
  - Π = “adicionar 10 20 30 40 50 60 70 80” é uma instância *size*(π) = 8.

$T_{seq}(8) = 7$ etapas.
$T_{par}(8) = 3$ etapas, com p = 4
Em geral:

$$T_{par} = \lceil \log n \rceil = O(logn)$$

com $\left\lceil \frac{n}{2} \right\rceil = O(n)$ unidades de processamento,
levando um *speedup*:

$$S(n) = O(\frac{n}{logn})$$



Leis e Teoremas

- Teorema de Brent
  - Seja *M* uma máquina PRAM com *p* unidades de processamento.
  - Seja *W* o total de operações de operações executadas pelas *p* unidades de processamento de *M* para resolver um problema *P*.
  - $T_{par,M}(P)$ é o tempo de execução paralelo de *P* em *M*.
  - Suponha que consideremos *p* muito alto, vamos reduzi-lo para cotar custos.
  - A máquina resultante com *p* reduzido, chamada *R*, nos fornece $T_{par,R}(P)$.
  - Há grande possibilidade de $T_{par,R}(P) > T_{par,M}(P)$.
- Reduzir a quantidade de unidades de processamento irá aumentar o tempo de execução. Mas por quanto?

Leis e Teoremas

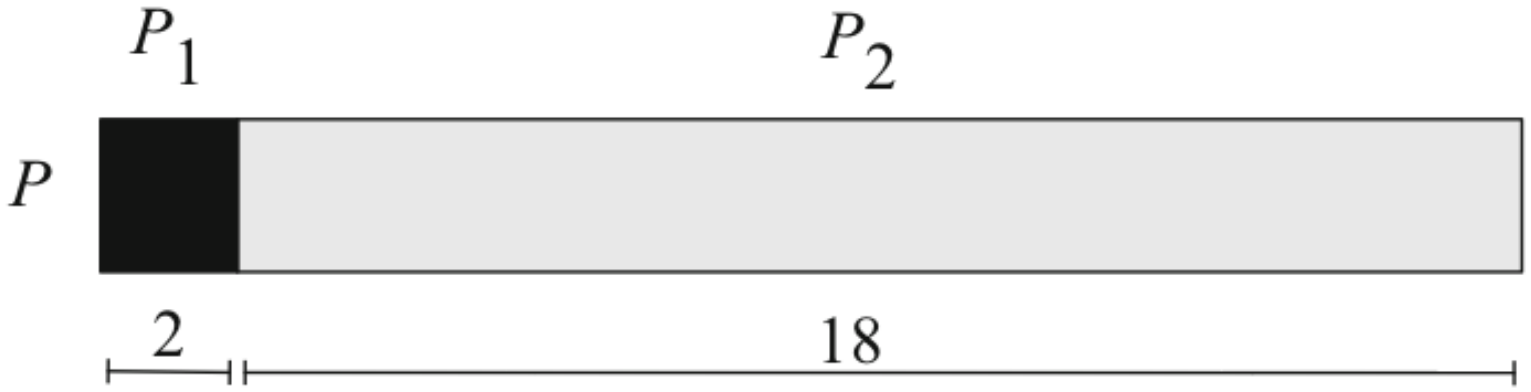
Teorema de Brent:

$$T_{par,R}(P) = O(\frac{w}{p} + T_{par,M}(P))$$

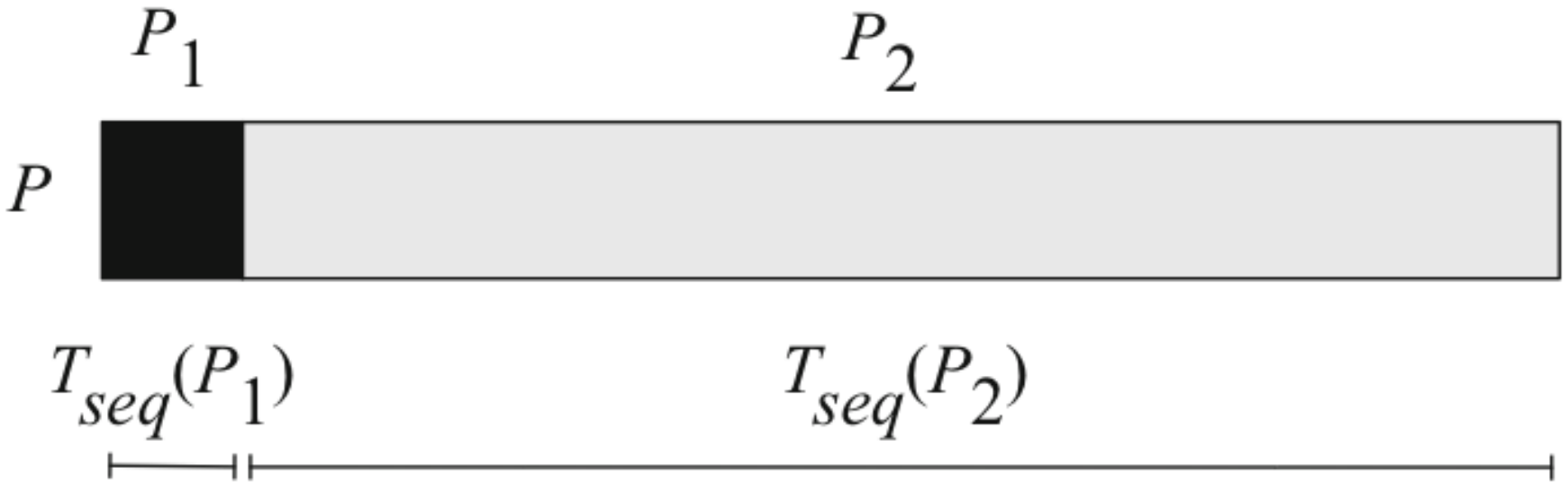
Como a máquina *R* tem menos unidades de processamento, para fazer o mesmo número de operações de *M*, mais etapas serão necessárias.

Lei de Amdahl

- Seja *P* um programa sequencial de processamento de arquivos:
  - *P* é na verdade, dois problemas, $P = P_1 P_2$.
  - P_1 faz a leitura de um diretório, gera a lista de arquivos e a submete para P_2.
  - P_2 faz o processamento de cada arquivo.
- P_1 não tem como ser aprimorado por adição de mais unidades de processamento, ao contrário de P_2.



Lei de Amdahl



O componente P_1 nunca executará mais rápido devido a paralelismo.

Lei de Amdahl

- $T_{seq}(P) = Ts_{eq}(P_1) + Ts_{eq}(P_2)$
- $T_{par}(P) = Ts_{eq}(P_1) + \frac{1}{s}T_{seq}(P_2)$, onde *s* é o *speedup* de P_2.
- Vamos definir *b* como a fração de P que é paralelizável:
  - $b = \frac{T_{seq}(P_2)}{T_{seq}(P)}$
- A fórmula do *speedup* original $S(P) = \frac{T_{seq}(P)}{T_{par}(P)}$ se transforma em:
  - $S(P) = \frac{1}{1-b+\frac{b}{s}}$

Lei de Amdahl

- Suponha que 70% de um programa seja paralelizável e executado em 16 unidades de processamento no lugar de uma.
  - Qual o *speedup* máximo atingível por todo o programa?
  - Se aumentarmos as unidades de processamento para 32, 64 e 128, o que acontece com esse limite?
  - Obs: considere que a parte paralela tem *speedup* linear.

FIM