# Arquiteturas Multicore*

**Douglas Camargo Foster**[1]

[1]Programa de Pós-Graduação em Informática (PPGI) – Universidade Federal de Santa Maria (UFSM)
Av. Roraima, 1000 – Centro de Tecnologia (CT) –
97.105-900 – Camobi, Santa Maria – RS – Brasil
dcfoster@mail.ufsm.br

***Abstract.*** *This article aims to present a review on multicore architectures, discussing the main definitions and characteristics of the subject. Addressed the main aspects on the development of communication networks, through the implementation of NoC (Networks-on-Chip), and it presents the definitions of architecture and organization of cache memory. Finally, summarizing the main points in research and development related to multicore architectures, NOC, and architectures of cache memory.*

***Resumo.*** *Este artigo tem como objetivo apresentar uma revisão sobre arquiteturas multicore, discutindo as principais definições e características inerentes a este assunto. São abordados os principais aspectos sobre o desenvolvimento de redes de comunicação, através da aplicação de NoC (Networks-on-Chip), e apresenta-se as definições de arquiteturas e organização de memória cache. Ao final, sintetiza-se os principais pontos de pesquisa e desenvolvimento que relacionam arquiteturas multicore, NoC, e arquiteturas de memória cache.*

## 1. Introdução

Desde a última década, a evolução dos processadores acontece de forma surpreendente. Isto se deve principalmente ao aumento na densidade de integração de transistores em áreas cada vez menores. Neste nível de integração, é necessário explorar técnicas de paralelismo para efetivamente utilizar ao máximo o desempenho dos processadores. *Pipeline*, superescalaridade, *multithreading*, são algumas técnicas aplicadas a exploração de paralelismo para aumento de desempenho.

*Pipeline* superescalar, aplica o processamento das instruções divididos em estágios, possibilitando a execução de mais de uma instrução por ciclo, e para isso, aumenta o número de unidades funcionais e técnicas para solucionar falsas dependências entre instruções. Os processadores superescalares são capazes de aumentar o desempenho na execução de cargas de trabalho com alto paralelismo no nível de instruções. O suporte a múltiplas *threads* foca a exploração de paralelismo de maior granularidade, explorando o paralelismo além do nível de instruções, mas também no nível de fluxo de instruções (*threads*). Isto aumenta a vazão de *threads* (possibilitando que mais de uma *thread* possa ser executada ao mesmo tempo) diferente da superescalaridade onde a vazão é de instruções de uma única *thread*. São várias as

* Artigo apresentado na disciplina ELC-888 – Arq. De Sist. de Alto Desempenho, no PPGI-UFSM. Julho de 2009

técnicas para exploração do paralelismo no nível de *threads*, sendo que a mais conhecida é a SMT (*Simultaneous Multithreading*) que tem suporte em uma arquitetura superescalar. [ALVES 2007]

O uso de processadores com múltiplos núcleos (*Chip Multiprocessors* – CMPs, ou simplesmente *multicore*) vem sendo consolidada como um bom método para aumento do desempenho de computação. Com vários núcleos, um processador passa a ter suporte nativo a várias *threads*, sendo os núcleos superescalares ou não. Os CMP não necessitam ser tão rápidos quanto chips *single-core*, mas aumentam desempenho devido ao processamento paralelo. [ALVES 2007]

Devido ao aumento do número de núcleos, há a necessidade de uma rede de comunicação de alto desempenho entre eles. A aplicação de NoC (*Network on Chip*) como solução da comunicação entre os diversos elementos do processador *multicore* (processadores, memórias, módulos especializados) apresenta-se como solução muito eficiente. A utilização desta tecnologia pode reduzir a complexidade do projeto da rede de comunicação devido a sua estrutura regular e controlada. Também provê separação entre os módulos de computação e comunicação, reusabilidade dos módulos através de interfaces padrões, gerência questões de sincronização, e portanto, aumenta a eficiência do processamento.[KOLODNY 2006] . Outro fator de relevante importância em projetos de processadores *multicore* está relacionado ao projeto de memória cache para os núcleos. Com o aumento do número de núcleos integrados em um único chip, aumenta o tráfego de dados e, conseqüentemente, a quantidade de acessos a memória. Isto implica em organizar a hierarquia de memória, compartilhamento e quantidade de memória cache dedicada para cada processador. Atualmente, estas duas questões são muito relevantes em projetos de processadores *multicores.*

Este artigo divide-se em 5 seções. A seção 2 trata dos conceitos e definições de processadores *multicores*. A seção 3 aborda os desafios de projetos de NoC's. A seção 4 trata da organização e arquiteturas de memória cache. Ao final, na seção 5, faz-se apontamentos acerca de projetos de processadores *multicore*.

