Arquiteturas Paralelas

Prof. César A. F. De Rose

Pontifícia Universidade Católica do Rio Grande do Sul
Programa de Pós-Graduação em Ciência da Computação
Cx. Postal 1429 90619-900 Porto Alegre
E-mail: derose@inf.pucrs.br
http://www.inf.pucrs.br/~derose
http://www.cpad.pucrs.br

1 Introdução

Máquinas paralelas vem tornando-se mais populares em função da demanda sempre crescente por poder computacional. Infelizmente, os sistemas que oferecem a capacidade de processamento para satisfazer esta demanda, muitas vezes, ainda têm um custo muito elevado e/ou são difíceis de programar. O estudo de arquiteturas paralelas contribui para o entendimento e busca de alternativas para os dois problemas.
No caso do custo, um melhor entendimento das alternativas de construção de máquinas paralelas, e a compreensão de como estas decisões de projeto repercutirão no desempenho final da máquina, podem possibilitar a escolha de uma arquitetura de menor custo que ainda obtenha o desempenho desejado.
Como a programação de aplicações paralelas ainda exige o conhecimento de características específicas da máquina para a obtenção de desempenho, sólidos conhecimentos sobre a arquitetura de máquinas paralelas auxiliam em todo o ciclo de desenvolvimento de programas paralelos, desde sua modelagem até a fase de depuração e otimização.
Este curso tem por objetivo trabalhar os conceitos básicos e apresentar as principais tendências na área de arquitetura de máquinas paralelas. Para o aprofundamento dos conhecimentos nestas áreas são recomendados no texto, sempre que possível, materiais mais específicos.

2 Classificações de máquinas paralelas

2.1 Fluxo de instruções / fluxo de dados

Para uma classificação inicial de arquiteturas paralelas pode ser usada a classificação genérica de Flynn [5]. Apesar de ter sua origem em meados dos anos 70, é ainda válida e muito difundida. Baseando-se no fato de um computador executar uma seqüência de instruções sobre uma seqüência de dados, diferencia-se o fluxo de instruções (*instruction stream*) e o fluxo de dados (*data stream*). Dependendo se estes fluxos são múltiplos ou não, e através da combinação das possibilidades, Flynn propôs quatro classes (Tabela 1):

Tabela 1: **Classificação de Flynn segundo o fluxo de instruções e o fluxo de dados**

	SD (*Single Data*)	**MD** (*Multiple Data*)
SI (*Single Instruction*)	SISD Máquinas von-Neumann convencionais	SIMD Máquinas Vetoriais (Cray 1, CM-2, MasPar)
MI (*Multiple Instruction*)	MISD Sem representante (até agora)	MIMD Multiprocessadores e multicomputadores (CM-5, nCUBE, Intel Paragon, Cray T3D)

Na classe SISD (*Single Instruction Single Data*) um único fluxo de instruções atua sobre um único fluxo de dados. Na Figura 1 o fluxo de instruções (linha contínua) alimenta uma unidade de controle (C) que ativa a unidade central de processamento (P). A unidade P por sua vez atua sobre um único fluxo de dados (linha tracejada) que é lido, processado e reescrito na memória (M). Nesta classe são enquadradas as máquinas von-Neumann tradicionais com apenas um processador como microcomputadores pessoais e estações de trabalho.

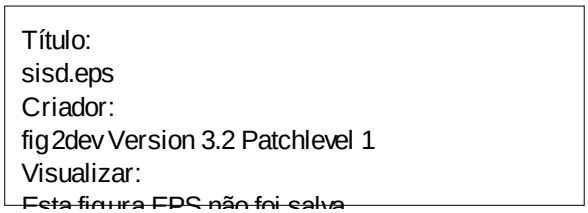

Figura 1: **Diagrama da classe SISD**

A classe MISD (*Multiple Instruction Single Data*) é bastante interessante. Neste caso, múltiplos fluxos de instruções atuariam sobre um único fluxo de dados. A Figura 2 mostra múltiplas unidades de processamento P, cada uma com sua unidade de controle própria C, recebendo um fluxo diferente de instruções. Estas unidades de processamento executam suas diferentes instruções sobre o mesmo fluxo de dados. Na prática,

diferentes instruções operam a mesma posição de memória ao mesmo tempo, executando instruções diferentes. Como isto até os dias de hoje não faz nenhum sentido, além de ser tecnicamente impraticável, esta classe é considerada vazia [8][1]. Na época o próprio Flynn não via utilidade para esta classe, mas resolveu manter todas as combinações possíveis na sua classificação.

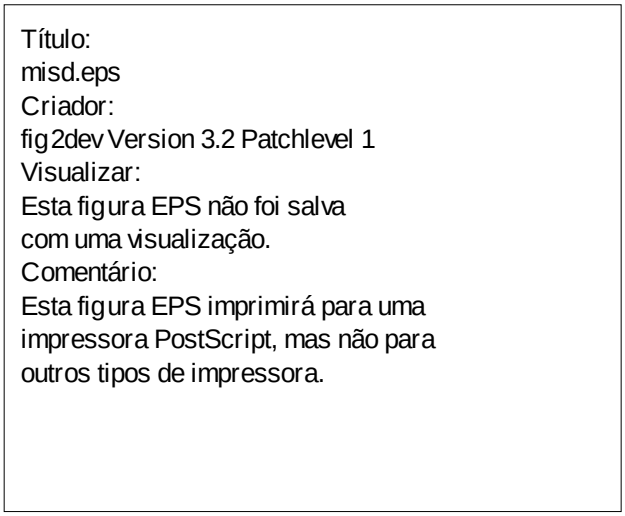

Figura 2: **Diagrama da classe MISD**

As máquinas paralelas se concentram nas duas classes restantes, SIMD e MIMD. No caso SIMD *(Single Instruction Multiple Data)* uma única instrução é executada ao mesmo tempo sobre múltiplos dados. O processamento é controlado por uma única unidade de controle C, alimentada por um único fluxo de instruções. A mesma instrução é enviada para os diversos processadores P envolvidos na execução. Todos os processadores executam suas instruções em paralelo de forma síncrona sobre diferentes fluxos de dados (Figura 3). Na prática pode se dizer que o mesmo programa está sendo executado sobre diferentes dados, o que faz com que o princípio de execução SIMD se assemelhe bastante ao paradigma de execução seqüencial. É importante ressaltar que para que o processamento das diferentes posições de memória possa ocorrer em paralelo, a unidade de memória M não pode ser implementada com um único módulo de memória, o que permitiria só uma operação por vez (ver seção 3.2). Nesta classe são enquadradas as máquinas Array e Vetorias como Cray 1 [9], CM-2 [10] e MasPar [10].

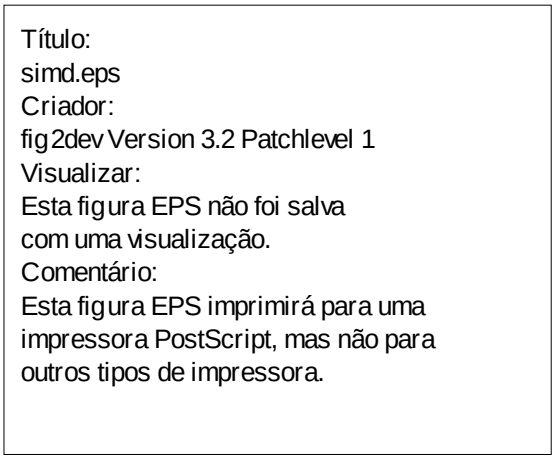

Figura 3: **Diagrama da classe SIMD**

Enquanto em uma máquina SIMD só um fluxo de instruções, ou seja só um programa está sendo executado, em uma máquina MIMD (*Multiple Instruction Multiple Data*) cada unidade de controle C recebe um fluxo de instruções próprio e o repassa para sua unidade processadora P para que seja executado sobre um fluxo de instruções próprio (Figura 4). Desta forma cada processador executa o seu próprio programa sobre seus próprios dados de forma assíncrona. Sendo assim o princípio MIMD é bastante genérico, pois qualquer grupo de máquinas, se analisado como uma unidade (executando por exemplo um sistema distribuído), pode ser considerado uma máquina MIMD. Neste caso, como na classe SIMD, para que o processamento das diferentes posições de memória possa ocorrer em paralelo, a unidade de memória M não pode ser implementada com um único módulo de memória, o que permitiria só uma operação por vez (ver seção 3.2). Nesta classe se enquadram servidores com múltiplos processadores (*dual*, *quad*), as redes de estações e máquinas como CM-5 [10], nCUBE [4], Intel Paragon [4], Cray T3D [4].

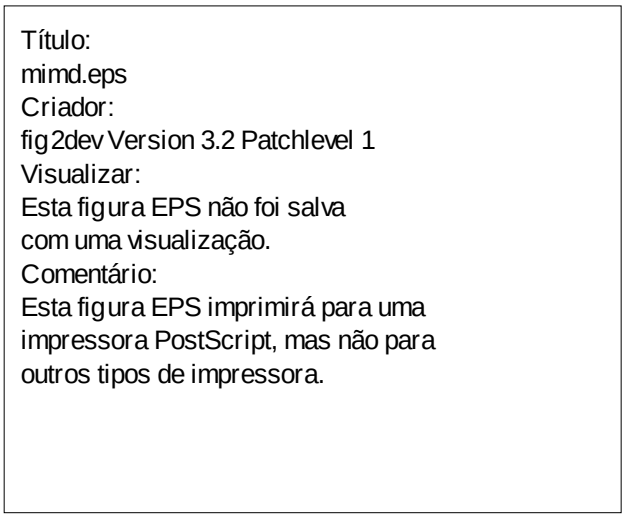

Figura 4: **Diagrama da classe MIMD**

2.2 Organização segundo o compartilhamento de memória

Um outro critério para a classificação de máquinas paralelas é o compartilhamento da memória. Quando se fala em **memória compartilhada** (*shared memory*) existe um único espaço de endereçamento que será usado de forma implícita para comunicação entre processadores, com operações de `load` e `store`. Quando a memória não é compartilhada existem **múltiplos espaços de endereçamento privados** (*multiple private address spaces*) um para cada processador. Isto implica em comunicação explícita, através de troca de mensagens com operações `send` e `receive`.

Memória distribuída (*distributed memory*) por sua vez, se refere a localização física da memória. Se a memória é implementada com vários módulos, e cada módulo foi colocado próximo de um processador, então a memória é considerada distribuída. Outra alternativa é o uso de uma **memória centralizada** (*centralized memory*) onde a memória se encontra a mesma distância de todos os processadores, independente se foi implementada com um ou com vários módulos [15]. A implementação de memória com vários módulos e seu endereçamento será detalhado na seção 3.

