3

NoC e NUCA: Conceitos e Tendências para Arquiteturas de Processadores Many-Core

Professores:

Henrique Cota de Freitas[1]
(hcfreitas@inf.ufrgs.br)
Marco Antonio Zanata Alves[2]
(marco.zanata@inf.ufrgs.br)
Philippe Olivier Alexandre Navaux[3]
(navaux@inf.ufrgs.br)

Resumo:

As decisões de projetos relativas às NoCs (Networks-on-Chip) e NUCAs (Non-Uniform Cache Architectures) são fundamentais para obtenção de alto desempenho, vazão de dados e escalabilidade nas futuras gerações de processadores many-core de propósito geral. As NoCs são as alternativas mais prováveis para suportar computação de alto desempenho em arquiteturas many-core devido aos diversos problemas e limitações físicas associadas aos fios. Desta forma, as NoCs deverão tomar lugar de interconexões tradicionais como barramentos e chaves crossbar para comunicação entre vários núcleos. Arquitetura não uniforme de memória cache é outra abordagem que está sendo discutida para os novos processadores many-core. As arquiteturas NUCA são uma alternativa para aumentar o desempenho dos processadores que possuem uma quantidade muito grande de núcleos de processamento. Em se tratando de memória compartilhada, é possível apontar a latência de acesso como o principal problema. As arquiteturas NUCA reduzem as latências de acesso a dados, aumentando a escalabilidade, além de possuírem grande potencial para o aumento de desempenho em relação as tradicionais arquiteturas uniformes de memória cache.

---

[1] Bacharel em Ciência da Computação pela Pontifícia Universidade Católica de Minas Gerais (2000) e mestre em Engenharia Elétrica pela mesma universidade (2003). Atualmente é doutorando em Ciência da Computação pela Universidade Federal do Rio Grande do Sul e professor assistente da Pontifícia Universidade Católica de Minas Gerais.

[2] Bacharel em Ciência da Computação pela Universidade Estadual Paulista (2006) e mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2009). Atualmente é doutorando em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

[3] Bacharel em Engenharia Elétrica pela Universidade Federal do Rio Grande do Sul (1970), especialização em Engenharia Nuclear pela Universidade Federal do Rio Grande do Sul (1970), mestrado em Física pela Universidade Federal do Rio Grande do Sul (1973) e doutorado em Informática pela Institut National Polytechnique de Grenoble (1979). Professor titular da Universidade Federal do Rio Grande do Sul.

3.1. Introdução

Aumentar o desempenho das aplicações é um dos principais objetivos durante o projeto da arquitetura de um processador. No entanto, devido a restrições no paralelismo de instruções, consumo de potência, além dos limites impostos pela latência do fio, o ganho de desempenho das gerações de processadores com um único núcleo (*single-core*) tem crescido a taxas inferiores ao que se esperava pela densidade de integração da Lei de Moore (HO, 2001) (KEYES, 2001) (KEYES, 2008). A solução para esta desaceleração na evolução e no ganho de desempenho dos processadores *single-core* está em uma nova geração de processadores com múltiplos núcleos (*multi-core*) (WOLF, 2004) (OLUKOTUN, 2005) (HILL, 2008). Atualmente os núcleos de um processador *multi-core* são mais simples, alcançando uma maior eficiência no consumo de potência e energia. Porém, múltiplos núcleos aumentam a capacidade de exploração do paralelismo no nível de *thread*, podendo ocasionar uma intensa comunicação interna.

Como conseqüência do aumento do número de núcleos, há também a necessidade de uma rede de comunicação entre os núcleos de alto desempenho e eficiente. Soluções tradicionais (ANDERSON, 1975) (AHMADI, 1989) (AGARWAL, 1991) (KUMAR, 2005) de interconexão, e.g., barramentos e chaves *crossbar*, possuem problemas de escalabilidade e limites físicos. Portanto, aumentar o tamanho destas soluções de interconexão implica diretamente nas seguintes conseqüências: aumento da resistência do fio, aumento da latência de comunicação e aumento da complexidade de roteamento do fio. Neste sentido, a comunidade tem trabalhado com o objetivo de reduzir a influência dos limites físicos dos fios através de uma arquitetura de rede chamada de NoC (*Network-on-Chip*) (BENINI, 2002) (BENINI, 2005) (BJERREGAARD, 2006) (OGRAS, 2007).

As NoCs são redes de comunicação em *chip* baseadas em troca de pacotes, que possuem os seguintes componentes básicos: roteadores, adaptadores de rede, e *links* de comunicação. A maioria das NoCs é projetada para que um roteador seja responsável pela interconexão de um núcleo à rede de comunicação. Por exemplo, em uma rede com topologia *mesh*, um roteador teria entre o núcleo e os demais roteadores adjacentes *links* curtos e com baixa resistência e latência de comunicação. Em contrapartida, cada roteador aumenta em complexidade a gerência de comunicação interna, podendo ou não aumentar a latência de comunicação. Como conseqüência do uso das NoCs, a possibilidade de uma maior quantidade de núcleos de processamento aumenta consideravelmente.

Uma vez que a rede em *chip* é responsável por interconectar núcleos de processamento e qualquer outro tipo de periférico, os acessos às memórias *cache* podem não ser mais uniformes. Isto significa que núcleos de processamento mais distantes levariam mais tempo para acessar uma memória *cache* compartilhada do que um núcleo mais próximo a esta memória. Esta arquitetura de memórias *cache* é chamada de NUCA (*Non-Uniform Cache Architecture*) (KIM, 2002) (KIM, 2003). Estudos e projetos relativos às NUCAs e NoCs ainda estão em desenvolvimento pela comunidade científica, mas possuem resultados promissores para as novas gerações de processadores *many-core*. *Chips* com milhares de núcleos ainda é uma projeção, mas os desafios para viabilizar *chips* com dezenas ou centenas de núcleos já começam a ser enfrentados e resolvidos. Os processadores com uma quantidade elevada de núcleos são

chamados de *many-core*. Um exemplo deste tipo de processador de propósito geral em desenvolvimento é da linha de pesquisa Tera-scale da Intel (INTEL, 2007).

Portanto, o objetivo deste curso é apresentar os principais conceitos e tendências relativas às arquiteturas de NoCs e NUCAs para as novas gerações de processadores *many-core* de propósito geral. Nesse sentido, o texto está dividido em três partes que focam em: arquiteturas de processadores *many-core*, arquiteturas de NoC, e arquiteturas NUCA. Ao final deste curso é apresentada uma conclusão analisando as diversas abordagens de arquiteturas apresentadas.

3.2. Arquiteturas de Processadores Many-Core

Apesar de processadores com múltiplos núcleos existirem desde a década passada, em sistemas dedicados, e.g., processadores de rede (INTEL, 2001) (COMER, 2003), o surgimento de processadores de propósito geral tem alguns anos. Os fatores que mais motivaram o surgimento destes processadores são os limites da Lei de Moore (KEYES, 2008) e o alto consumo de potência. A base do desempenho dos processadores se valia pelo paralelismo no nível de instrução, com *pipeline* superescalar, no máximo duas *threads* simultâneas (SMT: *Simultaneous Multihreading*) e a constante elevação da freqüência de operação. Porém, o consumo de potência já atingia limites impraticáveis, o paralelismo no nível de instrução já havia encontrado limites de desempenho e múltiplas *threads* simultâneas em um mesmo *pipeline* muitas vezes degradava o desempenho. Sendo assim, era necessária uma nova abordagem para a constante evolução no aumento do desempenho, atendendo limites de consumo de potência. Os processadores com múltiplos núcleos, já usados em outras áreas, se tornaram alternativas para computadores de propósito geral.

A Figura 3.1 mostra um exemplo de arquitetura *quad-core*, sendo que cada núcleo possui um *pipeline* superescalar de duas vias. Em comparação com outras arquiteturas, cada um dos núcleos pode ser considerado relativamente simples. Neste exemplo, o objetivo é suportar tanto paralelismo no nível de instrução quanto no nível de *thread*. No entanto, as *threads* são somente suportadas pela existência de mais de um núcleo, já que cada núcleo não é *multithreaded*, sendo responsável pela execução de apenas uma *thread*. Processadores *multi-core* que suportam múltiplas *threads* também podem ser chamados de *Chip Multithreading* (CMT) (SPRACKLEN, 2005) (FREITAS, 2006).

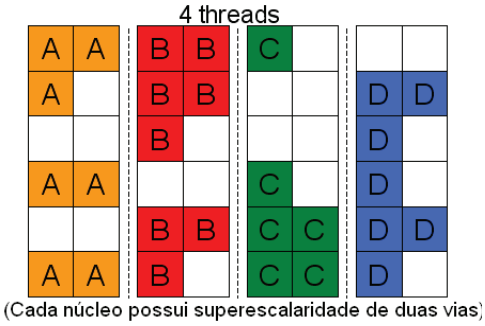

Figura 3.1. Execução de múltiplas threads em um processador multi-core

Existem processadores *multi-core* de propósito geral que suportam múltiplas *threads* em cada núcleo, tanto por entrelaçamento de instruções, (IMT: *Interleaved*

Multithreading) (KONGETIRA, 2005) quanto por execução simultânea de instruções (SMT) (KALLA, 2005). O motivo para escolha de uma técnica ou outra, assim como a escolha entre *single-core* ou *multi-core*, se deve muito à característica das cargas de trabalho. A Figura 3.2 ilustra um experimento realizado em 1996 (OLUKOTUN, 1996) que mostra as vantagens de usar uma arquitetura *multi-core* em relação a uma arquitetura superescalar. O experimento mostra que a área ocupada pelas duas soluções é a mesma, e as características de cada arquitetura são as seguintes:

- Ambas são baseadas em arquiteturas do processador MIPS R10000.
- A arquitetura (a) é um superescalar de seis vias de execução.
- A arquitetura *multi-core* (b) possui quatro núcleos, sendo que cada um possui *pipeline* superescalar de duas vias.

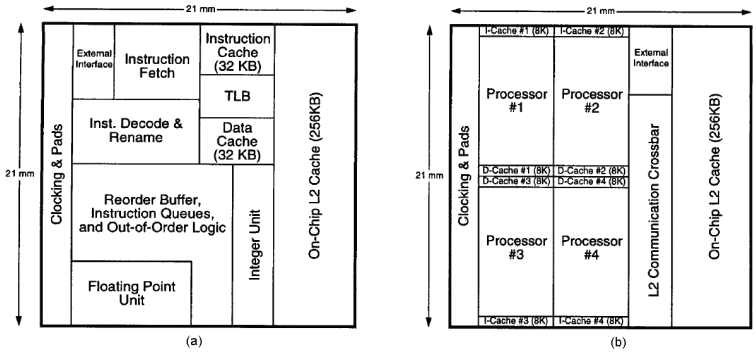

Figura 3.2. Comparação entre arquiteturas. (a) superescalar, (b) multi-core (OLUKOTUN, 1996)

Em resumo os resultados mostram que uma arquitetura superescalar possui desempenho melhor se a carga de trabalho possuir alto paralelismo no nível de instrução. Por outro lado, a arquitetura *multi-core* possui melhor desempenho se a carga de trabalho tiver um alto paralelismo no nível de *thread*. Este experimento foi o início para o surgimento do processador Niagara (Ultrasparc T1) (KONGETIRA, 2005), que possui oito núcleos escalares com suporte IMT cada um. Este processador possui alta vazão de *threads* e baixa vazão de instruções. Portanto, uma arquitetura como esta não é adequada para um computador pessoal, mas adequada para um sistema de aplicações *web* e banco de dados, onde a carga de trabalho possui alto paralelismo de *threads*.

Pode ser visto na Figura 3.2(b), que para quatro núcleos uma chave *crossbar* é suficiente para prover a comunicação interna. A mesma alternativa foi adotada para o processador Niagara, que possui oito núcleos. No entanto, para a nova geração de processadores *many-core*, uma única chave *crossbar* possui limites físicos que impedem o aumento da quantidade de núcleos interconectados a ela. A Figura 3.3 mostra que o projeto *Tera-scale* da Intel (INTEL, 2006) (INTEL, 2007) desenvolve um processador *many-core* com oitenta núcleos e uma *network-on-chip*. Em realce, cada núcleo possui um roteador e, portanto, a comunicação interna é por troca de mensagens. A Seção 3.3 descreve os principais detalhes das *Networks-on-Chip*.