## 2. Arquiteturas Multicore

O desenvolvimento de processadores *multicore* tem como motivação duas realidades: o grande consumo de potência e as limitações de manter a evolução de desempenho, como prevê a Lei de Moore (aumento do grau de integração relacionado ao aumento de performance). O aumento de desempenho alcançado pelos processadores se baseia na aplicação de técnicas que exploram o paralelismo de instruções, e também pelo aumento de freqüência de operação. Entretanto, o consumo de potência também atinge níveis impraticáveis, já que é proporcional ao aumento da freqüência de trabalho. Os processadores *multicores*, já usados em outras áreas, se tornaram alternativas para computadores de propósito geral.

Uma arquitetura *multicore* é geralmente um multiprocessamento simétrico (*Simultaneous Multiprocessing* - SMP, em que múltiplos núcleos de processador tipicamente compartilham um segundo ou terceiro nível de cache comum e interconectado), implementado em um único circuito VLSI (*Very Large Scale Integration*).

O objetivo de um sistema *multicore* é permitir maior utilização de paralelismo

no nível de *threads*, especialmente para aplicações que façam pouco uso do paralelismo a nível de instrução para fazer um bom uso de processadores superescalares. Os projetos de software estão cada vez mais utilizando múltiplos processos e *threads,* como aplicações de multimídia e o amplo uso de ferramentas de visualização [OLUKOTUN 1996.]. Em processadores *single-core* que executam vários programas, este define diferentes períodos para execução de cada programa, e isto pode gerar erros, conflitos ou queda de performance, quando precisa desempenhar muitas tarefas simultaneamente. As arquiteturas *multicores* apresentam a possibilidade de distribuir as diferentes tarefas pelos vários núcleos, obtendo maior eficácia (*troughput*) do sistema e desempenho aprimorado de aplicativos, mesmo executando vários aplicativos simultaneamente. Surgem novas possibilidades de extração de paralelismo, e então, a possibilidade de extrair paralelismo *inter-thread*. Esta característica pode ser explorada por máquinas capazes de executar mais de uma *thread* simultaneamente. Outras arquiteturas capazes de explorar essa características são as máquinas multiprocessadas e máquinas SMT (*Simultaneous Multithreaded*).

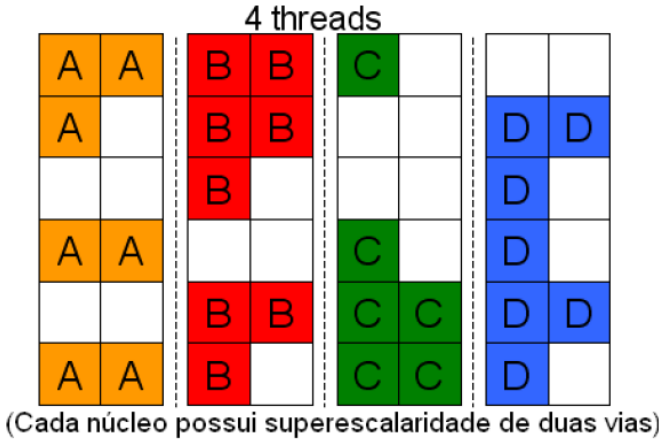


**Figura 1. Execução de múltiplas *threads* em um processador *multicore.* [FREITAS et al 2009]**

A Figura 1 representa um exemplo de uma arquitetura composta por 4 núcleos (*quad-core*), em que cada núcleo possui um *pipeline* superescalar de duas vias. Quando comparado com outras arquiteturas, cada um dos núcleos pode ser considerado relativamente simples. Neste exemplo, o objetivo é suportar tanto paralelismo em nível de instrução quanto no nível de *threads*, mas estas somente são suportadas pela existência de mais de um núcleo, já que cada núcleo é responsável pela execução de apenas uma *thread*. Processadores *multicore* que suportam a execução *multithreading* são chamados de *Chip Multithreading* (CMT). [SPRACKLEN 2005][FREITAS 2006]