Dependendo se uma máquina paralela se utiliza de uma memória compartilhada por todos os processadores, ou não, podemos diferenciar:

- **Multiprocessadores**

  Todos os processadores *P* acessam através de uma rede de interconexão uma memória compartilhada *M*. Este tipo de máquina possui apenas um espaço de endereçamento, de forma que todos os processadores *P* são capazes de endereçar todas as memórias *M* (Figura 5). A comunicação entre processos é feita através da memória compartilhada de forma bastante eficiente com operações do tipo `load` e `store`. Estas características resultam do fato deste tipo de máquina

paralela ser construído a partir da replicação apenas do componente processador de uma arquitetura convencional (destacados em negrito na Figura 5). Daí o nome **múlti**plos **processadores**.

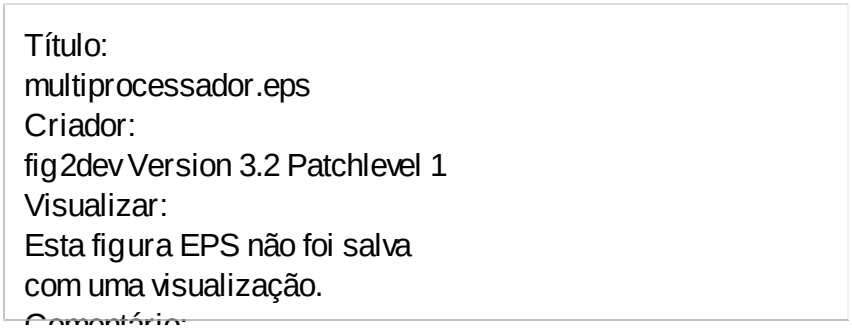

Figura 5: **Arquitetura de um multiprocessador**

Em relação ao tipo de acesso às memórias do sistema, multiprocessadores podem ser classificados como [10]:

- acesso uniforme à memória (*uniform memory access*, UMA)

  A memória usada nestas máquinas é centralizada e se encontra a mesma distância de todos os processadores (Figura 6), fazendo com que a latência de acesso a memória seja igual para todos os processadores do sistema (uniforme). Como o barramento é a rede de interconexão mais usada nestas máquinas, e suporta apenas uma transação por vez, como veremos na Seção 3.3, a memória principal é normalmente implementada com um único bloco. É importante ressaltar que máquinas com outras redes de interconexão e com memórias entrelaçadas (implementadas com múltiplos módulos e desta forma permitindo acesso paralelo em diferentes módulos - ver Seção 3) também se enquadram nesta categoria se mantiverem o tempo de acesso a memória uniforme para todos os processadores do sistema.

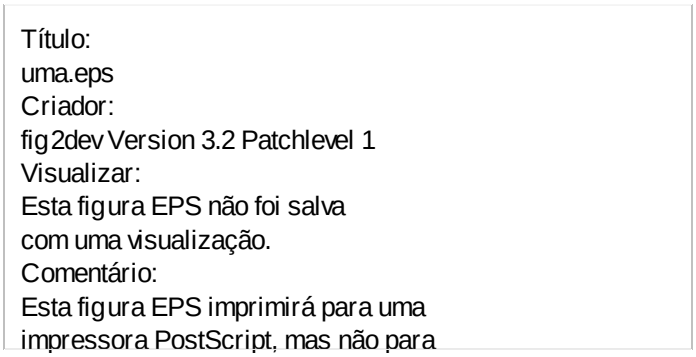

Figura 6: **Máquina UMA**

Muitas destas máquinas se utilizam de memórias cache para amortizar a diferença de velocidade entre processador e memória principal [15]. A memória cache é uma memória especial mais rápida que a memória principal e funciona como uma área intermediária de armazenamento. Para cada consulta feita à memória principal, é mantida uma cópia na cache, com o objetivo de acelerar um novo acesso ao mesmo endereço. Como nesta classe de máquinas todos os processadores podem endereçar toda a memória do sistema, em um determinado momento, várias cópias da mesma posição da memória principal podem existir em caches diferentes. Isto é um problema que precisa ser tratado, pois um processador pode trabalhar com sua cópia local e esta não refletir mais o estado atual da mesma posição na memória principal. Como o ideal é que o conteúdo das memórias caches sejam **coerentes**, este problema é chamado de coerência de cache (*cache coherence*) [4]. A maioria dos multiprocessadores UMA resolve este problema em hardware.

- acesso não uniforme à memória (*non-uniform memory access*, UMA)
  A memória usada nestas máquinas é distribuída, implementada com múltiplos módulos que são associados um a cada processador (Figura 7). O espaço de endereçamento é único e cada processador pode endereçar toda a memória do sistema. Se o endereço gerado pelo processador se encontrar no módulo de memória diretamente ligado a ele, dito local, o tempo de acesso será menor que a um módulo que está diretamente ligado a outro processador, dito remoto, que só pode ser acessado através da rede de interconexão. Por este motivo, estas máquinas possuem um acesso não uniforme à memória (a distância à memória não é sempre a mesma e depende do endereço desejado).

Título:
numa.eps
Criador:
fig2dev Version 3.2 Patchlevel 1
Visualizar:
Esta figura EPS não foi salva
com uma visualização.
Comentário:

Figura 7: Máquina NUMA

Dependendo se o problema da coerência de cache foi tratado, e se este tratamento foi feito em hardware ou em software esta classe pode ser sub-dividida em:

– acesso não uniforme à memória sem coerência de cache (*non-cache-coherent non-uniform memory access*, NCC-NUMA)

Variação de NUMA onde não tenho coerência de cache garantida em hardware.

- acesso não uniforme à memória com coerência de cache (*cache-coherent non-uniform memory access*, CC-NUMA)
  Variação de CC-NUMA onde tenho coerência de cache garantida em hardware.
- acesso não uniforme à memória com coerência de cache em software (*software-coherent non-uniform memory access*, SC-NUMA)
  Neste caso a coerência de cache não está implementada em hardware como nas máquinas CC-NUMA mas em software, de forma transparente ao usuário. Esta camada de software é também conhecida como DSM (*Distributed Shared Memory*) [14] e naturalmente só faz sentido sobre máquinas NCC-NUMA e NORMA que não tem coerência de cache em hardware.
  Uma DSM depende de extensões de software para a obtenção de um espaço de endereçamento único, compartilhamento de dados, e controle de coerência. Uma alternativa de implementação é chamada SVM (*Shared Virtual Memory*) onde o mecanismo de gerenciamento de memória de um sistema operacional tradicional é modificado para suportar estes serviços em nível de páginas ou segmentos. A vantagem é que desta forma não se fazem necessárias alterações nas aplicações. O projeto SHRIMP [13] usa esta abordagem.
  Outra possibilidade é não alterar o sistema operacional e usar compiladores e bibliotecas de funções para converter código desenvolvido para um espaço de endereçamento único em um código que roda em múltiplos espaços de endereçamento. Neste caso o código original tem que ser modificado para incluir compartilhamento de dados, sincronização e primitivas de coerência. O projeto TreadMarks [2] usa esta abordagem.
- arquiteturas de memória somente com cache (*cache-only memory architecture*, COMA)
  Em uma máquina COMA todas as memórias locais estão estruturadas como memórias cache e são chamadas de COMA caches [11] (Figura 8). Estas caches tem muito mais capacidade que uma cache tradicional. Arquiteturas COMA são as únicas que tem suporte de hardware para a replicação efetiva do mesmo bloco de cache em múltiplos nós (nas arquiteturas anteriores os blocos são invalidados e não atualizados). A memória principal destas máquinas é composta pelas caches COMA e o hardware de suporte tem que integrar a gerência das caches e a gerência de memória. Esta complexidade faz com que estas arquiteturas sejam mais caras de implementar que máquinas NUMA.

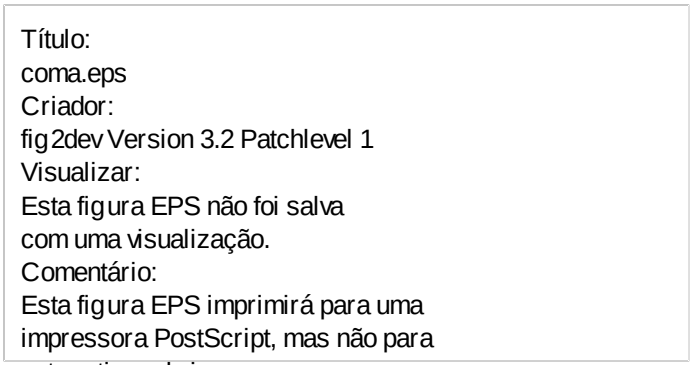

Figura 8: **Máquina COMA**

- **Multicomputadores**

Cada processador *P* possui uma memória local *M*, à qual só ele tem acesso. As memórias dos outros processadores são consideradas memórias remotas e possuem espaços de endereçamento distintos (um endereço gerado por P_1 só é capaz de endereçar M_1). Como não é possível o uso de variáveis compartilhadas neste ambiente, a troca de informações com outros processos é feita por envio de mensagens pela rede de interconexão (Figura 9). Por esta razão estas máquinas também são chamadas de sistemas de troca de mensagens (*message passing systems*). Estas características resultam do fato deste tipo de máquina paralela ser construído a partir da replicação de toda a arquitetura convencional e não apenas do componente processador como nos multiprocessadores (destacados em negrito na Figura 9). Daí o nome **múlti**plos **computadores**.

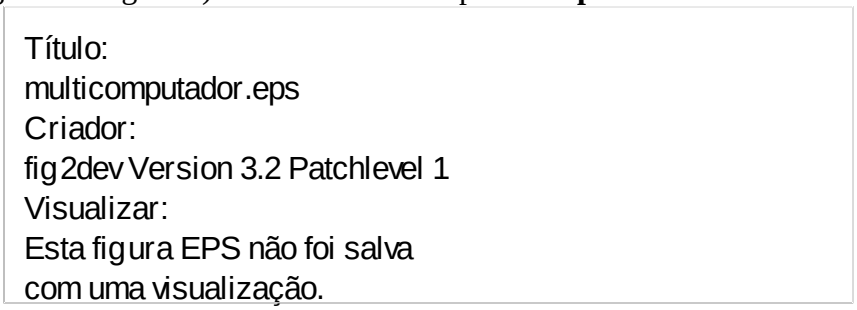

Figura 9: Arquitetura de um multicomputador

Em relação ao tipo de acesso às memórias do sistema, multicomputadores podem ser classificados como:

- sem acesso a variáveis remotas (*non-remote memory access*, NORMA)

  Como uma arquitetura tradicional inteira foi replicada na construção destas máquinas, os registradores de endereçamento de cada nó só conseguem endereçar a sua memória local.