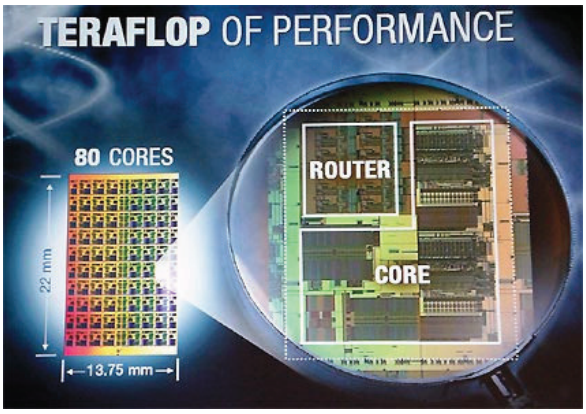

Figura 3.3. Arquitetura many-core de desempenho Teraflop (INTEL, 2006)

As principais vantagens dos processadores *many-core*, além do aumento do desempenho, podem ser ilustrados pela Figura 3.4, tais como:

(a) Ativação somente dos núcleos necessários.

(b) Redistribuição de carga de trabalho em função de altas temperaturas atingidas pelos núcleos.

(c) Núcleos reservas para substituição de núcleos com problemas.

(d) Expansão da funcionalidade com diversos dispositivos dentro do *chip*.

Outro detalhe que pode ser percebido pela Figura 3.4 é a necessidade de uma rede de comunicação eficiente e que possibilite as vantagens apresentadas. Na seção seguinte as diversas abordagens são descritas.

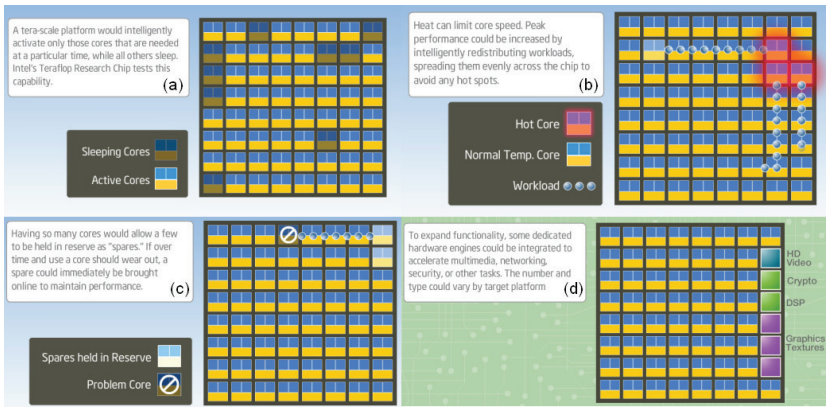

Figura 3.4. Justificativas do projeto Tera-scale: (a) utilização inteligente de núcleos, (b) redistribuição de carga de trabalho, (c) núcleos reservas, (d) integração de dispositivos dedicados (INTEL, 2007)

3.3. Conceitos e Arquiteturas de NoCs

De acordo com os limites de escalabilidade impostos pelos fios, as tradicionais soluções de interconexão largamente utilizadas em arquiteturas *multi-core*, tais como barramento e chave *crossbar*, são impraticáveis para arquiteturas *many-core*. A solução que vem

sendo estudada e proposta através de várias pesquisas é a *Network-on-Chip* (BENINI, 2002) (BENINI, 2005) (BJERREGAARD, 2006) (OGRAS, 2007).

Nesta seção é dada uma ênfase às NoCs apresentando as principais abordagens relativas às arquiteturas da rede ou roteador, topologias e protocolos, além de uma análise da viabilidade em comparação com as alternativas de interconexões tradicionais.

3.3.1. Arquiteturas de NoCs

Uma *Network-on-Chip* é composta por três elementos básicos: roteador, interface e *links* de comunicação. O roteador é o elemento principal responsável pela interconexão da rede, pela definição de rotas, pelo controle de fluxo, qualidade de serviço e, portanto, pela garantia de entrega do pacote de dados. Por se tratar de uma rede de comunicação composta por roteadores, o mecanismo de entrega de dados é através de passagem de mensagem ou pacotes de rede. Interligando os roteadores existem os *links* de comunicação. Estes *links* são os fios responsáveis pela existência do caminho a ser percorrido pelos pacotes. A forma como os roteadores estão interconectados pelos *links* dá origem à topologia da rede. A Seção 3.3.2 apresenta detalhes e comparações entre topologias propostas para NoCs. O último elemento que compõe uma NoC é a interface de rede. Esta interface também é chamada de adaptador ou *wrapper*, sendo necessária para garantir a correta comunicação entre a rede (roteadores da NoC) e os núcleos ou periféricos que estão interconectados. Esta interface garante que haja uma correta comunicação entre protocolos diferentes (NoC, núcleo, memória, etc).

A Figura 3.5 ilustra um exemplo de NoC baseada na topologia *mesh*. Neste caso, a NoC é uma *mesh* 3x3 que interconecta três núcleos de processamento através de três interfaces de rede. Estas interfaces estão interconectadas, cada uma, a um roteador diferente da NoC e, portanto, cada roteador é específico para um núcleo de processamento. Os demais roteadores da NoC nesta figura não estão interconectados a outros componentes do *chip*, mas poderiam ser memórias, ou qualquer *hardware* dedicado.

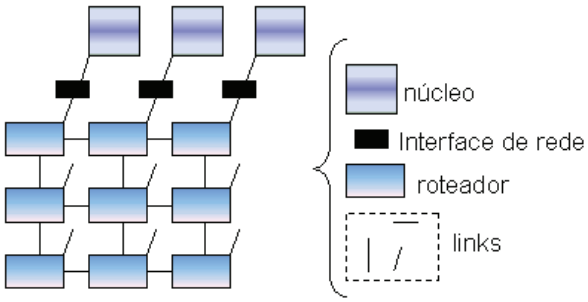

Figura 3.5. Arquitetura básica de uma NoC com topologia mesh

Um *chip* composto por uma NoC, núcleos de processamento, memórias e outros *hardwares* dedicados representam um sistema de computação. Portanto, para este caso existe um termo muito comum conhecido como Sistema-em-*Chip* ou *System-on-Chip* (SoC). Os projetos de arquiteturas de processadores *many-core* caminham para uma heterogeneidade de componentes em *chip* que os aproximam desta classificação.

A Figura 3.6 ilustra uma arquitetura típica de roteador para uma NoC *mesh* apresentada na Figura 3.5. Estes roteadores trabalham com roteamento XY e, portanto, possuem quatro portas de entrada/saída para norte, sul, leste e oeste. Além disso, é necessária uma porta para o núcleo de processamento através da interface de rede. Para a interconexão entre as portas o roteador se baseia em uma arquitetura simples de chave *crossbar*, neste exemplo 5x5, normalmente sem nenhum adicional de complexidade, como o suporte a *broadcast*. Para definição das conexões que devem ser realizadas, é necessário um Árbitro. A função principal deste mecanismo de arbitragem está na solução de conflitos na utilização da chave *crossbar* e, por conseqüência, na liberação de pacotes dos *buffers* de entrada. Neste exemplo de roteador, os *buffers* estão presentes apenas nas entradas. É importante ressaltar, que a arquitetura deste roteador ilustra um exemplo de circuito dedicado e não programável.

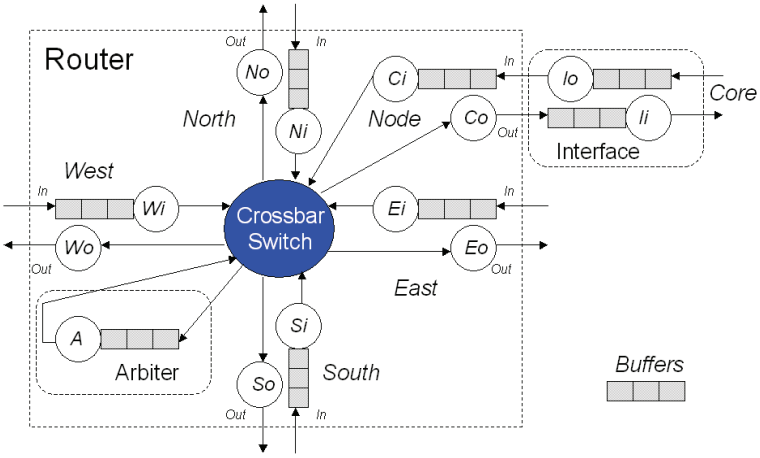

Figura 3.6. Arquitetura básica de um roteador de NoC para topologia mesh (modelo de filas)

Roteadores programáveis também podem ser utilizados no projeto de uma *Network-on-Chip*. A Figura 3.7 ilustra o projeto de um roteador com um processador de rede gerenciando um aglomerado de núcleos de processamento interconectados a uma chave *crossbar* e *buffers* de entrada. Neste caso o roteador não gerencia apenas um núcleo como ilustrado pela Figura 3.6, mas oito núcleos. Uma NoC constituída por um roteador deste tipo teria a interconexão entre dois roteadores através de uma das oito portas de entrada. Projetos de NoCs para *clusters* ou aglomerados de núcleos também podem ser encontrados na literatura (LENG, 2005) (NIEMANN, 2005) (FREITAS, 2008a) (FREITAS, 2008c).

O impacto da utilização de *buffers* pode estar relacionado ao tamanho, quanto maior a profundidade do *buffer*, maior o tamanho da NoC, ou pelos seguintes motivos relacionados à chave *crossbar* (AHMADI, 1989):

- *Buffers* de entrada: As técnicas de arbitragem são relativamente simples, possui uma melhor relação de área e potência, além de proporcionar um melhor desempenho para a chave *crossbar*.
- *Buffers* de saída: Em função de N entradas conectadas a cada um dos *buffers* de saída, a chave *crossbar* precisa ser N vezes mais rápida. A adoção de *buffers* de

saída não é a mais adequada para alto desempenho. No entanto, existem vantagens em se tratando da eliminação do bloqueio de pacotes que não receberam permissão de envio porque o primeiro pacote da fila ainda não teve liberação de uma determinada saída. Este problema é conhecido como *head of the line blocking* e pode acontecer nas soluções com *buffers* de entrada.

- *Buffers* de *crosspoint*: Cada ponto de conexão da chave *crossbar* possui um *buffer*. É utilizada a técnica de roteamento chamada de *self-routing*. Neste caso, em cada *crosspoint* seria necessário além do *buffer* um decodificador para decisão de envio ou não do pacote. Esta solução aumenta o tamanho e a potência consumida da chave *crossbar*.

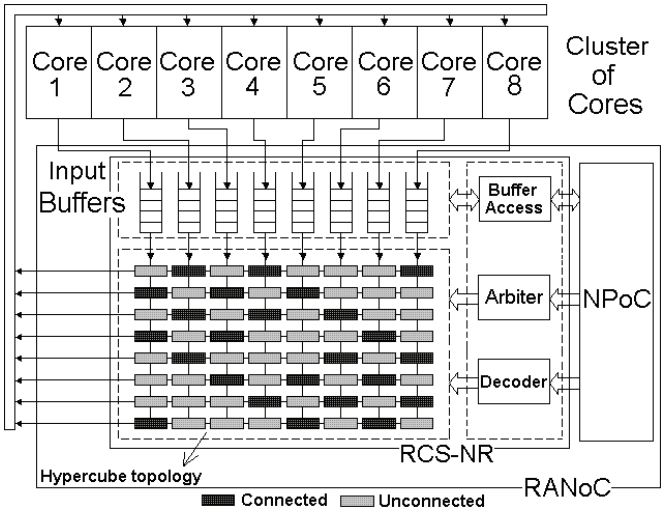

Figura 3.7. Arquitetura de um roteador programável (FREITAS, 2008b)

3.3.2. Topologias de NoCs

A definição de uma topologia de NoC, tal como de um roteador, está relacionada à carga de trabalho que será executada. A Tabela 3.1 ilustra as principais NoCs e suas respectivas topologias. Nesta seção é feita uma análise das principais características de cada uma destas topologias.