## 3. Arquiteturas de Redes em Chip (NoC – *Network on Chip)*

O projeto de redes de comunicação entre os elementos de um processador *multicore* encara as limitações de modularidade e escalabilidade, com especificações mais rígidas e complexas, acerca da organização da comunicação entre os elementos. O desenvolvimento e aplicação de NoCs apresenta-se como uma ótima resposta para esta questão. A premissa básica é simples: o desenvolvimento das interconexões em chip devem seguir os mesmos princípios que são aplicados em projetos de rede de comunicação em um nível macroscópico, que demonstram escalabilidade sustentável,

aumento de performance exponencial, notável confiança e robustez. A natureza modular e escalável de NoCs e seu apoio para uma eficiente comunicação on-chip potencialmente levam-na para aplicação em processadores *multicores* caracterizados pela alta complexidade estrutural e diversidade funcional. Por um lado, esses recursos devem ser devidamente abordados por meio de nova concepção de metodologias, enquanto por outro lado, mais esforços devem ser dedicados a modelagem de arquiteturas de NoCs e integrá-los em um único ambiente de modelamento e simulação combinando ambos elementos de processamento e comunicação.[BENINI 2005]

A Figura 2 mostra exemplos de estruturas de comunicação aplicados a SoCs (*Systems-on-Chip*), mas que utilizam os mesmos conceitos quando desenvolvidos para NoCs. O barramento é desenvolvido sob conceitos bem compreendidos e fácil de ser modelado. Em um sistema *multicore* altamente interconectado, porém, pode rapidamente tornar-se uma comunicação não eficiente. Quanto mais unidades são adicionados a ele, o consumo de potência na utilização da comunicação em cada evento cresce. O uso de *crossbar* supera algumas das limitações do barramento. No entanto, não é escalável e, como tal, é uma solução intermediária. O uso de *links* dedicados ponto-a-ponto são ótimas em termos de disponibilidade de largura de banda, latência, potência e usos que são concebidos especialmente para este determinado fim. Além disso, eles são simples para desenvolver e verificar e de fácil modelagem. Mas o número de ligações necessárias aumenta exponencialmente como o número de núcleos aumenta. Assim, problemas acerca de área e roteamento surgem. Do ponto de vista do esforço de desenvolvimento, em sistemas com menos de 20 núcleos, uma estrutura de comunicação *ad hoc* é viável. Mas, como a tendência é aumentar o número de elementos integrados e o tempo de projeto diminuir, a necessidade de soluções mais generalizadas torna-se necessária. Para a máxima flexibilidade e escalabilidade, é explorada a idéia de uma comunicação segmentada, dividida. Este conceito traduz-se em uma rede de roteamento de dados composto de *links* de comunicação e roteadores que são implementados no chip.[BJERREGAARD 2006]

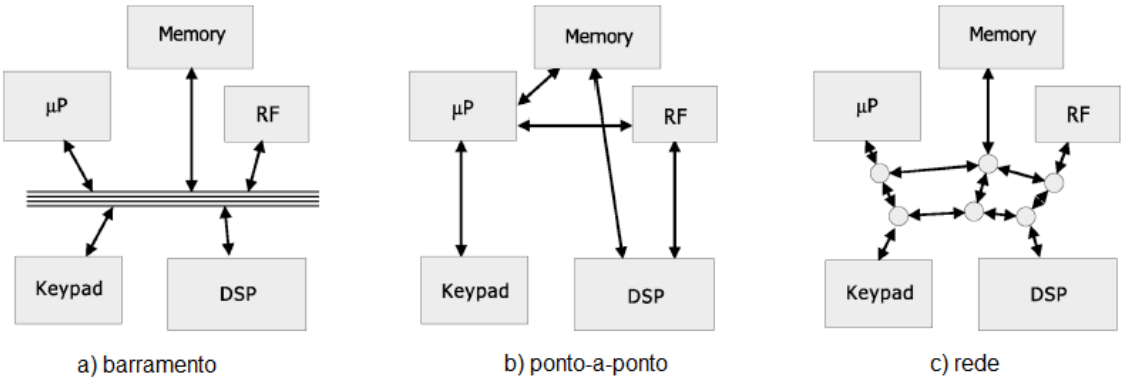


**Figura 2. Exemplos de comunicações em SoCs. a) comunicação baseada em barramento, b) *links* ponto-a-ponto, c) estrutura de rede. [BJERREGAARD 2006]**

