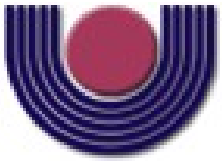

UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Simulação de modelos de coerência de cache utilizando plataformas virtuais

Marcelo Schuck

CASCAVEL

2010

MARCELO SCHUCK

SIMULAÇÃO DE MODELOS DE COERÊNCIA DE CACHE UTILIZANDO PLATAFORMAS VIRTUAIS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Dr. Marcio Seiji Oyamada

CASCAVEL

2010

MARCELO SCHUCK

SIMULAÇÃO DE MODELOS DE COERÊNCIA DE CACHE UTILIZANDO PLATAFORMAS VIRTUAIS

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Marcio Seiji Oyamada (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Aníbal Mantovani Diniz
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Jorge Bidarra
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 29 de Novembro de 2010.

DEDICATÓRIA

Este trabalho é dedicado à minha família, que sempre me apoiou e me incentivou nesta jornada, agora cumprida, e à todos os meus amigos que me deram forças para que eu não desistisse.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus e também a minha família por todo o apoio desde o início desta longa jornada, que sempre me apoiou em todos os momentos.

Agradeço a todos os amigos que estiveram ao meu lado todos esses anos. Seja em momentos de diversão ou em momentos de desespero, para a finalização de um trabalho, seja na faculdade ou em qualquer lugar, a participação de todos, sem duvida, não será esquecida. Um agradecimento em especial aos amigos Padal, Jhonata, Adriano, Mauro, Juliano Lamb. entre tantos outros.

Gostaria de deixar um agradecimento especial a todos os professores do colegiado de Ciência da Computação de forma geral, pela compreensão e dedicação, motivos que me auxiliaram a continuar na luta diária. Em especial ao professor Márcio, amigo e orientador deste trabalho, pelos anos de convívio no Laboratório de Sistemas de Computação, e por toda a confiança depositada em todos esses anos de trabalho em projetos de iniciação científica. Também gostaria de pedir desculpas caso tenho deixado algo a desejar.

A todos com quem eu estive envolvido nesses anos eu gostaria de agradecer, na esperança de que nossos caminhos continuem interligados, para manter vivo esse período muito especial da minha vida.

Marcelo Schuck

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ABS	Anti-lock Braking System
ARM	Advanced RISC Machine
CISC	Complex Instruction Set Computer
CMOS	Complementary metal-oxide-semiconductor
COMA	Cache Only Memory Access
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DVD	Digital Vídeo Disc
EDA	Electronic Design Automation
ES	Embedded Systems
E/S	Entrada e Saída
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
HDL	Hardware Description Language
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LFU	Last Frequently Used

LRU	Last Recently Used
MESI	Modified, Exclusive, Shared, Invalid
MIPS	Microprocessor without Interlocked Pipe Stages
MOESI	Modified, Owned, Exclusive, Shared, Invalid
MPSoC	Multiprocessor System-on-Chip
MS-DOS	MicroSoft Disc Operating System
MSI	Modified, Shared, Invalid
NoC	Network-on-Chip
NUMA	Non Uniform Memory Access
PC	Program Counter
PISA	Portable Instruction Set Architecture
PowerPC	Power Optimization With Enhanced RISC–Performance Computing
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SMP	Symetric MultiProcessor
SoC	System-on-Chip
SoCIN	SoC Interconnection Network
SPARC	Scalable Processor Architecture
SPIN	Scalable Programmable Interconnection Network
SRAM	Static Random Access Memory
TLM	Transaction Level Messages
UMA	Uniform Memory Access
UML	Unified Modeling Language
VIPRO-MP	Virtual Prototype for Multiprocessor Architectures