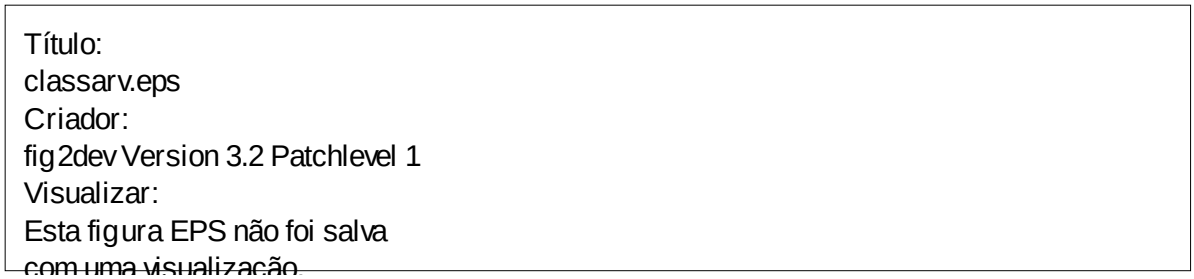

Figura 10: **Visão geral das classificações vistas [11]**

3 Organização da memória principal

Em uma máquina paralela a memória principal é um recurso que tem um papel fundamental, como vimos na seção de classificações, especialmente em multiprocessadores onde é compartilhada por todos os processadores. É necessário que se tenha cuidado na escolha do tipo de organização de memória para que se evite um drástica degradação de desempenho causada por dois ou mais processadores tentando acessar os mesmos módulos do sistema de memória. Sendo assim não é desejável que a memória principal seja implementada por um módulo único de forma monolítica mas sim particionada em vários módulos independentes com um espaço de endereçamento único distribuído entre estes módulos. Esta forma de implementação é chamada de entrelaçamento (*interleaving*) e atenua a interferência entre processadores no acesso à memória permitindo acessos concorrentes em diferentes módulos (Figura 11b). O entrelaçamento de endereços entre M módulos de memória é chamado de entrelaçamento de M-vias.

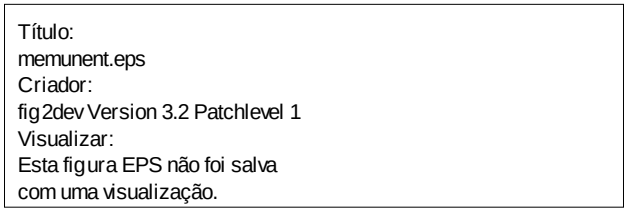

Figura 11: **Memória com bloco único e entrelaçada**

3.1 Formas de endereçamento de uma memória entrelaçada

Existem dois métodos básicos de se distribuir o espaço de endereçamento em uma memória entrelaçada. Assumindo uma memória principal com $N = 2^n$ palavras e conseqüentemente um endereço físico de cada uma destas palavras com n bits, $a_{n-1}\ a_{n-2}\ \ldots\ a_1\ a_0$ respectivamente. O método, denominado entrelaçamento mais significativo (*high-order interleaving)*, distribui os endereços entre os $M = 2^m$ módulos de memória de forma que cada módulo i , para $0 \leq i \leq M - 1$, contenha endereços consecutivos de $i2^{n-m}$ até $(i+1)2^{n-m}-1$ inclusive. Os m bits mais significativos são usados para selecionar o módulo, enquanto os n-m bits restantes selecionam a palavra dentro do módulo (Figura

12). Com esta atribuição de endereços, palavras consecutivas são armazenadas no mesmo módulo.

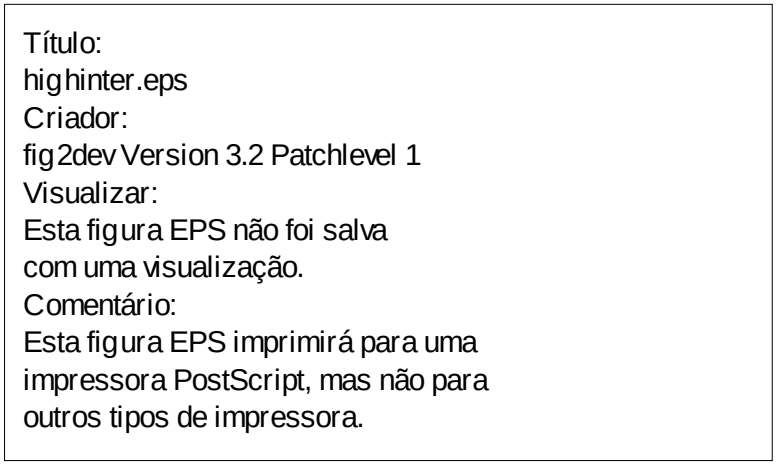

Figura 12: **Entrelaçamento mais significativo com endereços consecutivos no mesmo módulo**

O segundo método, entrelaçamento menos significativo (*low-order interleaving*) distribui os endereços de forma que endereços consecutivos sejam atribuídos a consecutivos módulos de memória. Desta forma os *m* bits menos significativos do endereço selecionam o módulo, enquanto os *n-m* bits restantes selecionam a palavra dentro do módulo (Figura 13). Neste método um endereço A é atribuído ao módulo $A \bmod M$ (resto da divisão inteira de *A* por *M*), ou seja, palavras consecutivas são armazenadas em módulos consecutivos.

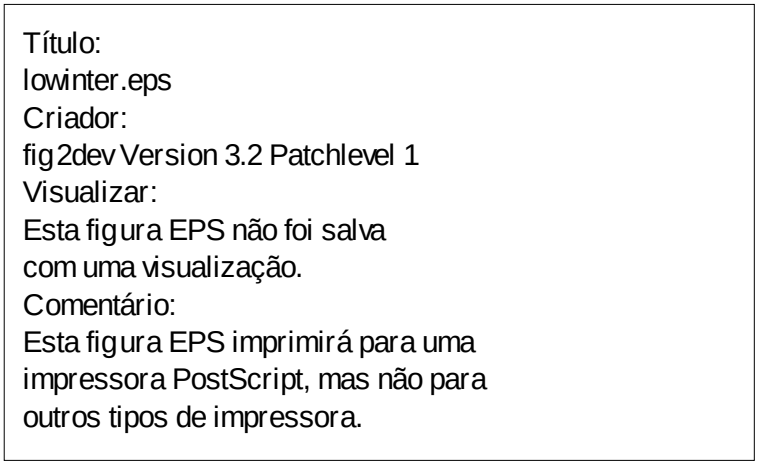

Figura 13: **Entrelaçamento menos significativo com endereços consecutivos em módulos consecutivos**

Os métodos vistos acima representam dois extremos na distribuição de endereços. As principais vantagens do entrelaçamento mais significativo são a fácil expansão da

memória e a melhor capacidade de tolerar falhas. Se necessário, a memória pode ser facilmente expandida com a adição de mais módulos até o limite de *M*-1. Além disto, a confiabilidade da memória principal aumenta, pois a falha de um dos módulos só afeta uma parte localizada da memória. Porém, a atribuição de endereços contíguos em um mesmo módulo pode causar conflitos no acesso à memória no caso de máquinas *pipeline*, vetoriais ou multiprocessadores SIMD. Nestes casos a seqüência de instruções de um programa ou uma seqüência de dados em um vetor resultaria no armazenamento no mesmo módulo. Como o ciclo da memória é muito maior que o ciclo do pipeline do processador, uma nova requisição à memória seria feita antes da anterior ter sido atendida, atrasando a execução de instruções. No caso de multiprocessadores SIMD, se vários elementos de um vetor de dados estiverem armazenados em um mesmo módulo, o processamento dos dados em paralelo por parte dos múltiplos processadores será impossibilitado, porque os elementos não conseguem ser acessados simultaneamente na memória.

O método de entrelaçamento menos significativo por sua vez é muito usado quando se deseja reduzir o conflitos de acesso a memória principal em multiprocessadores. Como os endereços contíguos estão localizados em módulos diferentes, os problemas de conflito que ocorrem no método mais significativo são amenizados neste caso. Porém, uma grande desvantagem é que a falha de um módulo compromete toda a memória principal e não apenas uma parte isolada da memória, devido a distribuição dos endereços.

Uma forma de amenizar as desvantagens de ambos os métodos acima é a utilização de um método de entrelaçamento que pode ser visto como um compromisso entre os dois extremos vistos acima. A parte do endereço que seleciona o módulo é subdividida em duas seções, S_{m-r} e S_r de forma que S_r sejam os r bits menos significativos do endereço da memória e S_{m-r} os $m-r$ bits mais significativos do endereço. Desta forma o endereço do módulo é composto pela concatenação das seções S_{m-r} e S_r (Figura 14). Neste método os endereços são distribuídos em grupos de 2^r módulos de memória. Isto tende reduzir a interferência entre acessos à memória a um segmento de memória compartilhada. A memória pode ser expandida em blocos de 2^r módulos e a falha em um dos módulos compromete um bloco de 2^r módulos.

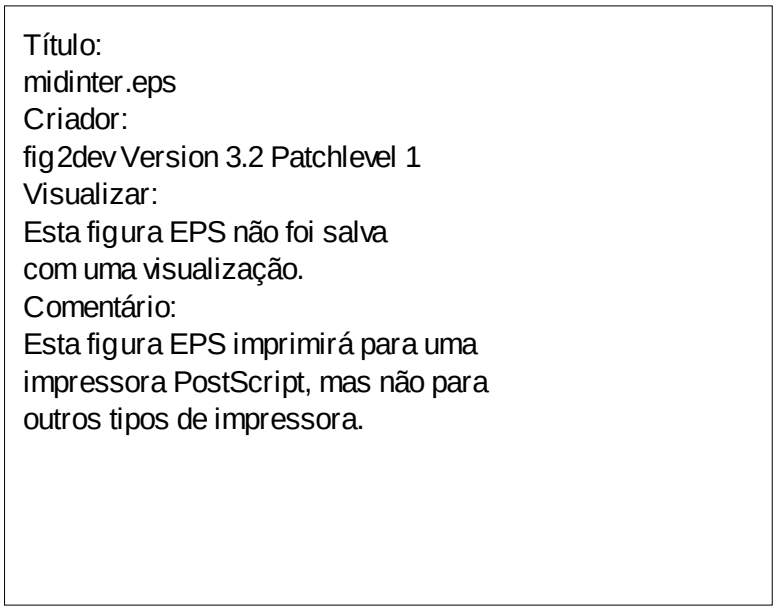

Figura 14: Compromisso de entrelaçamento com endereços distribuídos em grupos de 2^r módulos

Uma grande vantagem desta técnica é sua parametrização através da escolha do tamanho de *r*. Um *r* pequeno aumenta a interferência no acesso e melhora a tolerância a falhas da memória principal. Um *r* grande tem o efeito inverso, diminuindo a interferência no acesso e comprometendo a capacidade por parte da memória principal de tolerar falhas de módulos.

3.2 Memórias com múltiplas portas

É importante destacar que de nada adianta a quebra da memória principal em vários módulos se a rede de interconexão utilizada na máquina não suporta múltiplas transações. Uma máquina UMA, por exemplo, que se utilize de um barramento para ligar os processadores a memória principal, não se beneficia dos múltiplos canais de uma memória entrelaçada, já que o próprio barramento é o gargalo.

A utilização de redes de interconexão que possam se aproveitar dos múltiplos canais de memórias entrelaçadas pode, por sua vez, ter um custo muito alto, como por exemplo, matrizes chaveadoras para um elevado número de processadores.

Uma alternativa é a utilização de memórias **multiporta**. A idéia é mover a lógica de arbitragem e de chaveamento da rede de interconexão para dentro do controlador de memória. Com a utilização deste tipo de memória os processadores do sistema são ligados diretamente na memória principal, sem a necessidade de uma rede de interconexão.