Basicamente, a NoC é composta por três elementos [FREITAS et al 2009]:

- Roteador: responsável pela interconexão da rede, pela definição de rotas, pelo controle de fluxo, qualidade de serviço e 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;

- Links de comunicação: são os elementos que interligam os roteadores . Responsáveis pelo caminho entre fonte e destino dos pacotes a serem trafegados. A forma como os roteadores estão interconectados pelos links dá origem à topologia da rede;
- Interface de rede: também 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).

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]. 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. Dentre as topologias estudadas e desenvolvidas para arquiteturas NoC, as 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). [FREITAS et al 2009]

## 4. Memória Cache: Arquitetura e Organização

A memória cache é um bloco de memória para o armazenamento temporário de dados que possuem uma grande probabilidade de serem utilizados novamente pelo processador. A vantagem principal consiste em evitar o acesso ao dispositivo de armazenamento, que têm um período de acesso relativamente maior, armazenando os dados em meios de acesso mais rápidos. Basicamente segue-se duas arquiteturas de projeto de memória cache: UCA (*Uniform Cache Architecture*) e NUCA (*Non-Uniform Cache Architecture*). Dois modelos básicos mais utilizados atualmente de memória *cache* com arquitetura uniforme são de um ou dois níveis. Ao adotar um nível a mais na hierarquia de memória *cache* obtém-se um ganho no tempo de acesso aos dados. 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 em memórias UCA, porém sem os atrasos de espera por resposta de todos os sub-blocos; o segundo modelo proposto de NUCA utiliza um tipo diferente de interconexão, utilizando uma NoC para interligar os diversos blocos ao processador.

Para manter uma estrutura regular no projeto da memória UCA deve-se empregar estratégias de mapeamento. As posições de memória cache e de memória principal são divididas em “blocos”, ou “páginas”, de mesmo tamanho. Assim é possível a leitura de um conjunto de posições de memória principal para um bloco na memória cache. A identificação dos blocos de memória principal que encontram-se

mapeados em blocos de memória cache é realizada através do uso de "*tags*" (etiquetas), obtidos a partir do endereço de memória principal, segundo a forma de mapeamento empregada. Quando processador deseja acessar um dado que está no local de armazenamento, primeiramente verifica a cache. Se uma entrada é localizada com uma etiqueta correspondente ao dado requisitado, então o elemento da cache é utilizado ao invés do dado original no dispositivo de armazenamento. Essa situação é conhecida como *cache hit*. Um modo usual de mapeamento de dados na memória *cache* é o mapeamento direto, em que 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*. Entretanto apresenta menor flexibilidade na alocação, visto que os blocos tem posição pré-determinada. O método de mapeamento completamente associativo permite qualquer bloco de memória principal ser mapeado em qualquer bloco da cache. O *tag* é utilizado para a identificação do bloco mapeado. Em função deste mecanismo, apresenta grande flexibilidade de alocação, mas necessita de componentes adicionais para localizar bloco de forma rápida através de uso de busca associativa, comparando *tag* do endereço procurado com os *tags* de todos os blocos armazenados em cache. Um alternativa equilibrada entre estes métodos anteriores é o uso de mapeamento associativo por conjunto, em que os blocos de memória cache são divididos em $S$ conjuntos, com $E = M/S$ blocos por conjunto (sendo $M$ o número total de blocos em memória cache). Cada bloco de memória principal a ser mapeado é associado a um conjunto nos moldes do mapeamento direto, mas como um conjunto possui diversos blocos, utiliza-se um *tag* para identificar o bloco do conjunto que foi utilizado para o mapeamento. Diminuição da complexidade de localização dos blocos mapeados em relação ao método completamente associativo e mantém-se uma flexibilidade de alocação maior do que a encontrada no método diretamente mapeado. [FREITAS et al 2009]