VLSI	Very Large Scale Integration
WAN	Wide Area Network

Sumário

Resumo

Visando o aumento de desempenho, o projeto de arquiteturas embarcadas tem intensificado o uso de arquiteturas multiprocessadas. A utilização desse conceito resulta em melhorias de desempenho, menor consumo de potência e do tamanho do *chip* a ser projetado e com custos reduzidos. Apesar dos benefícios, a complexidade gerada por um sistema desse tipo deve ser vista como o principal problema a ser resolvido. Plataformas virtuais são utilizadas para auxiliar o projetista na escolha dos melhores módulos que irão integrar o Sistema Embarcado. Desta forma, é necessário que uma plataforma virtual possibilite a utilização de uma ampla variedade de módulos de *hardware* na simulação. Visando prover esta versatilidade, este trabalho estendeu o VIPRO-MP (uma plataforma virtual multiprocessada), com protocolos de coerência de cache e análise do impacto no desempenho desta melhoria na arquitetura. Com base nos dados obtidos nos estudos de caso, foi observado um ganho de até 74,64% de desempenho na arquitetura que apresentava uso da memória cache, se comparado a aplicação sendo executada sem o uso desta memória. Analisou-se também, que uma variação na forma em que os dados são manipulados pode gerar impactos significativos na arquitetura, aonde nas simulações ¨pode ser obtido uma variação de até 21,07% de desempenho. Assim, com este trabalho foi possível validar, os ganhos da utilização da cache em Sistemas Embarcados, e como os protótipos virtuais contribuem para o desenvolvimento de um ES de qualidade.

Palavras-chave: Arquiteturas multiprocessadas, *chip,* coerência de cache, desempenho, sistemas embarcados.

Capítulo 1

Introdução

Sistemas Computacionais Embarcados (*Embedded Systems* - ES) são caracterizados por serem dispositivos de aplicação específica, que podem utilizar múltiplos processadores juntamente com os componentes necessários ao seu funcionamento, como memória, barramento e periféricos [Garcia, 2008]. De modo geral, ES são tidos como dispositivos que devem possuir funcionamento contínuo e prolongado por anos, com ocorrência mínima de erros e/ou falhas. Sistemas de controle de vôo, telefones celulares, freios ABS (*Anti-lock Braking System)*, eletrodomésticos, equipamentos médicos e vídeo games são exemplos de ES.

A necessidade das empresas projetarem Sistemas Embarcados, ou circuitos integrados, dentro de janelas de tempo cada vez mais reduzidas, satisfazendo às pressões mercadológicas, e a constante evolução tecnológica, obrigam o desenvolvimento de projetos otimizados [Wagner e Carro, 2003]. Dentre os avanços tecnológicos de fabricação desses sistemas, destacam-se as soluções MPSoC (*Multiprocessor System-on-Chip*) [Jerraya, 2004]. Estas provêem em um único *chip*, múltiplos processadores que podem ser incluídos juntamente com interfaces digitais, componentes de aplicação específica, memória, entre outros dispositivos. Como problema do uso desse tipo de arquitetura, tem-se uma maior complexidade agregada ao projeto. Para identificar potenciais problemas em um estágio inicial do projeto, ou seja, antes da finalização do *hardware*, a comunidade de EDA (*Electronic Design Automation*) propõe o uso de protótipos virtuais. Protótipos virtuais são modelos de simulação de sistemas completos (processadores, interfaces de E/S (Entrada e Saída), hierarquia de memória, etc). Com a simulação é possível analisar o desempenho obtido variando componentes, como

memória, número e frequência dos processadores a serem utilizados em um MPSoC, eliminando a necessidade de sintetização do *hardware* para se obter esses comparativos.

Em uma arquitetura multiprocessada que apresenta memória global, uma forma de elevar o desempenho da arquitetura é a possibilidade de armazenar na memória cache dos processadores, dados e instruções. Este fato se deve ao acesso a esta memória ocorrer de forma mais rápida que na memória global. Desta forma, reduzindo o tempo de acesso aos dados a energia gasta também sofre redução. Portanto, o uso da cache promove um acesso mais rápido e consequentemente aumenta o desempenho dos processadores. Porém, como cada processador tem uma cache, são necessários mecanismos para garantir a coerência, ou seja, garantir que os dados apresentem valores válidos em todos os processadores. Isso é necessário, uma vez que, um mesmo dado por ser acessado e manipulado por mais de um processador. Métodos que garantem a consistência dos dados manipulados são denominados protocolos de coerência de cache [Patterson e Hennessy, 2002], sendo classificados em:

a) Rastreamento (*Snooping*): neste tipo de protocolo, a coerência é garantida através do rastreamento do canal onde é realizada a movimentação dos dados, sendo este canal denominado barramento. Assim, cada processador fica responsável por verificar a atividade do barramento. Quando algum dado armazenado na cache local for alterado em outro processador, este dado é invalidado localmente;
b) Diretório: neste tipo de protocolo, um módulo de *hardware* ligado ao barramento, chamado de diretório, armazena os endereços da cache que estão mapeados nos diversos processadores do sistema. Em uma atualização, o diretório deve avisar os processadores que os valores referentes ao endereço atualizado são inválidos.

Em ES, geralmente existe uma memória global no sistema, onde são armazenados tanto dados como instruções. Com o desenvolvimento de soluções MPSoC, ampliou-se a necessidade de garantir uma forma eficiente de acesso à memória. Por ser uma solução multiprocessada, é desejável que um acesso simultâneo de vários processadores à memória não se torne um gargalo causando perda de desempenho na execução de uma aplicação.

Além disso, soluções MPSoC passam a disponibilizar arquiteturas com um número elevado de processadores em um único *chip*. Este fato torna o barramento, gerenciador de

acessos serializados à memória, uma solução não escalável em longo prazo. A Figura 1.1 [Garcia, 2008] mostra um exemplo de uma arquitetura com dois processadores, onde é necessário um acesso serializado ao barramento, desta forma, o barramento torna-se o gargalo do sistema para a movimentação de uma grande quantidade de dados [Garcia et al., 2010].

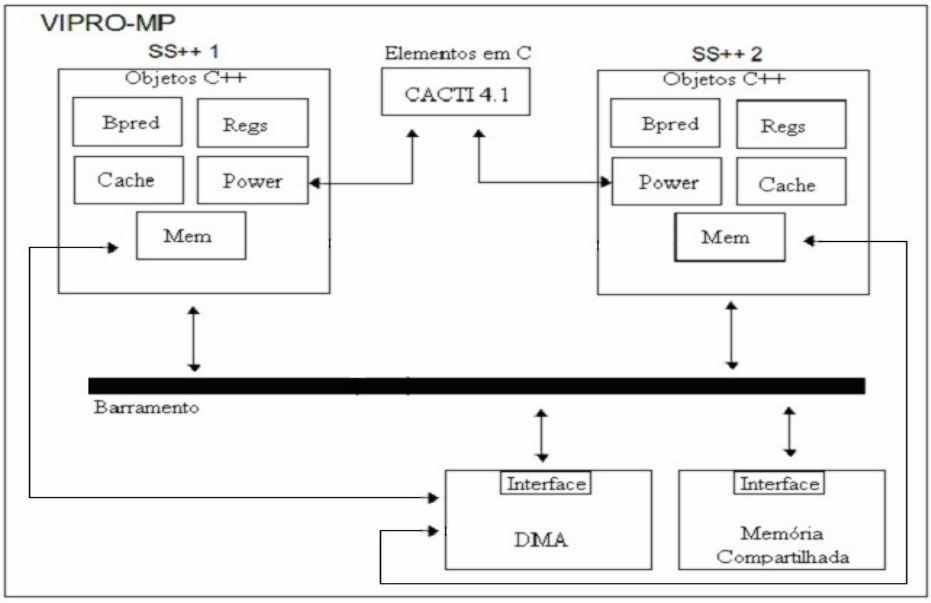

Figura 1.1: Arquitetura com dois processadores e barramento seqüencial [Garcia, 2008].

No intuito de avaliar o ganho que a manipulação de dados na memória cache pode apresentar em uma arquitetura multiprocessada, neste trabalho são propostas extensões, com a utilização da memória cache, no ambiente VIPRO-MP [Garcia, 2008]. Além disso, um comparativo entre formas diferentes de acesso aos dados presentes nesta memória cache será realizando, enfatizando que este processo também deve ser considerado no projeto de um ES.