Através da Tabela 3.1 é possível citar três tipos de topologias: fixas, sem fio e reconfiguráveis. As topologias fixas são alternativas clássicas de adoção de uma determinada forma de interconexão que privilegie um comportamento específico de uma determinada carga de trabalho (BERTOZZI, 2005) (HO, 2006). Topologias sem fio são alternativas recentes para eliminar as limitações do fio no projeto de NoCs através de uma tecnologia chamada de *Radio-on-Chip* (CHANG, 2001). Por fim, as topologias reconfiguráveis utilizam plataformas programáveis para que sejam realizadas adaptações na forma de interligações em função de mudanças no padrão de comunicação das cargas de trabalho. Espera-se através da reconfiguração um aumento na flexibilidade de topologias da NoC.

Tabela 3.1. Exemplos de topologias de NoCs

Network-on-Chip	**Topologia**
SPIN (ANDRIAHANTENAINA, 2003)	Árvore gorda
ASoC (LIANG, 2000)	*Mesh* 2D
Dally (DALLY, 2001)	*Torus* 2D
Nostrum (MILLBERG, 2004)	*Mesh* 2D
Sgroi (SGROI, 2001)	*Mesh* 2D
Octagon (KARIM, 2002)	Anel Chordal
Marescaux (MARESCAUX, 2002)	*Torus* 2D
AEtheral (RIJPKEMA, 2003)	*Mesh* 2D
Eclipse (FORSELL, 2002)	*Mesh* 2D hierárquica
Proteo (SIGÜENZA-TORTOSA, 2002)	Anel bi-direcional
Hermes (MORAES, 2004)	*Mesh* 2D
SoCIN (ZEFERINO, 2003)	*Mesh/Torus* 2D
SoCBUS (WIKLUND, 2003)	*Mesh* 2D
QNoC (BOLOTIN, 2004)	*Mesh* 2D
T-SoC (GRECU, 2004)	Árvore gorda
Bouhraoua (BOUHRAOUA, 2006)	Árvore gorda
CHNoC (LENG, 2005)	*Clusters* com topologia irregular
GigaNoC (NIEMANN, 2005)	*Clusters* com topologia regular. *Mesh* 2D.
IPNoSys (FERNANDES, 2008)	Programável. Roteadores ativos. *Mesh*, *torus* e *butterfly 2D*.
PNoC (HILTON, 2006)	Reconfigurável. Baseado em FPGA.
Bartic (BARTIC, 2005)	Reconfigurável. Baseado em FPGA.
CoNoChi (PIONTECK, 2008)	Reconfigurável. Baseado em FPGA.
Wireless NoC (WANG, 2007)	Sem fio irregular
RECONNECT (JOSEPH, 2008)	*Honeycomb mesh*. Topologia fixa usada para suportar o conceito *Polymorphic ASIC*.
Polymorphic NoC (MERCALDI-KIM, 2008)	Reconfigurável. Conjunto de Chaves *crossbar* interconectadas para reconfiguração de topologias.
MCNoC (FREITAS, 2008c)	Reconfigurável e Programável. Roteadores para *clusters* de núcleos interconectados em topologia estrela.

As topologias fixas descritas pela Tabela 3.1 podem ser analisadas segundo alguns critérios (STOJMENOVIC, 1997) (HWANG, 1998) (DE ROSE, 2003), conforme Tabela 3.2:

Tabela 3.2. Características de algumas topologias fixas

	Número de ligações	Grau do nó	Diâmetro
Árvore Binária*	$2^{h+1} - 2$	3	2h
Anel Chordal	2n	4	2
Anel Bi-direcional	n	2	n/2
Mesh 2D	2n-2r	4	2(r-1)
Torus 2D	2n	4	$n^{1/2}$-1
Honeycomb mesh 2D	g *n/2	3	$1{,}63\sqrt{n}$
Totalmente conectada**	$(n^2 - n) / 2$	n - 1	1

*Árvore Gorda é uma alternativa tolerante a falhas baseada na replicação de conexões ou uso de conexões com maior vazão nas ligações perto da raiz.
**Totalmente conectada serve como referência para análise.
h = altura, n = número de nós, r = linhas, g = grau do nó.

A topologia totalmente conectada é apresentada na Tabela 3.2 como referência e alternativa para alcançar melhor desempenho de uma rede. Neste caso, as características desta topologia mostram que ela possui um alto número de ligações, um alto grau do nó e um baixo diâmetro. Estas são características que descrevem um baixo número de saltos e alta redundância de ligações. Quanto menor o número de ligações e grau do nó, maior é o diâmetro (maior distância entre dois componentes quaisquer) da rede. O grande problema de uma topologia totalmente conectada é o custo. É uma rede grande com problemas físicos para roteamento dos fios.

As topologias mais encontradas na literatura são baseadas em *mesh* e *torus*. A principal característica está relacionada à capacidade de suportar aplicações cujos problemas podem ser particionados (e.g., operações com matrizes e processamento de imagens). As topologias *honeycomb mesh* ou *mesh* hexagonal são consideradas da mesma família e possuem desempenho similar. A vantagem de uma topologia *honeycomb* em relação à *mesh* está na diminuição do custo em função de uma menor área ou quantidade de *links*, além de facilitar o mapeamento de aplicações.

A topologia anel possui o menor custo entre as apresentadas, mas em contrapartida possui um diâmetro que cresce de forma linear em função do número de nós. Uma alternativa que mantêm um baixo custo é a topologia anel *chordal* que possui caminhos alternativos, aumentado o grau do nó e diminuindo o diâmetro. Esta alternativa é mais tolerante a falhas do que a versão original (anel) e, portanto, mais confiável.

A topologia árvore binária é uma solução interessante para aplicações baseadas em algoritmos de divisão e conquista. O diâmetro cresce de forma linear em relação a altura e a confiabilidade é relativamente baixa, já que a perda de um nó pode separar a topologia em duas partes. Outro problema está relacionado ao nó raiz que é um gargalo entre as subárvores da esquerda e direita. A solução para este problema está na árvore gorda. Esta solução é mais tolerante a falhas, pois possui um maior número de ligações

entre os nós próximos da raiz. Outra solução é aumentar a largura de banda destas ligações, reduzindo as contenções de comunicação do efeito concentrador / gargalo da raiz da árvore.

Encontrar uma topologia, que atenda os melhores requisitos de desempenho, escalabilidade, confiabilidade e custo não é tão trivial (KREUTZ, 2005). No projeto *Tera-scale* da Intel (Figura 3.8) duas topologias estão em teste: anéis interconectados e *mesh*. Devido ao grande número de diferentes domínios de aplicação ou padrões de comunicação, algumas propostas apontam para o uso de conceitos de reconfiguração para aumentar a adaptabilidade da rede (BJERREGAARD, 2006) (OGRAS, 2007).

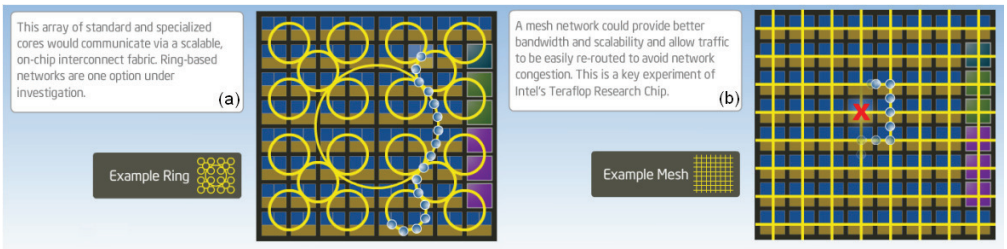

Figura 3.8. Topologias do projeto Tera-scale: (a) anéis interconectados, (b) mesh (INTEL, 2007)

3.3.3. Tipos de Protocolos

Políticas e estratégias de transporte de dados em uma NoC é de responsabilidade dos protocolos. A definição do protocolo descreve as principais características de funcionamento da rede. Neste sentido, os protocolos são capazes de garantir a entrega dos dados, a confiabilidade da rede, a melhor rota, além do melhor desempenho, entre outras. Os principais aspectos dos protocolos para NoCs (BJERREGAARD, 2006) são apresentados a seguir:

- Chaveamento por circuito: Existe uma rota dedicada para o caminho do dado. Esta rota (circuito) permanece reservada até que a transmissão do dado acabe.
- Chaveamento por pacote: Pacotes trafegam pela rede e o caminho é definido através de roteamento por salto. Não há reserva de caminho e, portanto, vários pacotes podem compartilhar o mesmo meio.
- Orientada a conexão: Existe uma conexão lógica entre origem e destino. Portanto, a transmissão só inicia após a confirmação de um estado de pronto entre o transmissor e receptor.
- Não orientada a conexão: Não existe uma conexão lógica. A comunicação é estabelecida sem o acordo entre origem e destino.
- Roteamento determinístico: A definição da rota é feita pela origem e destino. Um exemplo utilizado pelas topologias *mesh* e *torus* é o roteamento XY. Primeiro são percorrida as linhas e depois as colunas até o destino.
- Roteamento adaptativo: A definição da rota é feita a cada salto de roteamento. Neste caso, um roteador intermediário pode alterar o caminho que o pacote seguirá.

- Roteamento mínimo ou não mínimo: O roteamento é mínimo se sempre é feita a escolha do menor caminho entre origem e destino.
- Atraso versus perda: No caso de um protocolo baseado em atraso, o pior caso é um pacote atrasado. No caso de um protocolo baseado em perda, é possível excluir um pacote da rede, portanto, uma retransmissão seria o pior caso.
- Controle central ou distribuído: Pelo controle central, o roteamento é feito de forma global. No caso do distribuído, cada roteador define a melhor rota para o pacote.

Existem três técnicas de encaminhamento de pacotes conforme descrição a seguir:

- *Store-and-Forward*: Todo o pacote é armazenado em *buffers* do roteador para que o cabeçalho seja analisado. Em seguida o pacote é encaminhado.
- *Wormhole*: Enquanto o pacote é recebido seu cabeçalho é analisado. Definida a rota, todo o pacote é encaminhado sem a necessidade de armazenamento temporário em *buffers* para o nó seguinte.
- *Virtual cut-through*: Segue o mesmo mecanismo do *wormhole*, mas o nó antes de encaminhar espera uma confirmação do próximo nó destino para envio do pacote.

Os mecanismos para controle de fluxo são responsáveis por garantir o funcionamento da rede. Entre os principais benefícios estão:

- Garantir que pacotes não sejam descartados.
- Evitar retransmissões de pacotes.
- Diminuir o congestionamento da rede ou contenções nos roteadores.
- Otimizar o uso de recursos da rede.
    - *Buffers* menores.
    - Menor número de transmissão de pacotes.
    - Menor uso dos *buffers* e roteadores.
    - Menor consumo de potência e energia.
    - Melhor desempenho da rede.

Um dos conceitos básicos do controle de fluxo é ajustar a taxa de saída de dados de um transmissor para um receptor. Três abordagens clássicas são descritas a seguir:

- *Handshake*: também chamado de "aperto de mão", consiste em um acordo entre transmissor e receptor através de linhas adicionais de controle para liberação do envio de pacotes. Transmissor envia sinal de solicitação e receptor envia sinal de *buffer* livre.
- Créditos: o receptor envia a quantidade de espaço livre no *buffer* de entrada para que o transmissor saiba quantos créditos estão disponíveis para envio de pacotes.

- Canais virtuais: técnica para eliminar o problema *head of the line blocking* em que o primeiro pacote bloqueia os demais da fila. Através de canais virtuais é possível criar diversas saídas do *buffer* de entrada, como se este tivesse várias pequenas filas.

3.3.4. Análise da Viabilidade de NoCs

A Tabela 3.3 apresenta as vantagens e desvantagens da adoção de uma NoC em relação às soluções tradicionais (barramento e chave *crossbar*) mais utilizados atualmente nos processadores *multi-core*.