Durante a busca de um bloco de dados para a memória *cache*, esta deve escolher qual será a posição onde o bloco será gravado (qual bloco de dados será substituído) através de uma política de substituição de dados. Quando se aplica o modo de mapeamento direto, não há escolha a ser feita para a substituição de dados [STALLINGS 1996], devendo apenas gravar o novo dado em seu endereço referente. Ao aplicar modo de mapeamento completamente associativo ou associativo por conjuntos, há uma escolha a ser feita sobre qual bloco será substituído pelo novo bloco de dados. Existem quatro estratégias principais:

- Aleatória – bloco a ser substituído será escolhido aleatoriamente, ou seja, o sistema irá escolher o bloco alvo ao acaso.
- Menos Recentemente Usado (*Least Recently Used* – LRU) – baseia-se em reduzir a chance de descarte dos blocos usados mais recentemente. A informação sobre quão recentemente o bloco foi utilizado deverá permanecer junto ao mesmo para a decisão.
- Menos Freqüentemente Usado (*Least Frequently Used* – LFU) – substituição dos blocos que foram menos referenciados. Esta informação sobre quão freqüente cada bloco foi referenciado permanece junto a cada bloco para a decisão.
- Primeiro a Entrar, Primeiro a Sair (*First In, First Out* – FIFO) – devido a complexidade do cálculo do bloco que não é utilizado por um período mais

longo, esta estratégia baseia-se apenas nas informações sobre os dados mais antigos para achar o bloco alvo a ser substituído.

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. [FREITAS et al 2009]

Em projetos de memórias NUCA, os métodos de mapeamento podem ser planejados de diferentes formas por ter uma organização dividida em diversos bancos. O método estático (S-NUCA – *Static* NUCA) utiliza apenas mapeamento estático para cada endereço, mas isto acarreta que dependendo da estrutura de mapeamento, alguns dados podem ter um alto tempo de acesso, degradando performance. Em memória D-NUCA (*Dynamic*-NUCA) há uma total flexibilidade em que o dado pode ser escrito em qualquer banco NUCA. A penalização está no processo de localização, pois todos os bancos devem ser pesquisados. 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. Aplicando esta solução, há 3 políticas de alocação de bancos: *simple mapping* (mapeamento simples), que aplica a seleção da coluna de banco e depois a via da coluna para a pesquisa de dados; *fair mapping* (mapeamento equilibrado), que objetiva a equalização do tempo de acesso às vias dentro dos diversos conjuntos associativos; *shared mapping* (mapeamento compartilhado), que busca prover acesso rápido a todos os conjuntos associativos compartilhando os bancos próximos entre diversos conjuntos.

A operação de busca de dados pode ser aplicada a partir de 2 formas básicas. A primeira 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, reduzindo o número de mensagens nas interconexões e mantém baixo o consumo de potência e reduzindo o desempenho do sistema. A segunda política chamada de *multicast search* (busca em múltiplos destinos), o endereço pesquisado é enviado a todos os bancos de uma só vez: a pesquisa é feita toda em paralelo a não ser pelos diferentes tempos de roteamento da mensagem pela interconexão. Conseqüentemente aumenta o consumo de potência, mas ganha em desempenho. Além das duas políticas principais, existem soluções híbridas como a *limited multicast* (múltiplos destinos limitados), em que os bancos formam conjuntos pesquisados seqüencialmente, e os bancos de cada conjunto são pesquisados em paralelo, e a solução *partitioned multicast* (múltiplos destinos particionados), que os bancos são divididos em conjuntos que são pesquisados em paralelo, sendo que os bancos dentro de cada conjunto são pesquisados seqüencialmente (similar a UCA de múltiplos níveis). Afim de aumentar acertos na busca dos dados, aplica-se um migração de dados de um banco para outro em D-NUCAs, principalmente para bancos mais próximos ao processador.