Desta forma, porém, os módulos da memória ficam mais caros devido a adição de múltiplas portas de acesso e a lógica associada. A Figura 15 demonstra a ligação de *n* processadores em *m* blocos de uma memória entrelaçada multiporta. Cada um dos *m* módulos de memória só pode atender um processador de cada vez.

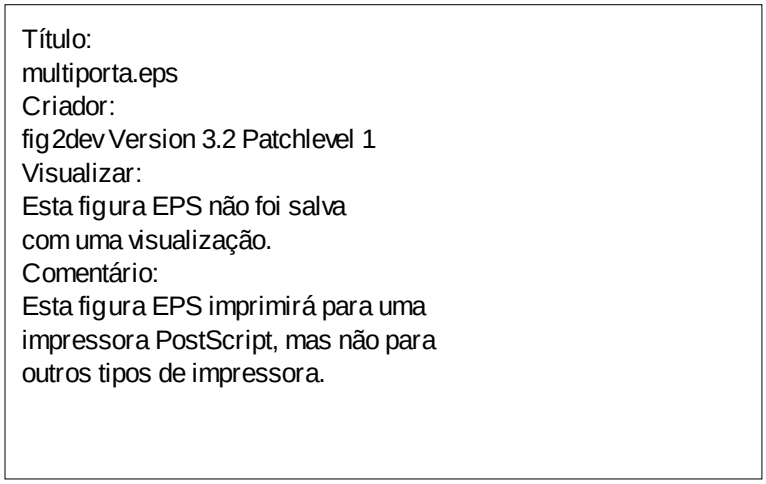

Figura 15: **Memória multiporta com *n* processadores e *m* módulos de memória**

A utilização de uma memória principal com múltiplas portas é um compromisso entre a utilização de um rede de interconexão de baixo custo e baixa performance como o barramento e a utilização de uma rede de interconexão de alto custo e alta performance como uma matriz chaveadora.

As principais desvantagens de uma memória multiporta são o alto custo para valores elevados de *n* e *m* e a impossibilidade de expansão do número de processadores depois de estabelecido o número de portas da memória. O alto custo para um grande número de componentes pode ser amenizado com a implementação de uma memória multiporta que não tenha a ligação de todos os processadores para todos os módulos. A Figura 16 apresenta esta alternativa com alguns dos módulos ligados apenas a um dos processadores do sistema.

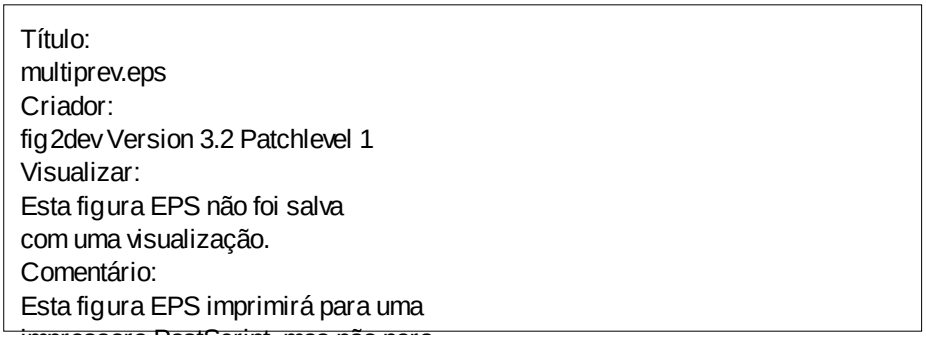

Figura 16: **Memória multiporta com módulos dedicados a certos processadores**

4 Redes de Interconexão

A forma como os processadores das arquiteturas vistas acima são ligados entre si e com

outros componentes do sistema (normalmente com a(s) memória(s)) é dada pela rede de interconexão. Nas duas principais classes de arquiteturas vistas acima, a rede de interconexão desempenha um papel muito importante: nos multiprocessadores ela pode ajudar a amenizar o problema dos conflitos de acesso, e em multicomputadores ela influencia diretamente a eficiência da troca de informações. Para avaliação das diferentes redes de interconexão serão utilizados os seguintes critérios [7]:

- Escalabilidade

  Capacidade de adaptação às necessidades do usuário. No caso das redes de interconexão refere-se à capacidade da rede de interligar componentes adicionais, por exemplo caso o usuário necessite mais desempenho, mantendo as características originais.

- Desempenho
  Indica a capacidade e a velocidade da transferência dos dados na rede. É influenciado por vários fatores como: desempenho físico das ligações utilizadas na rede, as distâncias a serem percorridas e o grau de paralelismo na transferência (quantos nós podem transferir dados simultaneamente). Os principais indicadores de desempenho em redes de interconexão são a *latência* e a *vazão*. A latência indica o tempo que uma unidade de dados demora para ser transferida e a vazão quantas unidades de dados foram transferidas por unidade de tempo. Um carro com capacidade para 5 passageiros que leve 10 minutos para chegar a seu destino tem uma latência de 10 minutos e uma vazão de 30 passageiros/hora. Outro fator a ser considerado é se as ligações são unidirecionais ou bidirecionais, sendo o último capaz de transferir dados simultaneamente em ambas as direções (*full-duplex*) ou não (*half-duplex*).

- Custo
  No caso das redes de interconexão, o custo cresce proporcionalmente em função número de ligações e sua capacidade de transferência (vazão e latência).
- Confiabilidade
  Existência de caminhos alternativos redundantes entre componentes, que aumentam a tolerância da rede de interconexão em caso de falhas.
- Funcionalidade
  Juntamente com a função de transferência de dados propriamente dita, as redes de interconexão podem eventualmente implementar outros serviços, como por exemplo armazenamento temporário dos dados transmitidos (como um *buffer*), garantia de ordenação dos dados transmitidos (quem foi enviado primeiro chega primeiro) ou roteamento automático implementado em hardware.

4.1 Redes estáticas

Se os componentes da máquina (processadores, memórias) estão interligados através de ligações fixas, de forma que entre cada dois componentes exista uma ligação direta dedicada, a rede de interconexão é denominada *estática* (ponto-a-ponto). A Figura 17 apresenta alguns exemplos de redes estáticas. Redes estáticas são utilizadas na maioria dos casos em multicomputadores. Neste caso a *topologia* (estrutura da interligação) determina as características da rede. Máquinas paralelas possuem quase sempre estruturas regulares com ligações homogêneas enquanto sistemas distribuídos, por causa da posição geográfica a sua integração gradual, possuem estruturas irregulares com ligações heterogêneas (ligações heterogêneas entre redes locais).

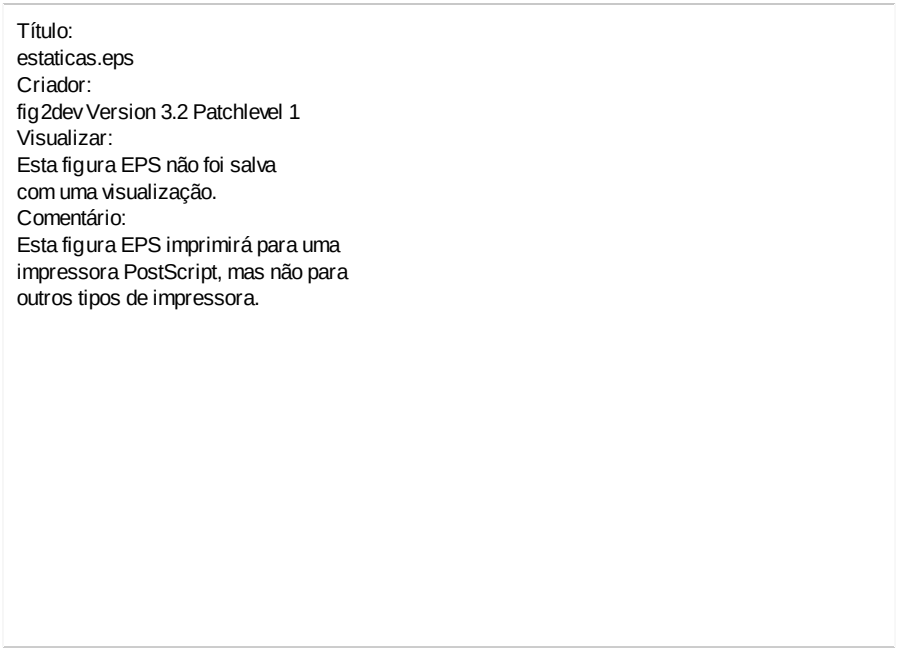

Figura 17: **Algumas topologias estáticas**

Os principais critérios para a avaliação de uma topologia são o número total de ligações entre componentes, quantas ligações diretas cada componente possui (*grau do nó*) e a maior distância entre dois componentes quaisquer da rede (*diâmetro*). O grau do nó pode ser constante ou variar de acordo com o número total de nós da rede (por exemplo em uma rede em forma de estrela). Para cada ligação que um componente possui é necessária uma interface física correspondente (porta) o que aumenta consideravelmente o custo do componente para muitas ligações. Em relação ao custo, o melhor caso seria uma rede com um grau de nó baixo e constante. Um processador *Transputer* [12], por exemplo, possui capacidade para a ligação direta de 4 vizinhos, não podendo ser usado em topologias que necessitem um grau de nó maior do que 4 (por exemplo um Hipercubo de grau 5). A Tabela 2 apresenta uma comparação dos critérios acima para as topologias estáticas da Figura 17.

Tabela 2: **Comparação entre critérios das topologias**

	Número de ligações	Grau do nó	Diâmetro
Anel	1	1	log *n*
Anel Chordal	1/*n*	1	1
Estrela	*n*	n^2	*n* log *n*
Totalmente Conectada	*n*	*n*	*n* log *n*

A estrutura com a menor conectividade (relação entre o número ligações e o número de nós) é o anel (Figura 17a). Com um grau de nó constante igual a 2 o seu custo é baixo, mas seu diâmetro cresce de forma linear em relação ao número total de nós. Se todos os processadores necessitarem trocar dados entre si, pode ocorrer uma sobrecarga do anel, o que acarretaria em atraso na transferência dos dados. Outra desvantagem do anel é a falta de caminhos alternativos entre os nós, o que resulta em uma baixa confiabilidade. Para diminuir o tráfego no anel principal e aumentar a sua confiabilidade podem ser incluídos caminhos adicionais. A topologia resultante é denominada anel *chordal* (Figura 17b). O outro extremo em relação ao anel é a topologia totalmente conectada (Figura 17c), com um grau de nó igual ao número de nós, um crescimento quadrático de número de conexões em relação ao número de nós, mas com o diâmetro ideal de 1.