Tabela 3.3. Análise comparativa para uso de NoCs (adaptado de BJERREGAARD, 2006)

Tipo de Interconexão		Prós (+) e Contras (-)	
Barramento	Fio	O aumento do fio aumenta a resistência degradando o desempenho.	-
Chave *Crossbar*		O aumento do fio aumenta a resistência degradando o desempenho.	-
Network-on-Chip		Os fios são ponto-a-ponto entre roteadores e o desempenho não degrada em função do aumento de nós.	+
Barramento	Árbitro	O árbitro é um gargalo à medida que o número de nós aumenta.	-
Chave *Crossbar*		O árbitro pode ser centralizado ou descentralizado e não é o fator principal para degradação do desempenho em função do aumento dos nós.	+-
Network-on-Chip		As decisões de roteamento são distribuídas e não representam um gargalo.	+
Barramento	Largura de banda	A largura de banda é limitada e compartilhada por todos os nós.	-
Chave *Crossbar*		Cada interconexão é independente e a largura de banda de comunicação por conexão não é afetada pelas demais.	+
Network-on-Chip		A largura de banda não é afetada pelo aumento da rede.	+
Barramento	Latência	Latência é afetada pelo fio.	+
Chave *Crossbar*		Latência é afetada pelo fio.	+
Network-on-Chip		Latência é afetada pelas contenções em roteadores	-
Barramento	Compatibilidade	Em sua maioria são compatíveis com qualquer IP (*Intelectual Property*) incluindo os softwares.	+
Chave *Crossbar*		Em sua maioria são compatíveis com qualquer IP (*Intelectual Property*) incluindo os softwares.	+
Network-on-Chip		São necessários adaptadores (*wrappers*) entre os IPs e os softwares precisam de sincronização em sistemas *multi-core*.	-
Barramento	Complexidade	Conceitos simples e bem compreendidos.	+
Chave *Crossbar*		Conceitos simples e bem compreendidos.	+
Network-on-Chip		Projetistas precisam de uma reeducação em função dos novos conceitos.	-

Apesar das vantagens das soluções tradicionais no que diz respeito à simplicidade, compatibilidade e latência, os limites físicos impostos pelo fio, questões

relacionadas a escalabilidade e largura de banda apontam para a NoC como a melhor alternativa para futuras gerações de processadores *many-core*.

A próxima seção descreve quais as propostas de arquiteturas NUCA e como a rede em *chip* se insere neste projeto de memórias *cache*.

3.4. Conceitos e Arquiteturas NUCA

Nesta seção os principais conceitos e arquiteturas referentes às NUCAs (*Non-Uniform Cache Architectures*) são descritos. Porém, é importante fazer uma apresentação dos conceitos UCA (*Uniform Cache Architecture*) antes de descrever a nova tendência.

3.4.1. Conceitos de Arquiteturas de *Caches* Uniformes (UCA)

Como uma memória *cache* possui diversos espaços para gravação de dados, e cada local da *cache* pode armazenar conteúdo de diversos locais da memória, algumas informações adicionais devem ser adicionadas para sabermos, durante uma pesquisa, se um dado requerido encontra-se na memória *cache*.

A utilização de *tags* junto ao dado copiado à memória *cache* é útil uma vez que a *tag* contêm as informações de endereço necessárias para identificar se a palavra pesquisada corresponde com o dado requisitado. Assim podemos observar na Figura 3.9 um exemplo de memória *cache* com dados copiados da memória principal e o *tag* de identificação. Podemos ver no caso ilustrado, que o *tag* de dados precisa conter apenas a parte superior do endereço de memória referente para que seja possível sua identificação.

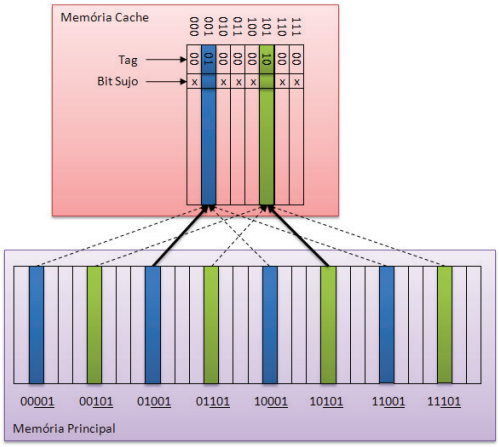

Figura 3.9. Diagrama de uma memória *cache* com mapeamento direto, apresentando a tag e o bit de validade (adaptado de PATTERSON, 2005)

Além da *tag* de dados, é necessária uma maneira para saber se um bloco de dados possui informações válidas. Um exemplo claro dessa necessidade pode ser ilustrado pela inicialização do sistema, quando a memória *cache* encontra-se apenas com possíveis sujeiras, sem dados reais copiados. O método mais comum para saber se a *tag* deve ser ignorada ou não é a inclusão de um *bit* de validade para indicar se a entrada contém um bloco válido.

3.4.1.1. Estratégias de Mapeamento de Dados

A estratégia adotada para definir o modo em que os blocos de dados deverão ser posicionados na memória *cache* é chamado de estratégia de mapeamento de dados.

Um modo usual de mapeamento de dados na memória *cache* é o mapeamento direto. Nesse modo de mapeamento cada local da memória é mapeado exatamente para um local na *cache*. Assim, a *tag* trará informações adicionais do endereço e o endereço base será diretamente associado ao endereço da *cache*. Este modo de mapeamento é atraente, pois é de fácil planejamento, uma vez que, se o número de entrada da *cache* for uma potência de dois, então o tamanho do endereço base pode ser calculado simplesmente usando os $\log^2$ bits menos significativos, e assim os dados podem ser associados a *cache* de forma direta.

Outra estratégia de mapeamento de dados na memória *cache* é o chamado mapeamento totalmente associativo. Neste mapeamento um bloco pode ser posicionado em qualquer local da *cache*. Porém, nesse esquema de mapeamento para um dado ser pesquisado na *cache*, todas as entradas precisam ser pesquisadas. Para a implementação desse tipo de mapeamento, é comum a utilização de comparadores paralelos, sendo que, para cada bloco deve existir um comparador de endereços para a pesquisa de blocos. A inclusão desses comparadores aumenta significantemente o custo de uma memória *cache*, sendo que esta técnica só é viável em memórias *cache* com pequeno número de blocos.

O ponto de equilíbrio entre os modos de mapeamento é o mapeamento associativo por conjunto. Neste modo de mapeamento, existe um número fixo de locais (no mínimo dois) onde cada bloco pode ser colocado. Uma dada memória *cache*, se for associativa por n conjuntos, sendo n o número de locais de mesmo endereço base onde cada bloco pode ser alocado, é chamada de *cache* associativa por conjunto de n vias. Neste modo de mapeamento cada bloco é mapeado para um conjunto único na *cache* de acordo com o endereço base, e um bloco pode ser alocado em qualquer posição desse conjunto. Logo, uma pesquisa de busca deverá acessar o conjunto referente ao endereço e depois pesquisar entre os elementos do conjunto o endereço requisitado.

A Figura 3.10 ilustra diversos modos de mapeamento de memória *cache*, onde pode ser visto os blocos utilizando mapeamento direto na Figura 3.10(a), ou associativo por conjunto na Figura 3.10(b) e ainda uma memória *cache* totalmente associativa ilustrado na Figura 3.10(c).

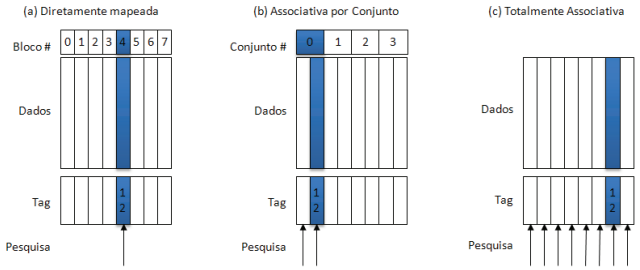

Figura 3.10. Alocação de um bloco de memória em uma memória *cache* com 8 blocos em diferentes tipos de mapeamento: (a) mapeamento direto, (b) mapeamento associativo por conjunto e (c) totalmente associativo (adaptado de PATTERSON, 2005)

O número total de blocos da *cache* é igual ao número de conjuntos multiplicados pela associatividade, porém, em uma *cache* associativa por conjunto, o número de comparadores de endereço costuma ser igual ao número de vias de associatividade. Assim, o aumento do grau de associatividade de uma dada memória tende a reduzir as faltas de conflito de endereços, mas também reduz a quantidade de endereços base, além de aumentar o custo relacionado aos comparadores de endereço.

3.4.1.2. Políticas de Substituição de Dados

Durante o processamento, quando ocorrer uma falta de dados, ao buscar o bloco de dados para a memória *cache*, esta deve escolher qual será a posição onde o bloco será gravado, ou seja, qual bloco de dados será substituído. Esta escolha é chamada de política de substituição de dados.

Quando se utiliza o modo de mapeamento direto em uma memória *cache*, ao trazer novos dados não existe escolha a ser feita para a substituição de dados (STALLINGS, 1996), devendo apenas gravar o novo dado em seu endereço referente. Porém, quando se utiliza modo de mapeamento completamente associativo ou associativo por conjuntos, existe uma escolha a ser feita sobre qual bloco será substituído para dar lugar ao novo bloco de dados. Existem quatro estratégias principais para política de substituição de blocos:

- Aleatória – Nessa política o bloco a ser substituído será encontrado ao acaso, ou seja, o sistema irá escolher o bloco alvo através de uma escolha aleatória.
- Menos Recentemente Usado – Também conhecida como política LRU (*Least Recently Used*), esta política baseia-se em reduzir a chance de descarte dos blocos usados mais recentemente. Desta forma, esta informação sobre quão recentemente cada bloco foi utilizado deverá permanecer junto a cada bloco para a decisão. Desta maneira pode-se dizer que esta política prevê o futuro baseando-se no passado, e também que esta política baseia-se no princípio de localidade temporal.
- Menos Freqüentemente Usado – Conhecida como política LFU (*Least Frequently Used),* esta política substitui os blocos que foram menos referenciados. Esta informação sobre quão freqüente cada bloco foi referenciado pode permanecer junto a cada bloco para a decisão.
- Primeiro a Entrar, Primeiro a Sair – Como pode ser difícil o cálculo do bloco que não é utilizado por um tempo mais longo, esta política também conhecida como FIFO (*First In, First Out*) baseia-se apenas nas informações sobre os dados mais antigos para achar o bloco alvo a ser substituído.

Podemos notar que a política LRU tende a ser de melhor desempenho em relação as outras, porém, a abordagem aleatória é a de mais fácil implementação, enquanto a política FIFO apresenta maior simplicidade em relação à política LRU, sendo que essa diferença se acentua na medida em que se aumenta o número de blocos a serem controlados.

3.4.1.3. Localizando um Bloco na Memória *Cache*

O acesso a dados da memória *cache* é uma das operações mais importantes da memória *cache* e também a que exige maior velocidade, uma vez que ao pesquisar dados na memória *cache*, o processador quase sempre ficará em estado latente até que o dado esteja disponível.

A busca por dados na memória *cache* ocorre em três etapas básicas:

- O endereço pesquisado é enviado para a memória *cache* apropriada.
- Se a memória *cache* sinalizar um sucesso na busca de dados, a palavra pesquisada estará disponível nas linhas de comunicação de dados.
- Se a memória *cache* sinalizar uma falta de dados, o endereço é solicitado no nível mais alto da hierarquia de memória *cache* ou então na memória principal, e só então após localizar a requisição o dado estará disponível.

Quando a solicitação chega até a memória *cache*, esta deve pesquisar entre os dados para indicar sucesso ou falta de dados e então prosseguir com o procedimento apropriado.

O dispositivo básico de busca de dados em uma memória *cache* associativa de 4 vias esta ilustrado na Figura 3.11, a palavra de endereço é decomposta em três blocos principais, de *tag*, *index* e *offset*. O bloco de *index* é utilizado para selecionar o endereço de interesse, onde as *tags* de todos os blocos dos conjuntos associativos devem ser comparadas. Assim, após acessar as linhas de endereço, o segundo passo é a comparação do *tag* vindo da pesquisa com os *tags* dos blocos de memória. Neste ponto, será identificado se houve um sucesso ou falta de dados, caso haja um acerto, o multiplexador é responsável por selecionar o dado correto e disponibilizar no canal de comunicação de dados, caso contrário a memória *cache* apenas indicará falta de dados.