As interconexões dos blocos em NUCA podem seguir projetos em que a estrutura dos dados permita um canal de comunicação dedicado para cada bloco obtendo alto desempenho, mas ocupando grande área em *chip*. A aplicação de NoCs, compartilhando o uso dos roteadores e canais de comunicação para ligação entre os blocos, permite aumentar o número de blocos e manter uma alta velocidade de acesso

aos bancos próximos ao processador.

### 4.1. Coerência de Cache

Um importante cuidado que deve-se ter é em relação ao compartilhamento dos dados entre os núcleos em arquiteturas *multicore*. Múltiplas cópias de mesma posição de memória podem existir em diferentes *caches*. Cada processador atualiza sua cópia local, não se preocupando com a existência de outras cópias em outros processadores, e se essas cópias existirem, cópias de mesmo endereço de memória poderão possuir valores diferentes. Uma alternativa para eliminar o problema é não permitir que dados compartilhados para operações de escrita sejam colocados nas caches do sistema: somente instruções e dados privados são cacheáveis e dados compartilhados são considerados não-cacheáveis. Uma arquitetura multiprocessada com *caches* privadas é coerente se e somente se uma leitura de uma posição *x* de memória efetuada por qualquer processador *i* retorne o valor mais recente desse endereço. Toda vez que uma escrita for efetuada por um processador *i* em um endereço de memória *x*, tem que ser garantido que todas as leituras subseqüentes de *x*, independente do processador, forneçam o novo conteúdo de *x*.

Para manter a coerência dos dados são adotadas políticas de escrita (*write-through e write-back*) mas que não são suficientes para manter a coerência em sistemas multiprocessados compostos por caches distribuídas. Para contornar este problema de inconsistência de dados são utilizados soluções baseadas em software e hardware. Estas soluções baseiam-se na utilização de protocolos centralizados e distribuídos (*broadcast*) para manter a coerência do sistema. Dentre estes protocolos destacam-se os protocolos de invalidação (*write-invalidate*) e de atualização (*write-update*) dos dados.

## 5. Conclusões

A expressão *multicore* é comumente utilizada para expressar processadores com determinado número de núcleos (ao redor de 10 núcleos). O termo mais atual e utilizado para referenciar arquiteturas que envolvem dezenas ou centenas de núcleos é *many*-core. A partir da integração de dezenas de núcleos, começam a surgir restrições de projetos de rede de comunicação e organização da memória cache.

A aplicação de NoCs em arquiteturas de processadores *multicore* abre uma vasta área de pesquisa e desenvolvimento. Devido ao aumento de elementos integrados dentro de um único chip, este tipo de solução deve buscar a melhor relação entre desempenho, área e consumo de potência. Dentre as várias soluções que se apresentam na literatura, deve-se buscar a melhor alternativa para cada aplicação.

A organização e arquitetura de memória cache também deve ser aprimorada visto o grande número de dados que são compartilhados entre os vários elementos do processador. Principalmente no que diz respeito a coerência de dados e tempo de acesso . Soluções mais atuais preocupam-se com o tamanho das memórias e localização, o que influencia diretamente nas características citadas anteriormente.

## Referências

ALVES, M. A. Z., FREITAS, H. C., WAGNER, F. R., e NAVAUX, P. O. A. (2007) "Influência do Compartilhamento de *Cache* L2 em um *Chip* Multiprocessado sob Cargas de Trabalho com Conjuntos de Dados Contíguos e Não Contíguos", VIII

Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD), Gramado, RS, Brasil, 2007.

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

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

KOLODNY, Avinoam. (2006), "What is Network-on-Chip?", disponível em: http://www.sigda.org/newsletter/2006/eNews_060415.html, ACM/SIGDA E-Newsletter, Vol. 36, No. 8, 15 de Abril 2006. Acessado em 25 de Junho de 2009.

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

FREITAS, H. C. ; ALVES, M. A. Z. ; NAVAUX, P. O. A. (2009) "NoC e NUCA: Conceitos e Tendências para Arquiteturas de Processadores Many-Core." 9ª Escola Regional de Alto Desempenho (ERAD). Caxias do Sul, RS, Brasil, p. 5-37, 2009.

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

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

BERTOZZI, D., et al., (2005) "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, 2005.

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

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

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

KIM, C., BURGUER, D., KECKLER, S. W., (2002) "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.