Outro fator que pode ser decisivo na escolha de uma topologia, além da relação custo/desempenho, é sua adequação a uma classe específica de algoritmos. No caso ideal, o padrão de interconexão da topologia corresponde exatamente ao padrão de comunicação da aplicação paralela que executa na máquina. A *árvore binária* (Figura 18a) por exemplo, favorece a execução de algoritmos de divisão e conquista (*divide and conquer* [17]). O seu diâmetro cresce de forma linear em relação a altura *h* da árvore *A(h)* e de forma logarítmica em relação ao número de nós. Outras características das árvores binárias são o seu grau de nó máximo de 3 (a raiz tem grau igual a 2) e sua baixa confiabilidade, já que a falha de um nó resulta na perda da ligação com toda a sub-árvore abaixo dele (particionamento da estrutura). Mesmo que não ocorram falhas, o uso de árvores pode ser tornar problemático, pois todo o fluxo de dados entre a sub-árvore esquerda e a direita tem que passar pela raiz, que se torna rapidamente o gargalo da rede.

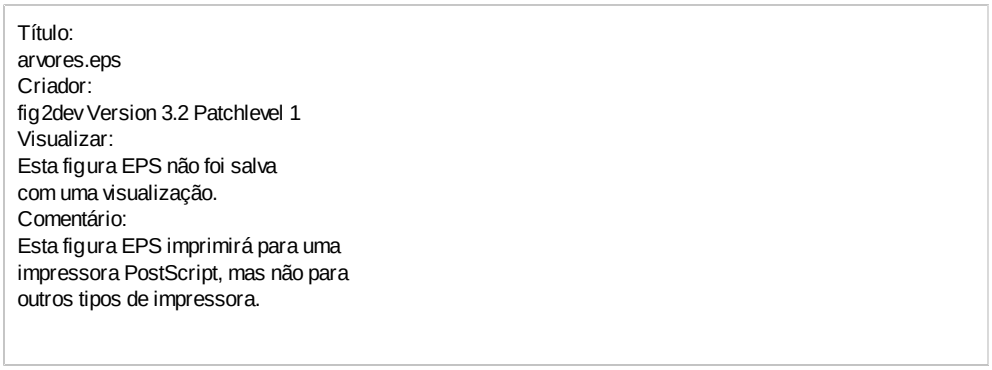

Figura 18: **Árvore Binária e *X-Tree***

Esta deficiência pode ser aliviada com uma árvore *X-tree X(h)* com altura *h*, que além das conexões normais da árvore binária adiciona conexões entre os nós do mesmo nível (Figura 18b). O grau do nó máximo é 5, mas a falha de qualquer nó não resulta em particionamento da estrutura.

Uma topologia muito utilizada em máquinas paralelas é a *malha* bidimensional *M(x,y)*, podendo ter as bordas não conectadas (Figura 19a) ou com bordas conectadas de forma cíclica, formando um *Torus* bidimensional (Figura 19b). Esta topologia também é conhecida como rede NEWS, por causa das conexões para os vizinhos nas quatro coordenadas (*North, East, West, South*).

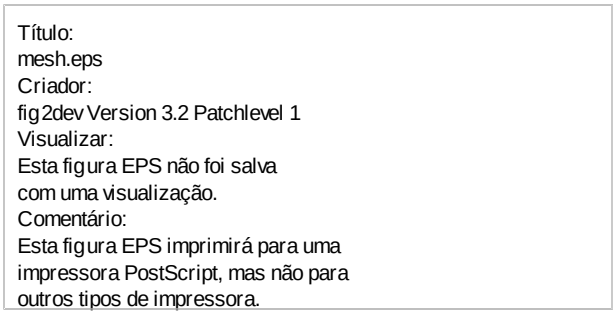

Figura 19: **Malha e torus**

O grau do nó é constante e igual a 4 (se não forem consideradas as bordas da malha) possibilitando facilmente um aumento do número de processadores em qualquer uma das dimensões (inclusão de linhas ou de colunas) e resultando em uma boa escalabilidade. Malhas não precisam necessariamente serem quadradas como na Figura 19, podendo uma das dimensões ser menor do que a outra, resultando em diferentes retângulos (o que pode se vantajoso no caso da escalabilidade, pois permite o aumento do número de processadores em múltiplos da menor dimensão). O diâmetro da malha cresce proporcional à raiz quadrada do número de nós e a existência de caminhos alternativos entre nós aumenta a confiabilidade da rede e diminui o risco de gargalos. Malhas são adequadas aos problemas nos quais uma estrutura de dados bidimensional

tem que ser processada de forma particionada. Muitos problemas atuais, que necessitam grande poder de processamento, possuem estas características, como por exemplo, operações com matrizes, processamento de imagens e equações diferenciais [6]. Malhas também podem ter mais de duas dimensões (*d >2*). O grau do nó é nestes casos *2d*. Uma atenção especial vem sendo dada às estruturas tridimensionais pois cada vez mais aplicações que necessitam alto desempenho modelam aspectos físicos do nosso mundo tridimensional. Alguns exemplos são a previsão do tempo, simulação de partículas e aerodinâmica. Todas estas aplicações se baseiam na divisão do espaço tridimensional em quadrantes e no cálculo de grandezas físicas dentro destes quadrantes. O cálculo dos diferentes quadrantes pode ocorrer em grande parte em paralelo, pois cada quadrante necessita apenas de dados das margens dos quadrantes que compõem sua fronteira. A comunicação está desta forma restrita às margens dos quadrantes e este padrão de comunicação corresponde às ligações de uma malha tridimensional *M(x,y,z)*.
Outra topologia muito difundida entre as máquinas paralelas é o *Hipercubo*, principalmente devido o seu pequeno diâmetro, que é igual ao seu grau e cresce de forma logarítmica em relação ao número de nós. Devido a estas características o hipercubo se adapta bem à aplicações com padrões de comunicação pouco localizadas (não restrita a regiões vizinhas). A Figura 20 apresenta dois hipercubos com a indicação de seu grau, número de nós e do maior caminho a ser percorrido entre nós (diâmetro).

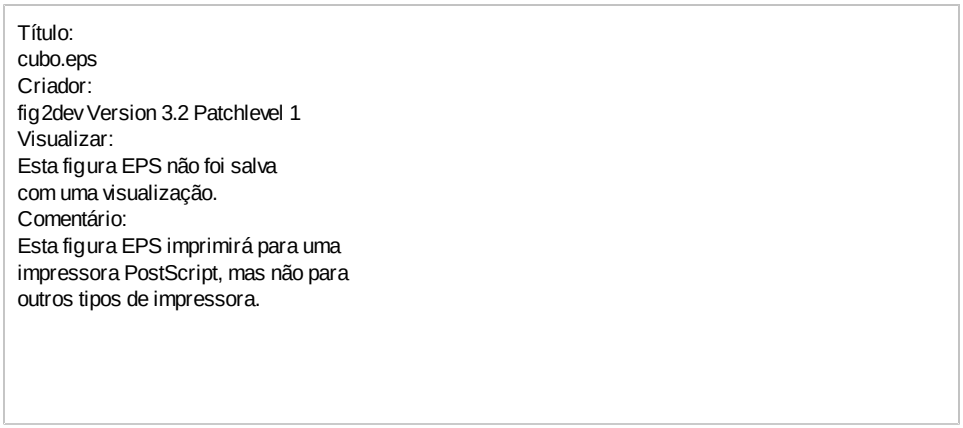

Figura 20: **Hipercubos de diferentes dimensões (grau)**

Por causa de sua estrutura o hipercubo não é muito flexível no que tange a sua escalabilidade, só podendo ter o seu número de nós aumentado em intervalos de potência de dois. A sua principal desvantagem porém, é o seu grau de nó que é sempre igual a sua dimensão. Isto dificulta a construção de hipercubos de maior dimensão. Com o grau 4 de um *Transputer* por exemplo (cada processador possui apenas 4 links de comunicação), seria apenas possível a construção de um hipercubo de dimensão 4 com 16 processadores (número de processadores = $2^{\text{grau do cubo}}$). Como esta é uma limitação bastante significativa, cresceu o interesse por topologias que, como o cubo, possuam um diâmetro que cresça de forma logarítmica em relação ao número de nós, mas possuam

um grau de nó constante. As três topologias apresentadas a seguir possuem esta característica.
O cubo *CCC(d)* (*Cube Connected Cycles*) de dimensão *d* se baseia em um hipercubo da mesma dimensão e substitui cada nó do hipercubo por um anel composto de *d* nós [16]. Desta forma o CCC possui um grau do nó constante igual 3 para qualquer dimensão, mantendo a relação logarítmica entre número de nós ($d2^d$) e diâmetro ($2d + \lfloor d/2 \rfloor$). A Figura 21 apresenta um CCC de grau 3.

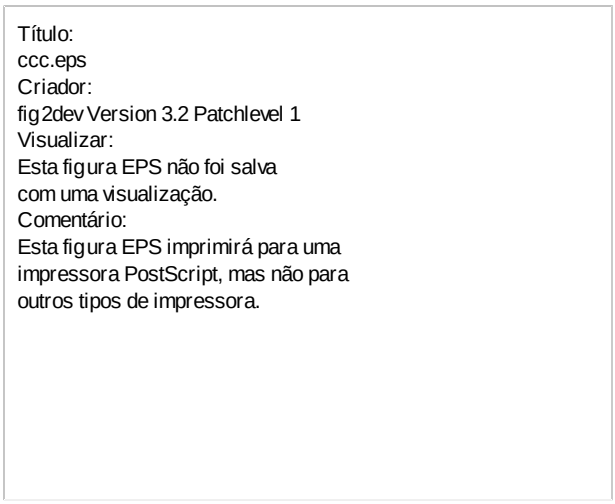

Figura 21: **CCC de grau 3**

A segunda topologia com esta característica é o grafo *Butterfly BF(d)* de dimensão *d* (Figura 22). Ele possui o mesmo número de nós que o CCC(d), porém, devido ao seu grau de nó igual a 4, um diâmetro menor ($2d + \lfloor d/2 \rfloor$).

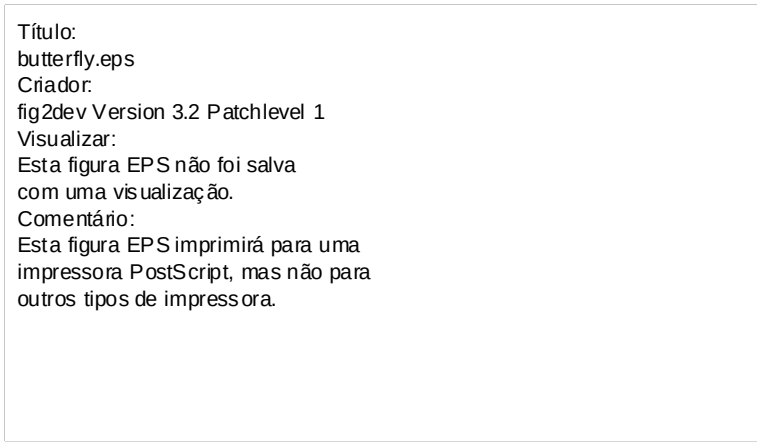

Figura 22: **Butterfly com dimensão 3**

Outro exemplo de grau de nó constante com diâmetro logarítmico são os grafos de *DeBrujn DB(d)*. Um grafo de DeBrujn com dimensão d possui *2d* nós e um diâmetro de *d*. O grau do nó é constante e igual a 4 (Figura 23).