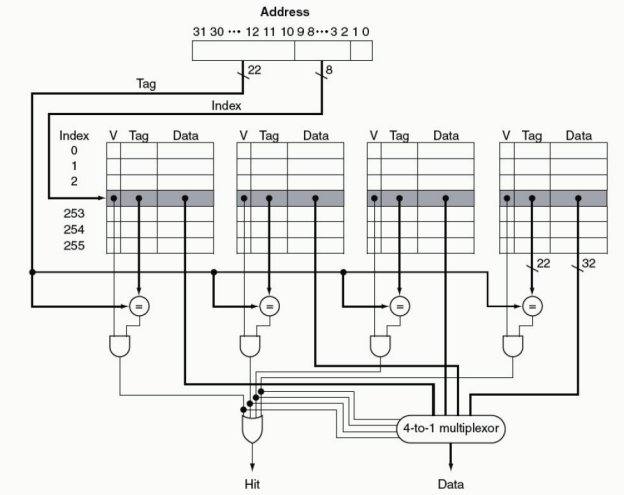

Figura 3.11. Implementação de um dispositivo de busca de dados em uma memória *cache* associativa em conjunto de 4 vias, utilizando quatro comparadores e um multiplexador (PATTERSON, 2005)

A estrutura interna de cada vetor de dados é um pouco mais complexa do que apresentada, porém, segue os mesmos princípios de funcionamento. A Figura 3.12 apresenta a estrutura de um vetor de dados da memória *cache*, em um esquema mais próximo da implementação física, onde existem dois blocos de informações, um contendo informações sobre as *tags* e o segundo contendo os dados propriamente ditos (WILTON, 1994). Assim, com a entrada do endereço o decodificador acessa a linha referente à palavra pesquisada. O *sense amplifier* é o responsável por detectar o que está dentro da célula de memória acessada. Como cada *sense amplifier* é compartilhado para diversas linhas de *bits*, os quais são detectados um de cada vez, um multiplexador deve ser utilizado antes de cada *sense amplifier*. Após identificar as informações de *tag* contidas no vetor de *tags*, o valor é comparado com o *tag* provido pelo endereço de entrada, onde o número de comparadores é igual ao número de conjuntos associativos da memória *cache*. Assim o resultado da comparação é utilizado para ativar a saída de validade (sucesso ou falta de dados) e também ativar a saída de dados quando o dado tenha sido encontrado.

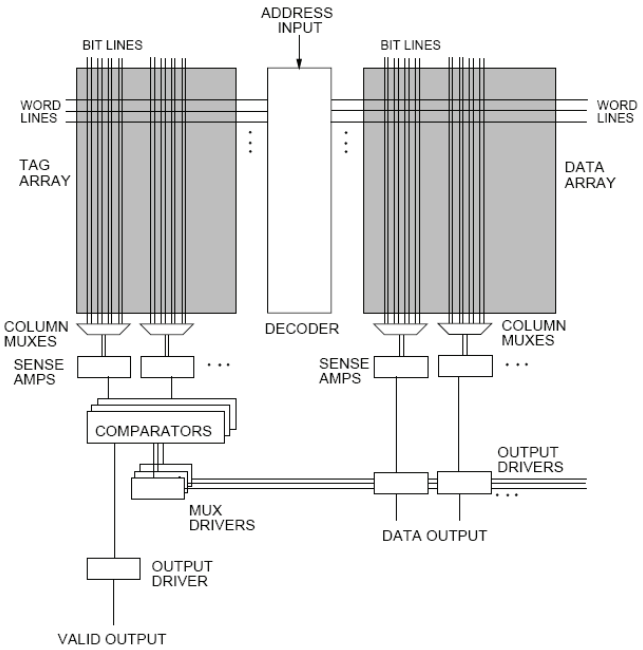

Figura 3.12. Estrutura de implementação de memória *cache* (WILTON, 1994)

3.4.1.4. Layout de Sub-blocos da Memória *Cache*

Com o aumento do tamanho das memórias *cache*, os diversos vetores de dados e *tags* foram divididos em vários bancos. Um layout utilizado com freqüência para os bancos de memória é o chamado H-tree (THOZIYOOR, 2007), assim a área ocupada pela memória *cache* atualmente é calculada não só considerando o tamanho dos bancos, mas também a área ocupada pelas interconexões.

A Figura 3.13 apresenta uma ilustração da disposição de 16 bancos de memória *cache* e as linhas de comunicação ligando todos os bancos.

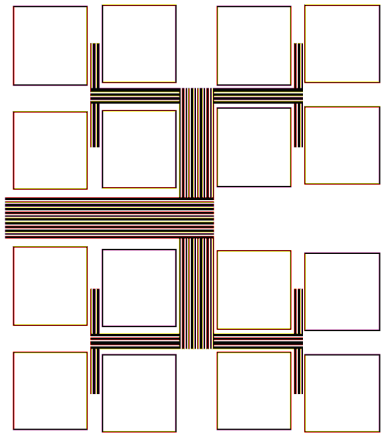

Figura 3.13. Estrutura de organização de sub-blocos de memória *cache* (THOZIYOOR, 2007)

Com as novas tecnologias de integração o tamanho das memórias *cache* tendem a continuar a crescer assim como a demanda por mais vazão de dados. Mesmo a memória *cache* sendo dividida em diversos vetores de dados, ela deve se comportar como um grande bloco, onde o tempo de acesso a dados é igual ao tempo de acesso ao vetor de dados mais distante. Portanto, as memórias *cache* devem ser projetadas considerando problemas de latência, largura de banda, interconexão, área entre outros.

Entretanto, para futuras tecnologias de integração, as memórias *cache* que utilizem o modelo atual de arquitetura vão sofrer com problemas de atraso do fio, desta maneira, o tempo de acesso aos dados para os grandes bancos de memória *cache* serão demasiadamente lentos, gerando grande contenção ao processamento (ALVES, 2009).

3.4.2. Memórias *Cache* de Arquitetura Não-Uniforme (NUCA)

As atuais memórias *cache* de múltiplos níveis são organizadas em alguns poucos níveis discretos. Tipicamente, cada nível inclui e replica o conteúdo dos níveis menores nos níveis acima, reduzindo assim os acessos aos níveis mais próximos da memória principal.

Ao escolher o tamanho de cada nível de memória *cache*, o projetista deve balancear entre o tempo de acesso e capacidade, preocupado sempre em manter os custos totais de área e potência.

Para as futuras tecnologias, grandes memórias *cache* dentro do *chip*, com um tempo único e discreto de acesso a dados não são apropriadas, uma vez que problemas relacionados ao atraso do fio dentro do *chip* estarão cada vez maiores. Este é o principal argumento para a criação de uma nova arquitetura de memórias *cache* que não seja uniforme. Além deste, podemos citar os problemas de eventuais falhas de componentes que vão estar presentes nos próximos processadores (AGGARWAL, 2007), onde o isolamento de módulos de memória e interconexão pode garantir que mesmo ocorrendo eventuais falhas de componentes os demais continuarão trabalhando.

O principal conceito envolvendo estas novas arquiteturas não uniformes é que os dados que residam mais próximos do processador podem e devem ter tempos de acesso

diferenciados. Sendo assim, dados mais próximos serão acessados a velocidades superiores que os dados mais distantes do processador. Além disso, problemas associados a número de portas e eventuais falhas podem ser em parte resolvidos por uma arquitetura não centralizada como veremos adiante.

3.4.2.1. Tipos Básicos de Arquiteturas de Memória *Cache* Não Uniforme

Pelo funcionamento básico envolvido por trás de memórias NUCA, cada sub-banco de memória *cache* deve retornar a sua pesquisa o mais rápido possível sem esperar os demais resultados. Diversos modelos de arquitetura podem ser descritos, a primeira proposta de memórias NUCA (KIM, 2002) levantou a possibilidade de três tipos básicos de arquiteturas não uniformes para memória *cache* como pode ser vista na Figura 3.14 em comparação com dois modelos UCA.

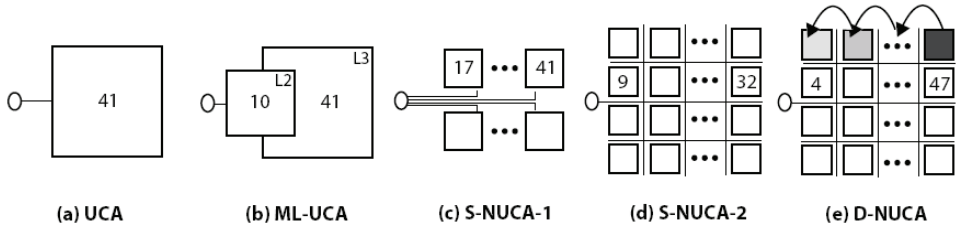

Figura 3.14. Diversas propostas de memória *cache*: (a) e (b) memórias UCA, (c) e (d) NUCAs estáticas, e (e) NUCA dinâmica. Apresentando o tempo em ciclos de relógio de acesso a dados dentro de cada banco de memória (KIM, 2002)

Dois modelos básicos mais utilizados atualmente de memória *cache* com arquitetura uniforme são de um ou dois níveis, que estão representados na Figura 3.14(a) e Figura 3.14(b) respectivamente. É possível ver o ganho que se tem em adotar um nível a mais na hierarquia de memória *cache* considerando o tempo de acesso a dados, dado em ciclos de relógio no centro de cada bloco de memória ilustrado.

Tratando de memórias de arquitetura não uniforme dois modelos estáticos foram propostos. O primeiro se assemelha bastante com a própria divisão da memória *cache* em sub-blocos como acontece internamente nas memórias UCA, porém sem os atrasos de espera por resposta de todos os sub-blocos (Figura 3.14(c)). Já na Figura 3.14(d), o modelo de NUCA apresenta um tipo diferente de interconexão, utilizando uma rede de interconexão em *chip* (NoC – *Network-on-Chip*) para interligar os diversos blocos ao processador.

Um modelo de memória NUCA dinâmica é apresentado na Figura 3.14(e). Nesta proposta, o conceito de migração de blocos está presente, visando assim, aumentar a velocidade do acesso a palavras de dados que são utilizadas várias vezes. Para isso, a política de migração de blocos adotada deverá ser capaz de reconhecer os blocos mais referenciados e assim escalonar estes para blocos de memória mais próximos ao processador.

Baseados nestas três propostas básicas de memórias NUCA serão discutidos nas próximas seções diversos aspectos relacionados a fim de esclarecer detalhes de implementação e problemas em aberto destas arquiteturas.

3.4.2.2. Mapeamento de Dados em Memórias NUCA

A forma de mapeamento dos dados na memória NUCA é o que define como os dados serão mapeados nos bancos e em quais bancos um dado pode residir. Diversas formas de mapeamento podem ser planejadas, uma vez que existe flexibilidade de utilização dos diversos bancos de memória (KIM, 2002).

Mesmo com a flexibilidade provida pela existência de diversos bancos de memória, as memórias NUCA estáticas (S-NUCA) prevêem apenas um mapeamento estático e único para cada endereço. Portanto, como em uma memória UCA, cada sub-banco está associado a diversos endereços os quais devem ser respeitados. Esta estratégia estática perde muito das vantagens de uma memória NUCA, pois dependendo do mapeamento estático adotado, certos dados sempre terão um alto tempo de acesso, podendo criar um gargalo no desempenho.

Já nas memórias NUCA dinâmicas (D-NUCA), uma total flexibilidade é planejada, onde cada dado pode ser mapeado em qualquer banco NUCA. Entretanto, esta estratégia deve considerar que o sobrecusto de pesquisa por dados será maior, uma vez que todos os bancos devem ser pesquisados, mesmo utilizando políticas de diretório de *tags* ou então fazendo *broadcast* do endereço para todos os bancos.

Uma solução intermediária para o mapeamento (KIM, 2002) de dados é chamada *spread sets*, que consiste em tratar diversos bancos de memória como sendo um conjunto associativo. Assim, cada conjunto estará espalhado por diversos bancos e cada banco representará uma via do conjunto associativo. Ainda utilizando esta solução de mapeamento, três políticas básicas de alocação de bancos são apresentadas na Figura 3.15, ilustrando uma memória NUCA de conjunto associativo de 4 vias, onde cada conjunto associativo está indicado pelas setas, e o bloco externo representa o processador.

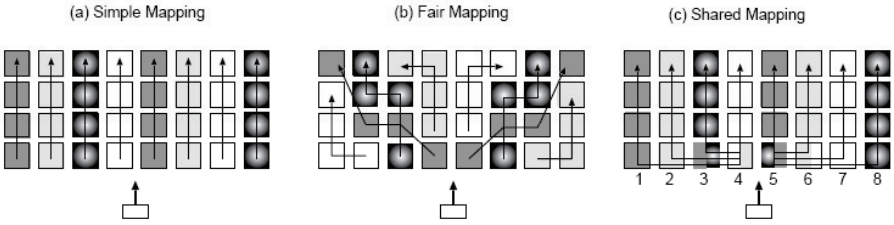

Figura 3.15. Diferentes políticas dinâmicas de alocação de blocos de memória NUCA: (a) mapeamento simples, (b) mapeamento balanceado e (c) mapeamento compartilhado

No *simple mapping* (mapeamento simples) apresentado na Figura 3.15(a) cada coluna de bancos representa um conjunto de bancos e assim, cada coluna compreende as vias de um conjunto associativo. Para a pesquisa de dados, primeiramente deve-se selecionar a coluna de bancos (conjunto), depois selecionar a via dentro da coluna, e então pesquisar pela *tag*. As desvantagens desse modelo é que o número de linhas pode não corresponder ao número de associatividade da memória e também a latência de acesso de um conjunto não é o mesmo para todas as vias de associatividade.

O mapeamento ilustrado na Figura 3.15(b) utiliza a política *fair mapping* (mapeamento equilibrado), esta política ataca os dois problemas apresentados pelo

simple mapping, com o sobrecusto de complexidade adicional. Esta política tenta equalizar o tempo de acesso às vias dentre os diversos conjuntos associativos, sendo este o principal ponto positivo desta política. Porém, ponto negativo desta equalização de tempos de acesso é o sobrecusto de roteamento de blocos inerente a esta política.

A política de *shared mapping* (mapeamento compartilhado), mostrado na Figura 3.15(c), tenta prover acesso rápido para todos os conjuntos associativos compartilhando os bancos próximos entre diversos conjuntos. Para que essa política funcione, os bancos próximos ao processador devem ser compartilhados por todas as vias dos conjuntos associativos o que reduz o número de endereços nestes bancos mais próximos.

Continuando no contexto de associatividade de memória NUCA, a proposta de (BARDINE, 2008a) utiliza técnicas de predição de utilização de vias associativas para desativar vias que tendem a não ser utilizadas durante a execução de determinadas aplicações. Isto cria uma memória NUCA dinâmica capaz de reduzir o consumo de potência sem grandes conseqüências no desempenho final.

3.4.2.3. Busca de Dados em Memórias NUCA

A política de busca de dados define como será pesquisado o conjunto de bancos a fim de encontrar determinado dado. Existem duas políticas distintas principais que podem ser utilizadas para a busca de dados entre os bancos de memória NUCA (KIM, 2002).

A primeira política chamada de *incremental search* (busca incremental), os bancos são pesquisados ordenadamente começando do banco mais próximo ao processador até que o bloco seja encontrado ou uma falta seja retornada pelo último banco do conjunto. Esta política reduz o número de mensagens nas interconexões da *cache* e mantém baixo o consumo de potência sob a penalidade de redução do desempenho do sistema.

Na segunda política chamada de *multicast search* (busca em múltiplos destinos), o endereço pesquisado é enviado a todos os bancos de uma só vez. Dessa forma, a pesquisa é feita toda em paralelo a não ser pelos diferentes tempos de roteamento da mensagem pela interconexão. Esta política aumenta o consumo de potência, mas ganha em desempenho.

Além das duas políticas principais, existem ainda algumas alternativas híbridas como a *limited multicast* (múltiplos destinos limitados), onde os bancos formam conjuntos, os quais são pesquisados seqüencialmente. Entretanto, os bancos de cada conjunto são pesquisados em paralelo. Na política híbrida *partitioned multicast* (múltiplos destinos particionados), os bancos são divididos em conjuntos que são pesquisados em paralelo, sendo que os bancos dentro de cada conjunto são pesquisados seqüencialmente. Neste caso, o paralelismo entre os bancos é similar ao das memórias UCA de múltiplos níveis.

3.4.2.4. Migração de Dados em Memórias NUCA

As políticas de migração de dados apresentam as condições sobre as quais deve haver uma migração de dados de um banco para outro. O objetivo desta migração de dados que acontece em memórias NUCA dinâmicas é maximizar o número de acertos na busca de dados nos bancos mais próximos ao processador (KIM, 2002). Uma política interessante seria a ordenação LRU entre as linhas nos conjuntos de bancos, com os

bancos mais próximos carregando as linhas MRU (*Most Recently Used*). O problema com esta política é que ela poderia levar a grandes movimentos de dados, desta maneira políticas factíveis de uso devem combinar o consumo de potência e o aumento na contenção sobre a migração de dados com os benefícios esperados de um ordenamento dos dados.

Uma solução dos problemas relacionados ao grande fluxo de migrações é a política conhecida como *generational promotion*. Nesta política ao acontecer um acerto na busca de dados, o bloco é enviado ao processador no mesmo tempo em que é migrado para o banco mais próximo ao processador, evitando assim movimentações de dados entre bancos distantes.

Ainda assim, existe outro problema a ser resolvido durante a migração de dados, que se refere ao que se deve fazer com os blocos retirados do banco mais próximo para dar lugar ao novo bloco. Para solucionar esse problema, existem duas políticas para tratar o chamado bloco vítima. A primeira política remove o bloco vítima da memória *cache*. Já na segunda política o bloco vítima troca de lugar com o bloco que ocupará seu lugar. Assim, evita-se que o bloco que poderá ser utilizado novamente seja retirado da memória *cache*, fazendo com que o mesmo se movimente mais lentamente para fora da memória *cache* de acordo com o tempo em que não esta sendo utilizado.

3.4.2.5. Interconexões de Blocos de Memórias NUCA

Diversos tipos de interconexão entre blocos de memória NUCA podem ser planejados. Nesta seção listamos alguns tipos conhecidos a fim de avaliar os pontos positivos e negativos de cada proposta.

O modelo de memória S-NUCA 1 apresentado em (KIM, 2002) (KIM, 2003) estrutura os blocos de tal forma que cada bloco possui um canal de comunicações privado como pode ser visto na Figura 3.16. Embora esta proposta de canais privados possa fornecer alto desempenho existem diversos problemas associados a esta abordagem, uma vez que o tamanho desta interconexão tende a ocupar grandes áreas do *chip* (KUMAR, 2005) caso haja muitos bancos a serem interligados, impossibilitando o aumento de tamanho da memória *cache*.

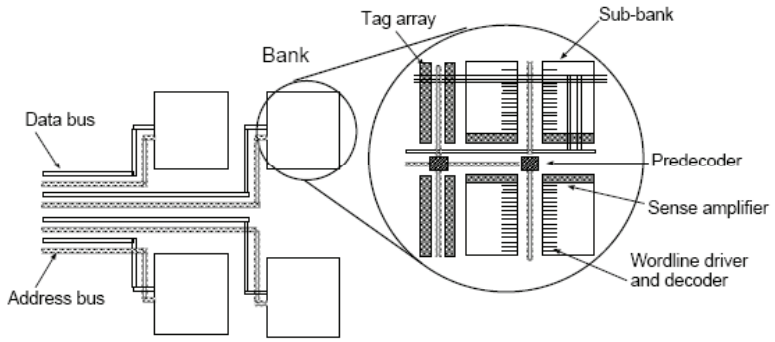

Figura 3.16. Estrutura de organização de memória S-NUCA1 (KIM 02)

A partir da proposta S-NUCA 1, as duas propostas seguintes, a S-NUCA 2 e a D-NUCA utilizam uma rede de interconexão *intra-chip* para interligar os diversos

blocos de memória, compartilhando assim os canais de comunicação e roteadores como pode ser visto na Figura 3.17. Por utilizar uma *Network-on-Chip* (NoC) para interligar os blocos, esta abordagem garante maior escalabilidade ao modelo, podendo facilmente aumentar o número de blocos sem maiores problemas. Além dessa vantagem, o modelo utilizando redes *intra-chip* pode ser projetado para utilizar mais agressivamente a estrutura de múltiplos bancos, aumentando ainda mais a velocidade de acesso aos bancos próximos ao processador, além de permitir migração de dados, priorizando blocos que são acessados mais freqüentemente.

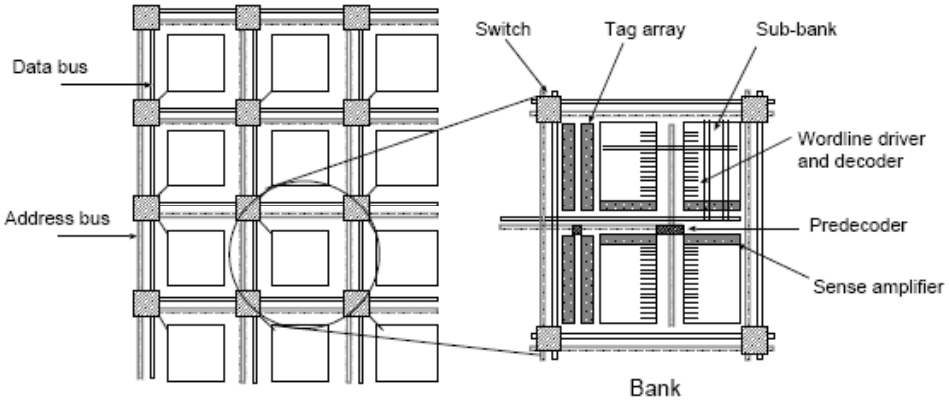

Figura 3.17. Estrutura de organização de memória S-NUCA2 e D-NUCA utilizando redes de interconexão *intra-chip* (KIM, 2002)

Uma mudança no número de portas no roteador da rede de interconexão foi proposta por (BARDINE, 2007) (BARDINE, 2008b), onde foi modificado o esquema de utilização de um roteador por bloco de *cache*, para a utilização de um roteador para cada 4 blocos de *cache*. Como pode ser visto na Figura 3.18, esta modificação garante *broadcast* mais eficiente, além disso, a redução no número de saltos até atingir todos blocos de memória pode representar aumento no desempenho e vazão de dados.

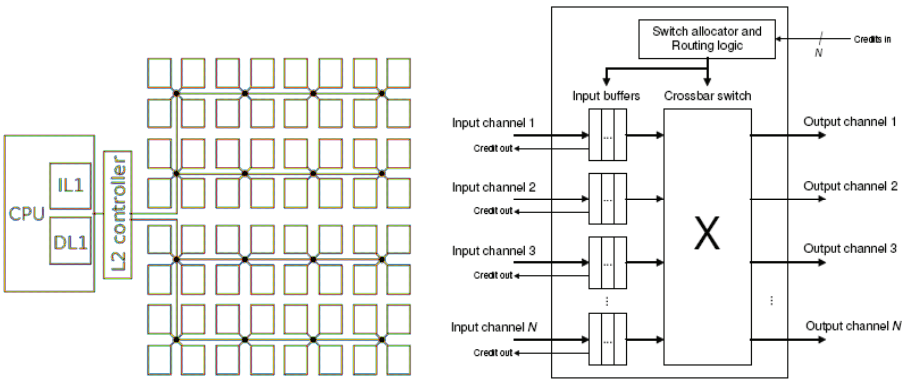

Figura 3.18. Modelo de NoC e roteador adaptado para interligar blocos de memória NUCA (BARDINE, 2007) (BARDINE, 2008b)

A proposta de uso de rede de interconexão híbrida visa diminuir o tempo de roteamento de endereços e dados entre o processador e os bancos de memória

(MURALIMANOHAR, 2007) (MURALIMANOHAR, 2008). Para isso, um modelo baseado em roteadores e barramentos compartilhados entre diversos núcleos é utilizado, como pode ser visto na Figura 3.19. Uma vez que o tempo de roteamento e transmissão de dados entre roteadores costumam corresponder a menos que um ciclo, a cada ciclo parte do tempo o sistema fica desocupado, aguardando novo ciclo de transmissão de dados. Baseado nessa afirmação a proposta híbrida ataca o problema reduzindo o número de roteadores e utiliza barramentos para aumentar o desempenho durante o *broadcast* de endereço nas buscas. Além disso, caso o barramento demore mais de um ciclo para completar a transmissão, os autores propõem o uso de barramentos *pipelined* como solução.