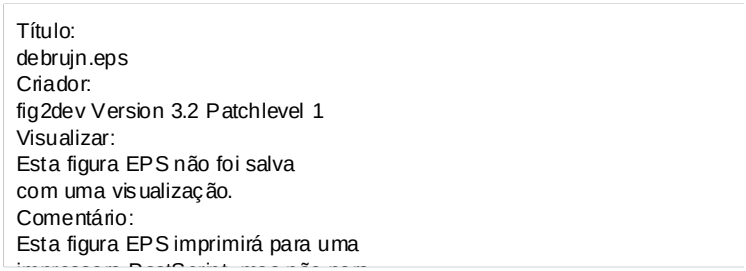

Figura 23: **Grafo de DeBrujn com dimensão 3**

Um resumo dos parâmetros das topologias apresentadas pode ser encontrado na Tabela 3 [7]. É importante lembrar que as topologias aqui apresentadas não representam a totalidade das topologias existentes e que podem ser combinadas de forma hierárquica, formando topologias híbridas, como é o caso no CCC, por exemplo, que combina o hipercubo e o anel.

Tabela 3: **Resumo das características das topologias apresentadas**

	Número de nós	Número de ligações	Grau do nó (máximo)	Diâmetro
Árvore Binária *B(h)*	$2^{h+1} - 1$	$2^{h+1} - 2$	3	$2h$
X-Tree X(h)	$2^{h+1} - 1$	$2^{h+2} - h - 4$	5	$2h-1$
Malha $M(a_1 \times a_2 \times \ldots \times a_d)$	$\prod_{k=1}^{d} a_k$	$\sum_{k=1}^{d}(a_k - 1)\prod_{i \neq k} a_i$	$2d$	$\sum_{k=1}^{d}(a_k - 1)$
Torus $T(a_1 \times a_2 \times \ldots \times a_d)$	$\prod_{k=1}^{d} a_k$	$d\prod_{k=1}^{d} a_k$	$2d$	$\sum_{k=1}^{d}\lfloor a_k / 2 \rfloor$
Hipercubo *H(d)*	2^d	$d2^{d-1}$	d	d
Cube Connected Cycles CCC(d)	$d2^d$	$3d2^{d-1}$	3	$2d + \lfloor d / 2 \rfloor$
Butterfly BF(d)	$d2^d$	$d2^{d+1}$	4	$d + \lfloor d / 2 \rfloor$
DeBrujn DB(d)	2^d	$2^{d+1} + 1$	4	d

4.2 Redes dinâmicas

Na interconexão de componentes com redes *dinâmicas* não existe uma topologia fixa que defina o padrão de comunicação da rede. Quando uma conexão entre dois pontos se faz necessária, a rede de interconexão se adapta dinamicamente para permitir a transferência dos dados. Uma rede dinâmica é dita *bloqueante* quando uma conexão estabelecida entre dois pontos P_1 e P_2 impedir o estabelecimento de outra conexão entre componentes quaisquer que não P_1 e P_2. Em uma rede dinâmica *unilateral* cada

componente possui uma ligação bidirecional com a rede (Figura 24a). No caso de uma rede dinâmica *bilateral* cada componente possui uma ligação de envio e outra de recebimento (Figura 24b).

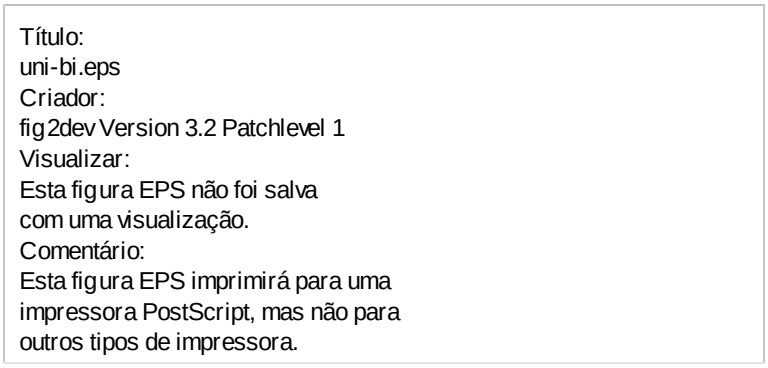

Figura 24: **Redes dinâmicas unilaterais e bilaterais**

Enquanto as redes estáticas vistas até agora são utilizadas na maioria dos casos na interconexão de nós de multicomputadores, redes dinâmicas por sua vez são tipicamente responsáveis pela interligação de processadores com memórias em multiprocessadores. Algumas redes dinâmicas são também empregadas em multicomputadores. Para facilitar sua análise as redes dinâmicas foram divididas em três grupos de acordo com suas características: *barramento*, *matriz de chaveamento* e *redes multinível*.

Barramento

Dentre as redes dinâmicas o *barramento* é a alternativa de menor custo. Porém, por se tratar de um canal compartilhado por todas as possíveis conexões, tem baixa tolerância a falhas (baixa confiabilidade) e é altamente bloqueante (Figura 25a). Sendo assim, acaba tendo sua escalabilidade comprometida, sendo utilizado em multiprocessadores com um número moderado de processadores (<50). As duas deficiências podem ser amenizadas com o uso de vários barramentos em paralelo (Figura 25b).

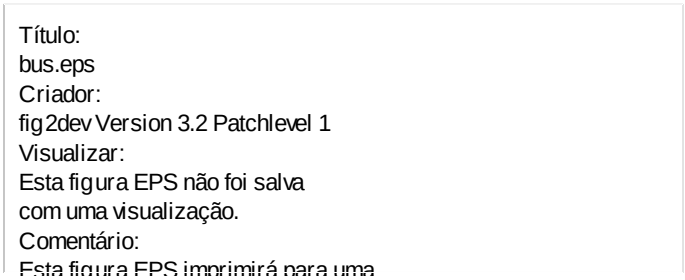

Figura 25: **Diferentes tipos de barramento**

Matriz de Chaveamento

A rede dinâmica de maior custo é a *matriz de chaveamento* (*crossbar switch*), que permite o chaveamento entre dois componentes quaisquer, desde que estes não se

encontrem já ocupados. Uma matriz de chaveamento pode ser usada como rede unilateral para ligar processadores a memórias em um multiprocessador (Figura 26a) ou como rede bilateral para interligar processadores de um multicomputador (Figura 26b).

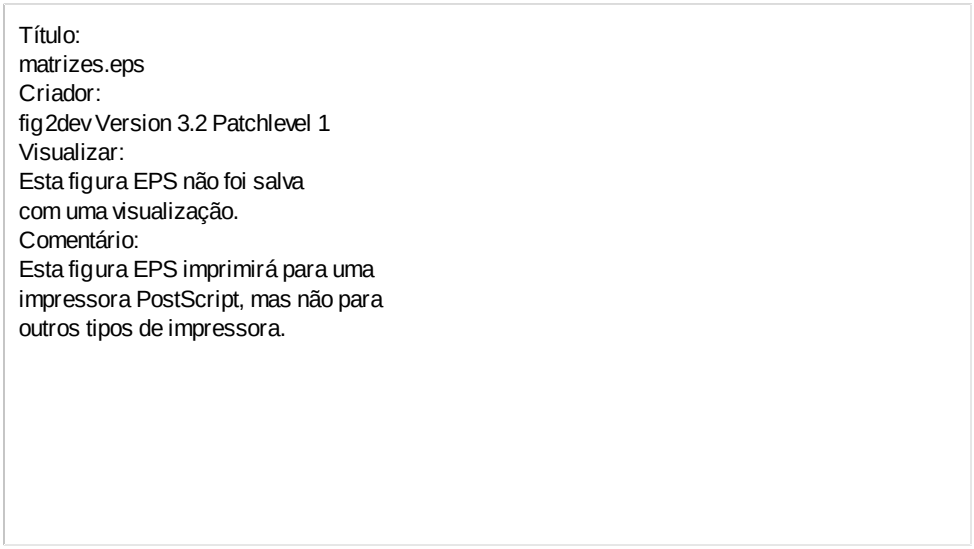

Figura 26: **Matriz de chaveamento unilateral e bilateral**

A matriz de chaveamento não é bloqueante e tem uma escalabilidade boa permitindo o acréscimo de componentes aos pares. O alto custo, que cresce de forma quadrática em relação ao número de componentes interligados, inviabilizam por razões econômicas a sua utilização para a interconexão de muitos processadores. Como solução alternativa pode ser usada uma rede hierárquica composta de várias matrizes de chaveamento, alternativa de menor custo, que por sua vez passa a ser bloqueante.

Redes Multinível

Uma outra aplicação para redes hierárquicas de matrizes de chaveamento é a construção de *redes de permutação multinível*. A idéia básica é a ligação de pequenas matrizes de chaveamento (normalmente de tamanho 2×2) em vários níveis consecutivos e conectá-las de forma a reduzir a probabilidade de conflitos entre conexões de diferentes pares [2]. Diferentemente das redes estáticas, latência neste tipo de rede é igual para qualquer par comunicante, crescendo porém de forma logarítmica de acordo com o número de possíveis conexões. As matrizes chaveadoras presentes na maioria das redes multinível tem tamanho 2×2 e permitem no mínimo 2, na maioria das vezes 4 posições de chaveamento (Figura 27).

Figura 27: **Posições de chaveamento**

A Figura 28 mostra um exemplo de rede multinível denominada *Omega*. O número de linhas é dado pela metade do número de possíveis componentes *n*, o número de níveis de $\log_2 n$, e no total $n/2 \log_2 n$ matrizes de chaveamento são utilizadas.

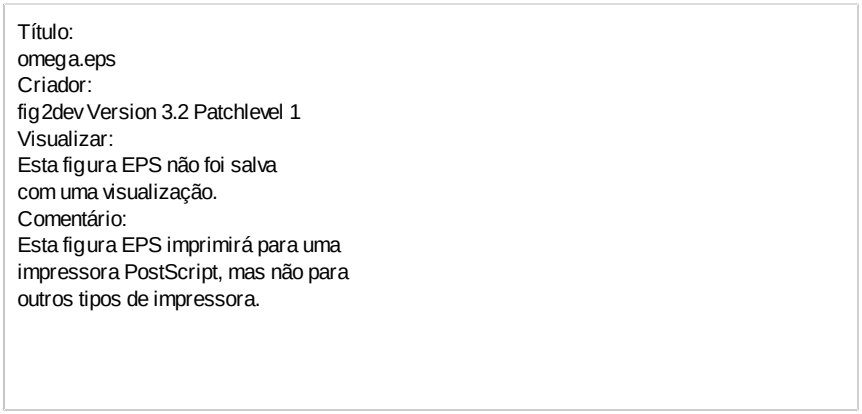

Figura 28: **Rede *Omega***

A Figura 29 mostra uma rede multinível denominada *SW-Banyan* que se utiliza do mesmo padrão de interligação da rede estática *Butterfly* para conectar as matrizes de chaveamento.