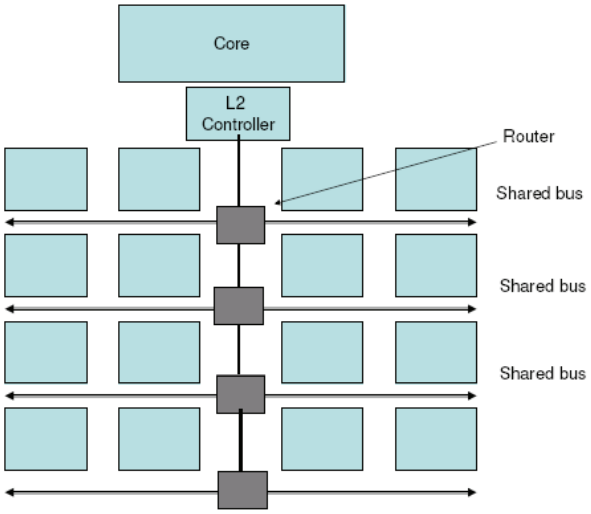

Figura 3.19. Topologia de rede híbrida para um sistema mono-processado (MURALIMANOHAR, 2007) (MURALIMANOHAR, 2008)

3.4.2.6. Memórias NUCA para processadores Multi-Core

Muitas das propostas de NUCA abordam apenas o caso da memória *cache* conectada a apenas um processador, uma vez que estas memórias *cache* no atual contexto já apresentam complexidade suficiente para que mesmo em sistemas monoprocessados existam diversas questões relacionadas ao projeto que ainda necessitem de maior estudo.

Porém, alguns trabalhos começam a avaliar métodos de utilizar as memórias *cache* NUCA em sistemas *multi-core*. Para estes sistemas, as principais questões de projeto são relacionadas ao sistema de migração de dados, organização dos blocos de memória e núcleos de processamento dentro do *chip*, políticas de controle de coerência de dados, além de controle de dados locais e compartilhados.

Portanto, projetos de processadores utilizando NUCA ainda são raros, porém, já começam a surgir idéias de projetos arquiteturais. Um dos primeiros estudos sobre o uso de NUCA em CMP (BECKMANN, 04) sugere a utilização de uma organização conhecida como CIM (*Cache In the Middle*) ilustrada na Figura 3.20. Nesta figura podemos ver 8 núcleos de processamento nas bordas do projeto, enquanto no centro

existe uma memória NUCA. O projeto de NUCA proposto utiliza três regiões de dados, a primeira, chamada de região local é a região mais próxima de cada núcleo, a região compartilhada formada de quatro bancos no centro é chamada de região central, nos meios entre as duas regiões existe ainda a região intermediária.

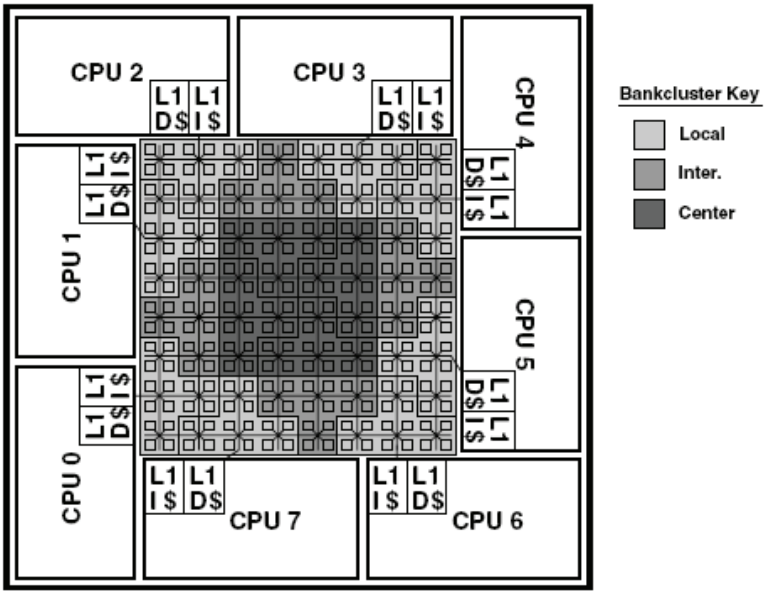

Figura 3.20. Projeto de CMP com NUCA estática e dinâmica combinadas (BECKMANN, 2004)

Mesmo esta primeira abordagem sendo bastante complexa, a idéia principal de utilizar a memória *cache* no centro de diversos núcleos de processamento é relativamente difundida e utilizada com algumas simplificações em outros trabalhos que visam avaliar aspectos relacionados, como controle de coerência (BOLOTIN, 2007) e controle de migração de dados (LI, 2008).

Outro projeto utilizando NUCA em CMP (GUZ, 2006) chama-se Nahalal, nesse projeto os processadores ficam centralizados em formato de anel enquanto as memórias *cache* os rodeiam e também ficam no centro. Neste projeto, as memórias localizadas no centro são compartilhadas entre todos os núcleos formando uma área comum de alta velocidade de acesso, enquanto as áreas fora do círculo são privadas para cada núcleo de processamento, como pode ser visto na Figura 3.21. Ainda com este projeto inicial de processadores organizados em anel, pode-se planejar a utilização de *clusters* desta organização, como apresentado também na Figura 3.21.

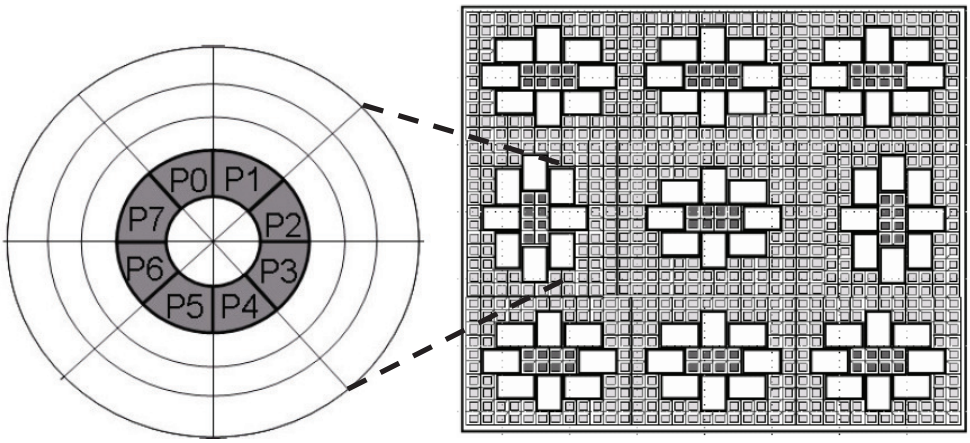

Figura 3.21. Projeto Nahalal de CMP com NUCA com espaços compartilhados no centro e isolados ao redor dos núcleos, apresentando o modelo com 8 núcleos e um projeto de cluster de Nahalal (GUZ, 2006)

Um terceiro tipo de organização para CMP utilizando NUCA (HUH, 2007) utiliza uma rede de interconexão de topologia *mesh* interligando os 16 núcleos de processamento e os bancos de memória. O projeto visa ainda diversos tipos de compartilhamento da memória *cache*, variando o grau de compartilhamento chamado também de SD (*Sharing Degree*) de 1 até 16 como ilustrado na Figura 3.22. Nesta proposta utilizando diversos graus de compartilhamento, o grau de compartilhamento 4 apresenta os melhores resultados de desempenho.

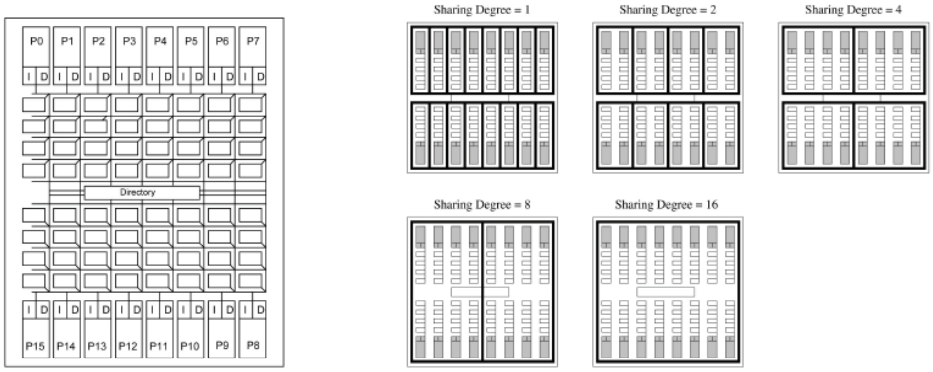

Figura 3.22. CMP utilizando NUCA em diversos graus de compartilhamento (HUH, 2007)

Mesmo com todos os esforços, os modelos de uso de NUCA para CMP apresentam problemas parecidos de coerência de dados, o que é acentuado na utilização de memórias NUCA dinâmicas, onde muitas vezes o custo extra de utilização de migração de dados e coerência de dados não é coberto pelo ganho de desempenho apresentado por estas técnicas (BECKMANN, 2004).

Portanto, podemos avaliar que os principais esforços para os próximos estudos sobre memórias NUCA devem estar direcionados para a redução do custo para o controle de coerência, políticas especiais de controle de coerência para NUCA, além do

estudo de uso de NUCA para processadores *many-core* com dezenas ou centenas de núcleos.

3.5. Conclusões

Através dos processadores *multi-core*, uma ampla oportunidade de pesquisa tem surgido em função do aumento da quantidade de núcleos. Decorrente desse aumento, a nova geração de processadores *many-core* também gera uma vasta relação de problemas com foco em interconexões e memórias *cache* conforme descrito neste curso.

A tendência apresentada pela literatura é que existem sérios problemas de escalabilidade nas soluções tradicionais de interconexões. Por este motivo, é necessário que as arquiteturas de *Networks-on-Chip* sejam projetadas para que atendam requisitos de desempenho, aumentando a vazão de pacotes e diminuindo o tempo de transmissão dos dados. Essas interconexões além de serem responsáveis pela comunicação entre núcleos, também são necessárias para comunicação com periféricos, principalmente as memórias *cache.* Como conseqüência, o compartilhamento das memórias *cache* distribuídas pelo *chip* do processador implicam em um projeto completamente diferente do que se tem visto nas soluções comerciais de processadores, ou seja, as NUCAs.

O que este curso procurou mostrar é que a área de arquitetura de processadores tem evoluído rapidamente, e que a nova geração de processadores *many-core* demanda estudo em áreas que antes não estavam diretamente associadas ao núcleo de um processador *single-core*. Resultados de pesquisas mostram que a academia e a indústria têm apresentado diversas soluções para os problemas em *many-core*. Portanto, o futuro da computação de alto desempenho também está associado a uma boa escolha de arquitetura de rede em *chip* e memórias NUCA. Muitos conceitos que ainda estão sendo formulados e validados implicarão diretamente no desenvolvimento e desempenho de diversos tipos de aplicações.

Referências

AGARWAL, A., Limits on Interconnection Network Performance, **IEEE Transactions on Parallel and Distributed Systems**, Vol. 2, No. 4, pp.398-412, October 1991.

AGGARWAL, N., et al., Isolation in Commodity Multicore Processors, **IEEE Micro**, pp. 49-59, June, 2007.

AHMADI, H., DENZEL, W. E., A Survey of Modern High-Performance Switching Technique, **IEEE Journal on Selected Areas in Communications**, Vol. 7, No. 7, pp. 1091-1103, September 1989.

ALVES, M. A. Z., FREITAS, H. C., NAVAUX, P. O. A., Investigation of Shared L2 Cache on Many-Core Processors for Scientific Parallel Applications, **Workshop on Many Cores**, March, 2009.

ANDERSON, G. A., JENSEN, E. D., Computer Interconnection Structures: Taxonomy, Characteristics, and Examples, **ACM Computing Surveys**, Vol. 7, No. 4, pp. 197-213, December 1975.

ANDRIAHANTENAINA, A., et al., SPIN: a scalable, packet switched, on-chip micro-network, **Design Automation and Test in Europe Conference and Exhibition (DATE)**, pp. 70–73, March 2003.

BARDINE, A., et al., Analysis of Static and Dynamic Energy Consumption in NUCA Caches: Initial Results, **ACM MEDEA Workshop - MEmory performance: DEaling with Applications, systems and architecture**, pp. 105-112, 2007.

BARDINE, A., et al., Leveraging Data Promotion for Low Power D-NUCA Caches, **11th IEEE EUROMICRO Conference on Digital System Design Architectures**, Methods and Tools, pp. 307-316, 2008.