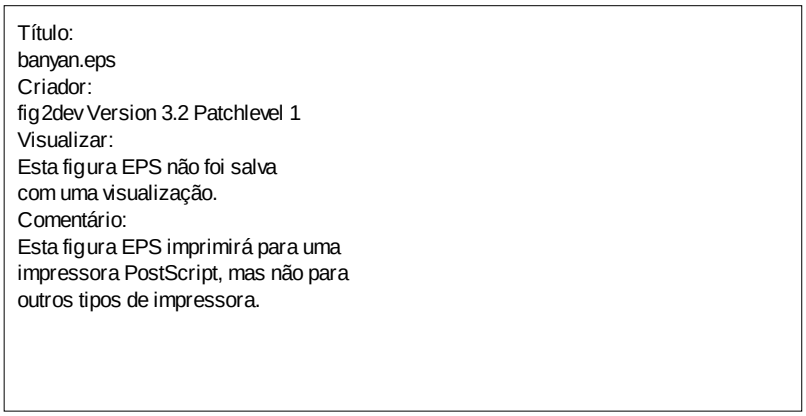

Figura 29: **Rede *Banyan***

Em ambas as redes existe apenas um possível caminho entre um entrada e uma saída. Sendo assim, a escolha do caminho é muito eficiente e pode ser feita de forma decentralizada. Porém, por causa desta falta de redundância, ambas são redes bloqueantes.

A Tabela 4 apresenta uma comparação entre as principais características das três classes de redes dinâmicas [10].

Tabela 4: **Comparação entre as redes dinâmicas**

	Barramento	Matriz de chaveamento	Redes multinível
Latência para Transferência de dados	Constante	Constante	$O(\log_k n)$
Largura de Banda por Processador	$O(w/n)$ até $O(w)$	$O(w)$ até $O(nw)$	$O(w)$ até $O(nw)$
Complexidade da Ligação	$O(w)$	$O(n^2w)$	$O(nw \log_k n)$
Complexidade de Chamamento	$O(n)$	$O(n^2)$	$O(n \log_k n)$
Observações	Barramento com n processadores e largura de w bits	Matriz $n \times n$ com largura da linha de w bits	Rede $n \times n$ usando $k \times k$ chaves com largura da linha de w bits

5 Roteamento de mensagens

Na construção de máquinas paralelas, por motivos de custo, na maioria dos casos são escolhidas redes de interconexão que não possuem ligações diretas entre todos os componentes de um sistema. Sendo assim, uma mensagem para chegar ao seu destino, pode precisar trafegar por nós intermediários. É dado o nome de *roteamento* ao procedimento de condução de uma mensagem através de nós intermediários até seu destino. Todos os nós envolvidos nesta condução participam do roteamento identificando se a mensagem já chegou ao seu destino e, se não for o caso, reenviando-a para um próximo nó. Fora o nó destino, todos os outros nós envolvidos neste procedimento, decidem o caminho seguido pela mensagem através da rede de interconexão e são chamados de nós roteadores.

Existem duas formas básicas de conduzir uma mensagem ao seu nó destino. Nas redes de telecomunicações é tradicionalmente usado o **chaveamento de circuito** (*circuit switching*) onde inicialmente é estabelecido um caminho fixo da origem ao destino e só depois são enviadas todas as mensagens (Figura 30). Este estabelecimento de conexão tem naturalmente um custo associado, que no caso das telecomunicações é pequeno em relação à duração da chamada. Esta forma de roteamento é usada por poucas máquinas paralelas, pois a comunicação entre dois nós nestes casos tem pouca duração (mensagens pequenas). Sendo assim, o estabelecimento da conexão teria uma representatividade significativa no tempo total de comunicação e a reserva de circuitos na rede para poucas mensagens subutilizaria os canais reservados e poderia ainda atrasar o estabelecimento de outras conexões.

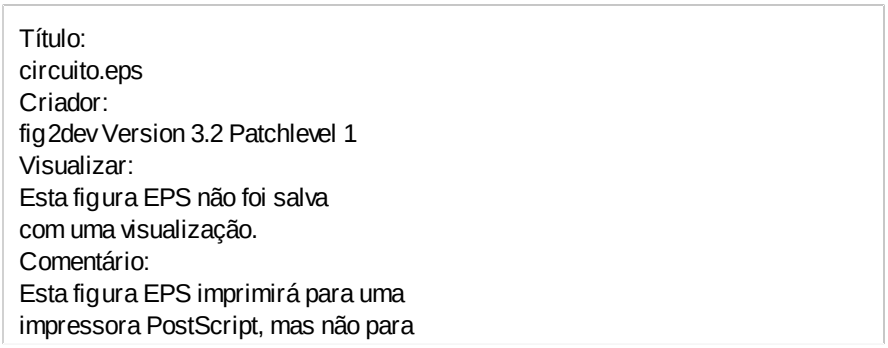

Figura 30: Chaveamento de Circuitos

Mais comum em máquinas paralelas é o **chaveamento de pacotes** (*packet switching*). Neste caso não existe caminho prévio pré-definido, mas cada mensagem decide a cada nó qual a direção que irá seguir na rede (Figura 31). Isto elimina o custo inicial de estabelecimento de circuito, mas embute um custo adicional para o roteamento de cada mensagem em cada um dos nós visitados.

Figura 31: **Chaveamento de Pacotes**

Duas grande vantagens do chaveamento de pacotes que tornam este tipo de roteamento atrativo para máquinas paralelas são a inexistência de uma reserva de canais da rede para uma única operação de comunicação e o estabelecimento dinâmico do caminho a ser seguido por uma mensagem. Não havendo reserva, mensagens com diferentes destinos podem compartilhar canais da rede de interconexão ao longo do seu caminho. O estabelecimento dinâmico do caminho, por sua vez, pode permitir que os algoritmos de roteamento reajam mais rapidamente a congestionamentos e falhas na rede de interconexão, optando por caminhos alternativos.

5.1 Políticas de Roteamento

As políticas de roteamento determinam como a mensagem a ser roteada vai ser manipulada em um nó intermediário. Esta manipulação diz respeito normalmente apenas ao armazenamento temporário da mensagem, sendo desta forma independente da rede de interconexão escolhida. As duas principais políticas para o roteamento de pacotes em redes de interconexão são apresentadas nesta seção. A política de *store-and-forward* foi usada na primeira geração de multicomputadores enquanto a política de *cut-through* ou *wormhole* é utilizada em máquinas mais modernas.

Roteamento Store-and-Forward

Na política *store-and-forward* (armazenar e repassar) um pacote tem que ser completamente armazenado em um *buffer* no nó intermediário antes que seja reenviado para o próximo nó. Desta forma, pacotes sucessivos são transmitidos de forma seqüencial, sem sobreposição no tempo. Na Figura 32 é mostrado o envio de um pacote com 4 células do nó 1 para o nó 4, através de dois nós intermediários (nó 2 e 3). O cabeçalho do pacote foi marcado com a letra **h** e é seguido de três células de dados, **a**, **b** e **c**.

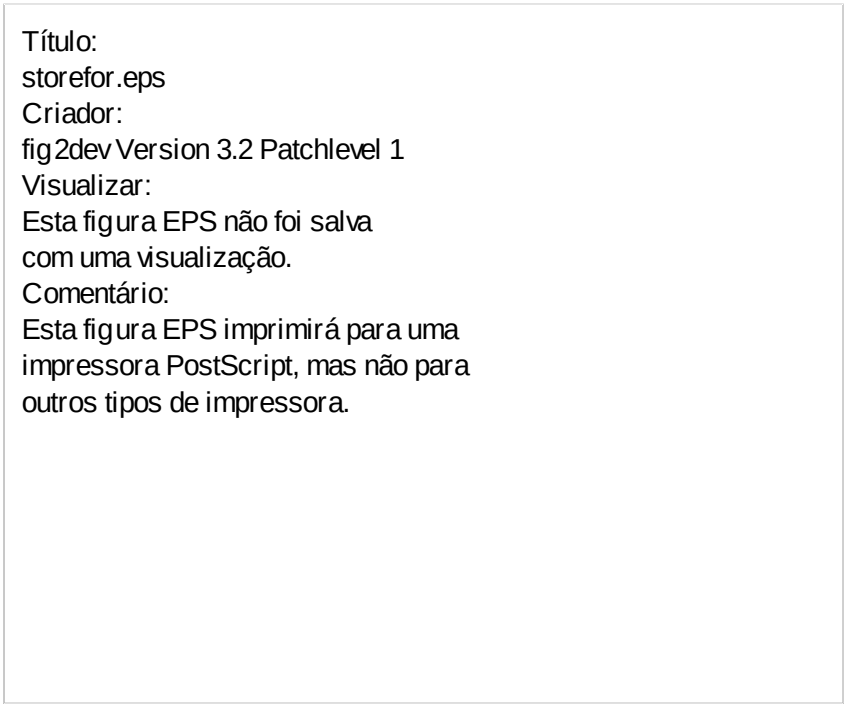

Figura 32: **Roteamento *Store-and-Forward***

O cabeçalho é usado para determinar o caminho de roteamento. Cada nó consome quatro unidades de tempo (igual ao tamanho do pacote) para repassar o pacote para o próximo nó. Deste forma, são necessárias 16 unidades de tempo (ciclos) para enviar uma mensagem da origem para o destino, se não ocorrer atraso por causa da disputa por recursos (contenção) ou congestionamento ao longo do caminho de roteamento. Sendo L o comprimento do pacote e N o número de nós da origem ao destino, o roteamento com *store-and-forward* precisa de $L(N-1)$ unidades de tempo.

Roteamento Cut-Through

Na política *cut-through* (cortar através) cada nó utiliza um *flit-buffer* para armazenar um *flit*, ou seja, uma célula do pacote de cada vez. A célula de cabeçalho é automaticamente repassada para o próximo nó, assim que decodificada (identificado qual o melhor caminho a seguir para chegar ao nó destino). Todas as células de dados que seguem, são retransmitidas pelo mesmo caminho que a célula cabeçalho, seguindo o princípio de um

pipeline (sobreposição de tempo). A analogia com o caminho que a cabeça da minhoca faz na terra (que é seguido pelo restante do seu corpo), fez esta política também ser muito conhecida por *wormhole* (buraco da minhoca).
Na Figura 33 somente 6 unidades de tempo são necessárias para o recebimento da uma mensagem de 4 flits, em comparação com os 12 ciclos necessários na política *store-and-forward*. Sendo L o comprimento do pacote e N o número de nós da origem ao destino, o roteamento com *wormhole* precisa de $L+N-2$ unidades de tempo. Para pacotes longos, a aceleração obtida com a política de *wormhole* sobre *store-and-forward* se aproxima de N vezes.

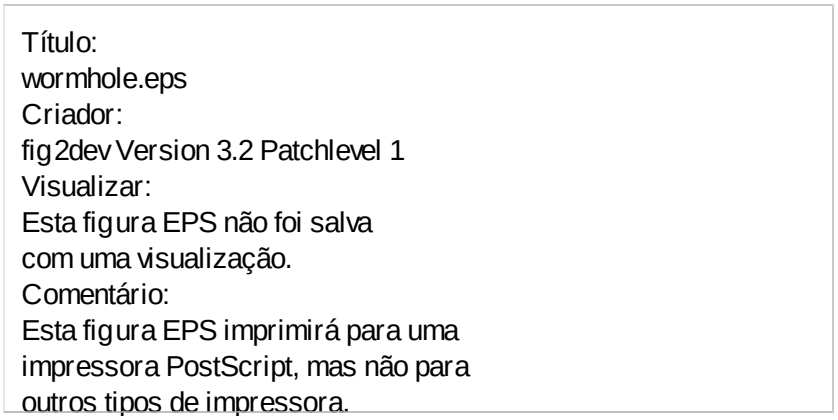

Figura 33: **Roteamento *Cut-Through* ou *Wormhole***

Um roteador que aplique a política de *wormhole* sobrepõe a transmissão de pacotes sucessivos, utilizando-se dos recursos da rede (canais de comunicação, chaveadores e interfaces de rede). Desta forma, um pacote passa por um nó intermediário sem atrasos resultantes de operações de cópia e/ou armazenamento temporário. Na verdade, os próprios recursos da rede funcionam como armazenamento temporário da mensagem roteada.
A característica mais significativa do roteamento *wormhole* é que pode ser facilmente provado que com este tipo de política a latência do roteamento é praticamente independente da distância entre o nó de origem e o nó destino [10]. Mais detalhes sobre a política de *wormhole* podem também ser obtidos em [4].

5.2 Algoritmos de roteamento

O algoritmo de roteamento determina por qual caminho uma mensagem vai ser roteada através de uma rede de interconexão até o seu destino. Como esta escolha de caminho depende naturalmente da topologia da rede de interconexão, os algoritmos de roteamento são especificados para cada topologia. Abaixo serão apresentados dois exemplos de algoritmos de roteamento com chaveamento de pacotes, um para Hipercubos e outro para Redes Multinível.

Algoritmo de roteamento para Hipercubos

Os nós de um hipercubo de grau *d* podem receber endereços binários, de tal forma que os *d* vizinhos de um nó possuam endereços que diferenciem-se em apenas um bit, representando esta alteração a troca de plano de um nó para outro. A Figura 34 apresenta um hipercubo de grau 3 endereçado desta forma. Cada um dos três bits do endereço representa um plano no espaço e todos os nós de um mesmo plano possuem o bit que representa este plano com o mesmo valor. Na figura o eixo *z* é representado pelo bit mais significativo do endereço, de forma que todos os nós da face frontal iniciam seu endereço com 0 (0**) enquanto que os nós da face de traz iniciam seu endereço com 1 (1**).

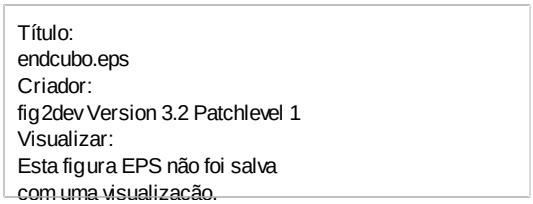

Figura 34: Hipercubo de grau 3 endereçado

Considerando este endereçamento, uma forma de efetuar o roteamento de mensagens no hipercubo é alterar o endereço de origem um bit por vez, até que se obtenha o endereço destino. A cada alteração do endereço, a mensagem é roteada para o respectivo vizinho (este vizinho garantidamente existe, pois por definição é possível endereçar um Hipercubo de grau *d* de forma que todos os nós possuam *d* vizinhos, cada um com o endereço variando em apenas um bit). Só faltaria definir em que ordem estes bits seriam alterados, ordem esta que afeta o caminho pelo qual uma mensagem é conduzida no Hipercubo. As duas possibilidades mais simples são do bit mais significativo para o bit menos significativo ou vice-versa. A Figura 35 apresenta como uma mensagem é roteada no Hipercubo do endereço 000 até o endereço 111 destas duas formas.

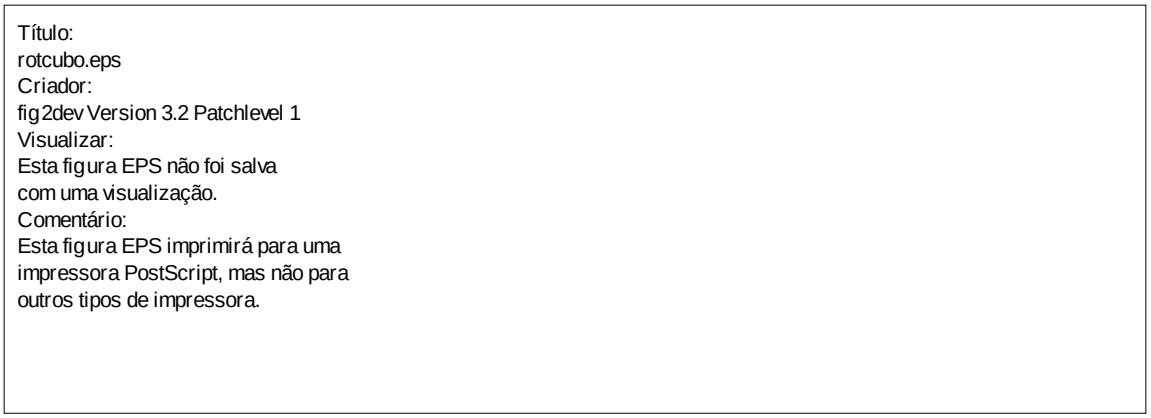

Figura 35: **Roteamento alterando os bits a partir do bit mais significativo (a) e a partir do bit menos significativo (b)**

Uma outra possibilidade seria a alteração de um bit escolhido de forma randômica em cada nó roteador. Isto resultaria que mensagens de um mesmo par de nós comunicante fossem provavelmente roteadas por diferentes caminhos na rede de interconexão.

Algoritmo de roteamento para Redes Multinível

Em redes dinâmicas multinível diversas matrizes chaveadoras são interligadas para permitir a comunicação entre os componentes da rede. Como vimos na seção 4.2, é a programação de forma dinâmica destas matrizes chaveadoras que determina a condução de uma mensagem na rede. Apesar de não existirem nós intermediários nesta condução, o procedimento é semelhante ao roteamento tradicional.

Uma forma interessante de rotear mensagens em redes Multinível é colocar a programação de todas as matrizes chaveadoras que uma mensagem irá transpor até o destino diretamente no seu cabeçalho (Figura 36). A Figura 37 apresenta como esta mensagem é roteada em uma rede Omega de três níveis do nó 000 ao nó 110. Neste caso, cada matriz roteadora retira a sua respectiva programação do cabeçalho da mensagem, dando seqüência na condução da mensagem.

Figura 36: **Mensagem com programação das matrizes chaveadoras no cabeçalho**

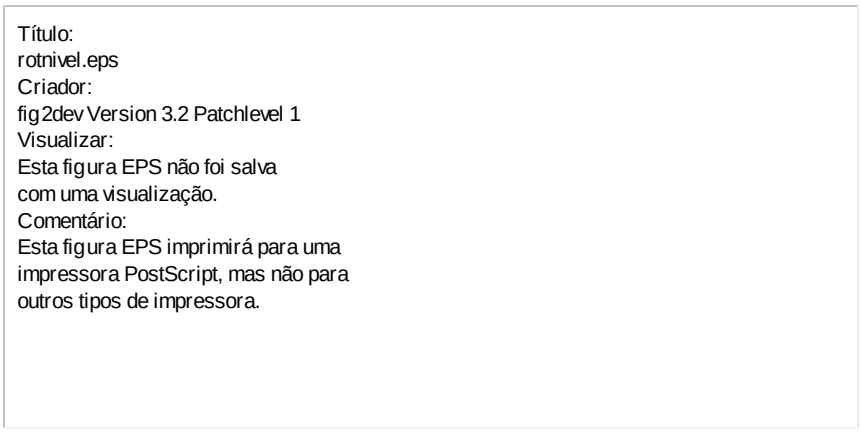

Figura 37: **Mensagem com programação das matrizes no cabeçalho (**Figura 36**) sendo roteada em uma rede Omega de três níveis**

É importante ressaltar que apesar do caminho a ser seguido pela mensagem já estar definido no momento do envio da mensagem até o seu destino, não se trata de chaveamento de circuito, já que o caminho não está reservado e pode ser usado por outras mensagens.

Referências

1. Almasi, G. S.; Gottlieb, A. *Highly Parallel Computing.* Benjamin/Cummings Publ. Comp., Redwood City, Cal., 1989.
2. C. Amza, A. L. Cox, S. Dwarkads, P. Keleher, L. Rajamony, W. Yu, W. Zwaenepoel. Trademarks: shared memory computing on networks of workstations. IEEE Computer, 29 (2): 18-28, February, 1996.
3. Broomel, G.; Heath, J. R. Classification Categories and Historical Development of Circuit Switching Topologies. ACM Comp. Surv. 15(2), 1983. pp. 95-133.
4. Culler, David E.; Pal, Singh J.; Gupta, A. *Parallel Computer Architecture: a hardware/software approach*. Morgan Kaufmann Publishers. 1999.
5. Flynn, M. J. Some Computer Organizations and their Effectiveness. IEEE TOC 21, 1972. pp. 948-960.
6. Fox, G. et al. Solving Problems on Concurrent Processors. Prentice Hall, 1998.
7. Heiss, H. Prozessorzuteilung in Parallelrechnern (Alocação de processadores em máquinas paralelas). Wissenschaftsverlag. 1994.
8. Hockney, R. W.; Jesshope, C. R. *Parallel Computers 2.* Adam Hilger, Bristol and Philadelfia, 1988.
9. Hwang, K.; Briggs, F. A. *Computer Architecture and Parallel Processing*. McGraw-Hill. 1985.
10. Hwang, K. Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill. 1993.
11. Hwang, K.; Xu, Z. Scalable Parallel Computing: technology, architecture, programming. McGraw-Hill. 1987.
12. Inmos Ltd. *Transputer Reference Manual.* Bristol, England. 1986.
13. Liao, C.; Jiang, D.; Iftode, L.; Martonosi, M.; Clark W., *Monitoring Shared Virtual Memory Performance on a Myrinet-base PC Cluster.* Proceedings of International Conference on Supercomputing, July 1998.
14. Nitzberg, B.; Lo, V. *Distributed Shared Memory: A Survey of Issues and Algorithms.* Computer, Vol. 24, No. 8, pp. 52-60. 1991.
15. Patterson, David A.; Hennessy, John L. *Computer Organization & Design: The Hardware Software Interface.* Morgan Kaufmann Publishers, Inc.
16. Preparata, F. P.; Vuillemin, J. The Cube-Connected Cycles: A Versatile Network for Parallel Computation. CACM 24(5), 1981. pp. 300-309.
17. Zomaya, Albert. *Parallel & Distributed Computing Handbook*. New York: McGraw Hill, 1996. 1232p.