BARDINE, A., et al., Performance Sensitivity of NUCA Caches to On-Chip Network Parameters, **20th IEEE SBAC-PAD - International Symposium of Computer Architectures and High Performance Computing**, pp. 167-174, 2008.

BARTIC, T. A., et al., Topology adaptive network-on-chip design and implementation, **IEE Proc. Comput. Digit. Tech.**, Vol. 152, No. 4, July 2005.

BECKMANN, B. M., WOOD, D. A., Managing Wire Delay in Large Chip-Multiprocessor Caches, **IEEE MICRO - 37th International Symposium on Microarchitecture**, 2004.

BENINI, L., MICHELI, G. D., Network-on-chip architectures and design methods, **IEE Proceedings Computers & Digital Techniques**, Vol. 152, Issue 2, pp.261-272, 2005.

BENINI, L., MICHELI, G. D., Networks on chips: a new SoC paradigm, **IEEE Computer**, vol. 1, pp. 70-78, January 2002.

BERTOZZI, D., et al., NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip, **IEEE Transactions on Parallel and Distributed Systems**, Vol. 16, No. 2, pp.113-129, February 2005.

BJERREGAARD, T. and MAHADEVAN, S., A Survey of Research and Practices of Network-on-Chip, **ACM Computing Surveys**, Vol. 38, No 1, pp. 1-51, March 2006.

BOLOTIN, E., et al., QNoC: QoS architecture and design process for network on chip, **The Journal of Systems Architecture, Special Issue on Networks on Chip**, 50 (2) pp.105–128, 2004.

BOLOTIN, E., et al., The Power of Priority: NoC based Distributed Cache Coherency, **IEEE NOCS - First International Symposium on Networks-on-Chip**, 2007.

BOUHRAOUA, A., ELRABAA, M. E., A High-Throughput Network-on-Chip Architecture for Systems-on-Chip Interconnect, **IEEE International Symposium on Systems-on-Chip**, pp. 1-4, 2006.

CHANG, M.-C. F., et al., RF/wireless interconnect for inter- and intra-chip communications, **Proc. of The IEEE**, vol. 89, no. 4, April 2001.

COMER, D. E., Network Systems Design Using Network Processors, **Prentice Hall**, 2003.

DALLY, W., TOWLES, B., Route packets, not wires: on-chip interconnection networks, **38th Design Automation Conference (DAC)**, pp. 684–689, June 2001.

DE ROSE, C., NAVAUX, P. O. A., Arquiteturas Paralelas, **Editora Sagra Luzzatto**, 2003.

FERNANDES, S., et al., IPNoSys: uma nova arquitetura paralela baseada em redes em chip, **Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD)**, pp.53-60, 2008.

FORSELL, M., A scalable high-performance computing solution for networks on chips, **IEEE Micro**, 22 (5), pp.46–55, 2002.

FREITAS, H. C., Arquitetura de NoC Multi-Cluster Programável e Reconfigurável para Suporte a Padrões de Comunicação em Processadores Many-Core, **Proposta de Tese de Doutorado, Universidade Federal do Rio Grande do Sul**, 18 de dezembro, 2008.

FREITAS, H. C., NAVAUX, P. O. A., Chip Multithreading: Conceitos, Arquiteturas e Tendências. (Desenvolvimento de material didático ou instrucional), **TI 1253**, **PPGC/UFRGS**, 2006.

FREITAS, H. C., SANTOS, T. G. S., NAVAUX, P. O. A., Design of Programmable NoC Router Architecture on FPGA for Multi-Cluster NoCs, **IET Electronics Letters**, Vol. 44, No. 16, pp. 969-971, 31st July 2008.

FREITAS, H. C., SANTOS, T. G. S., NAVAUX, P. O. A., NoC Architecture Design for Multi-Cluster Chips, **IEEE International Conference on Field Programmable Logic and Applications (FPL)**, Heildelberg, Germany, pp. 53-58, 2008.

GRECU, C., et al., A scalable communication-centric SoC interconnect architecture, **IEEE International Symposium on Quality Electronic Design (ISQED)**, 2004.

GUZ, Z., et al., Nahalal: Memory Organization for Chip Multiprocessors, **Technical Report CCIT 600, Technion Department of Eletrical Engineering**, September, 2006.

HILL, M. D., MARTY, M. R., Amdahl's Law in the Multicore Era, **IEEE Computer Society**, pp.33-38, July 2008.

HILTON, C. and NELSON, B., PNoC: A Flexible Circuit-Switched NoC for FPGA-based Systems, **IEE Proceedings of Computer & Digital Techniques**, Vol. 153, No. 3, pp. 181-188, May 2006.

HO, R., et al., The Future of Wires, **Proceedings of the IEEE**, Vol. 89, No. 4, April 2001.

HO, W. H., PINKSTON, T. M., A Design Methodology for Efficient Application-Specific On-Chip Interconnects, **IEEE Transactions on Parallel and Distributed Systems**, Vol. 17, No. 2, pp.174-190, February 2006.

HUH, J., et al., A NUCA Substrate for Flexible CMP Cache Sharing, **IEEE Transactions of Parallel and Distributed Systems**, Vol. 18, N. 8, pp. 1028-1040, August, 2007.

HWANG, K., XU. Z., Scalable Parallel Computing: Technology, Architecture, Programming, **Boston, WCB**, 1998.

INTEL, 80-Core Programmable Processor First to Deliver Teraflops Performance, **Intel Platform Research, http://www.intel.com/research/platform/index.htm**, Acesso em: out. 2007.

INTEL, IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture, **Intel**, March 2006.

INTEL, IXP1200 Network Processor Family, Hardware Reference Manual, **Intel**, December, 2001.

JOSEPH, N., et al., RECONNECT: A NoC for Polymorphic ASICs Using a Low Overhead Single Cycle Router, **IEEE**, pp.251-256, 2008.

KALLA, R., SINHAROY, B., TENDLER, J. M., IBM Power5 Chip: A Dual-Core Multithreaded Processor, **IEEE MICRO**, Vol. 24, Issue 2, p. 40-47, 2004.

KARIM, F., NGUYEN, A., DEY, S., An interconnect architecture for network systems on chips, **IEEE Micro,** 22 (5), pp.36-45, 2002.

KEYES, R. W, Moore's Law Today, **IEEE Circuits and Systems Magazine**, Vol. 8, Issue 2, pp.53-54, 2008.

KEYES, R. W., Fundamental Limits of Silicon Technology, **Proceedings of the IEEE**, Vol. 89, No. 3, pp.227-239, March, 2001.

KIM, C., BURGUER, D., KECKLER, S. W., An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches, **ACM ASPLOS X – 10th International Conference on Architectural Support for Programming Languages and Operational Systems**, 2002.

KIM, C., BURGUER, D., KECKLER, S. W., Nonuniform Cache Architectures for Wire-Delay Dominated On-Chip Caches, **IEEE Micro**, pp.99-107, November-December, 2003.

KONGETIRA, P., et al., Niagara: a 32-way multithreaded Sparc processor, **IEEE MICRO**, Volume 25, Issue 2, pp.21-29, March-April 2005.

KREUTZ, M., MARCON, C., CARRO, L., CALAZANS, N., SUSIN, A. A., Energy and Latency Evaluation of NOC Topologies, **International Symposium on Circuits and Systems (ISCAS)**, Kobe, pp. 5866-5869, 2005.

KUMAR, R., Zyuban, V., Tullsen, D.M., Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling, **32nd International Symposium on Computer Architecture (ISCA)**, pp.408-419, June 2005.

LENG , X., et al., Implementation and Simulation of a Cluster-based Hierarchical NoC Architecture for Multi-Processor SoC, **IEEE International Symposium on Communiations and Information Technology**, pp. 1163-1166, 2005.

LI, F., KANDEMIR, M., IRWIN, M. J., Implementation and Evaluation of a Migration-Based NUCA Design for Chip Multiprocessors, **ACM SIGMETRICS**, pp. 449-450, 2008.

LIANG, J., SWAMINATHAN, S., TESSIER, R., aSOC: A scalable, single-chip communications architecture, in: **IEEE International Conference on Parallel Architectures and Compilation Techniques**, pp. 37–46, October 2000.

MARESCAUX, T., Networks on chip as hardware components of an OS for reconfigurable systems, **Field-Programmable Logic and Applications (FPL)**, September 2003.

MERCALDI-KIM, M., Davis, J. D., Oskin, M., Austin, T., Polymorphic On-Chip Networks, **IEEE International Symposium on Computer Architecture (ISCA)**, pp. 101-112, 2008.

MILLBERG, M., The Nostrum backbone - a communication protocol stack for networks on chip, **Proceedings of the VLSI Design Conference**, January 2004.

MORAES, F., et al., HERMES: an infrastructure for low area overhead packet-switching networks on chip, **Integration the VLSI Journal**, pp.69-93, 2004.

MURALIMANOHAR, N., BALASUBRAMONIAN, R., Interconnect Design Considerations for Large NUCA Caches, **ACM ISCA – International Symposium on Computer Architecture**, pp. 369-380, 2007.

MURALIMANOHAR, N., BALASUBRAMONIAN, R., JOUPPI, N. P., Architecting Efficient Interconnects for Large Caches with CACTI 6.0, **IEEE Micro**, pp.69-79, January-February, 2008.

NIEMANN, J., PORRMANN, M., RÜCKERT, U., A Scalable Parallel SoC Architecture for Network Processors, **IEEE Annual Symposium on VLSI**, pp. 311-313, 2005.

OGRAS, U. Y. et al., On-Chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-Point, Bus, and Network-on-Chip, **ACM Transactions on Design Automation of Electronics Systems**, Vol. 12, No. 3, Article 23, August 2007.

OLUKOTUN, K. et al., The Case for a Single-Chip Multiprocessor, **7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)**, pp.2-11, 1996.

OLUKOTUN, K., HAMMOND, L., The Future of Microprocessors, **ACM Queue**, Vol. 3, Issue 7, pp. 26-29, September 2005.

PATTERSON, D. A., HENNESSY, J. L., Organização de Computadores a Interface Hardware/Software, **Campus**, terceira edição, 2005.

PIONTECK, T., et al., Adaptive Communication Architectures for Runtime Reconfigurable Systems-on-Chip, **Parallel Processing Letters**, Vol. 18, No. 2, pp.275-289, 2008.

RIJPKEMA, E., GOOSSENS, K., RADULESCU, A., Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip, **Design, Automation and Test in Europe (DATE)**, pp. 350–355, March 2003.

SGROI, M., Addressing the system-on-chip interconnect woes through communication-based design, **38th Design Automation Conference (DAC)**, pp. 667–672, June 2001.

SIGÜENZA-TORTOSA, D., NURMI, J., Proteo: a new approach to network-on-chip, **IASTED International Conference on Communication Systems and Networks (CSN)**, September 2002.

SPRACKLEN, L., ABRAHAM, S.G., Chip Multithreading: Opportunities and Challenges, **International Symposium on High-Performance Computer Architecture (HPCA)**, pp.248-252, February 2005.

STALLINGS, W. Computer organization and architecture: designing for performance, 4th Edition, **Prentice Hall**, 1996.

STOJMENOVIC, I., Honeycomb Networks: Topological Properties and Communication Algorithms, **IEEE Transactions on Parallel and Distributed Systems**, Vo. 8, No. 10, pp.1036-1042, October 1997.

THOZIYOOR, S., MURALIMANOHAR, N., JOUPPI, N. P., CACTI 5.0, **Advanced Architecture Laboratory - HP Laboratories**, 2007.

WANG, Y., ZHAO, D., Design and Implementation of Routing Scheme for Wireless Network-on-Chip, **IEEE International Symposium on Circuits and Systems (ISCAS)**, pp. 1357-1360, May 2007.

WIKLUND, D., LIU, D., SoCBUS: switched network on chip for hard real time systems, **International Parallel and Distributed Processing Symposium (IPDPS)**, April 2003.

WILTON, S. J. E., JOUPPI, N. P., An Enhanced Access and Cycle Time Model for On-Chip Caches, **WRL Research Report 93/5, Western Research Laboratory**, July, 1994.

WOLF, W., The Future of Multiprocessor System-on-Chip, **Proceedings of the DAC**, June, 2004.

ZEFERINO, C. A., SUSIN, A. A., SoCIN: A Parametric and Scalable Network-on-Chip, **IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)**, pp. 169-174, 2003.