1.1 Metodologia

A metodologia consistirá no estudo dos diversos protocolos de coerência de cache. Um protocolo do tipo rastreamento e outro do tipo diretório será escolhido e implementado no VIPRO-MP [Garcia e Oyamada, 2008], um ambiente livre para construção de ambientes multiprocessados implementado em SystemC [SystemC, 2010] e tendo o SimpleScalar

[Simplescalar, 2010] como simulador do processador. Para avaliação do protocolo de coerência de cache em sistemas multiprocessados uma aplicação JPEG (*Joint Photographic Experts Group*) paralela [Garcia, 2008] desenvolvida em trabalhos anteriores será utilizada. A avaliação do protótipo virtual será realizada em termos de desempenho, consumo de potência, além da complexidade adicional para implementação de cada protocolo de coerência de cache mencionado.

1.2 Justificativa(s)

O uso de memórias cache possibilitam que dados possam ser acessados de forma rápida. Este fato é de fundamental importância para ganho de desempenho, pois acelera o tempo de acesso e manipulação de dados. A cache provendo suporte ao armazenamento de dados da memória compartilhada, faz com que a memória compartilhada possa armazenar inclusive instruções, possibilitando que arquiteturas SMP (*Symetric MultiProcessor*) sejam simuladas. No entanto, em arquiteturas que apresentam múltiplos processadores, o uso de cache pode acarretar em inconsistências nos dados. Desta forma, em uma plataforma multiprocessada é fundamental que protocolos de coerência de cache sejam implementados, para que as aplicações possam obter melhor desempenho através da utilização desta memória.

Além disso, é necessário que haja uma forma eficiente de acesso aos dados. Tanto em sistemas embarcados, bem como em sistemas multiprocessados, geralmente os dados estão presentes numa memória compartilhada, de acesso serializado. Assim, o tempo em que um processador fica ocioso esperando o atendimento de uma solicitação de leitura ou escrita de um determinado dado é elevado. Variando a forma que ocorre um acesso aos dados, evitando o acesso serializado, buscando um comparativo entre tipos diferentes de acesso a dados que garantam coerência é outra forma de buscar elevação de desempenho.

1.3 Objetivos

O objetivo deste trabalho é gerar extensões ao ambiente VIPRO-MP, uma plataforma virtual com suporte a simulação de múltiplos processadores, sendo baseada na arquitetura

Simplescalar [Burger e Austin, 1997]. O VIPRO-MP foi desenvolvido no ano de 2008, originado do trabalho de conclusão de curso de Maxiwell Salvador Garcia [Garcia, 2008]. A arquitetura Simplescalar é um conjunto de ferramentas para análise de desempenho de processadores, desenvolvido e disponibilizado gratuitamente pelo Departamento de Computação da Universidade de Wisconsin-Madison.

A primeira extensão consiste em prover suporte para o armazenamento, na cache, de dados provenientes da memória compartilhada. Em seguida será desenvolvido dois métodos diferentes que garantam coerência de cache. A diferença entre esses modelos consiste na forma que ocorre o acesso aos dados presentes na memória cache e sua posterior movimentação entre os processadores. Tendo esses modelos implementados, é possível gerar um comparativo que auxilie um projetista no momento de escolha de qual modelo de coerência de cache adotar.

1.4 Estruturação do Texto

Para que os conceitos discutidos possam ser implementados e utilizados da melhor forma, no Capítulo 2 serão apresentados conceitos envolvendo a memória cache, bem como os principais algoritmos utilizados para garantir uma coerência dos dados manipulados. No Capítulo 3 será realizado um estudo inicial sobre Sistemas Embarcados. O VIPRO-MP será mostrados no Capítulo 4, sendo no Capítulo 5 demonstrado os métodos de coerência de cache implementados e os dados obtidos nas simulações. As conclusões obtidas com este trabalho serão expostas no Capítulo 6.

Capítulo 2

Memória Cache

A elevação da frequência do *clock,* apresentada pela evolução dos processadores, proporcionou uma elevação no número de transações possíveis de ocorrência em um determinado intervalo de tempo [Stacpoole e Jamil, 2000]. Desta forma, o processador necessita mais dados, que devem ser fornecidos de forma eficiente [Stacpoole e Jamil, 2000]. A memória principal apresenta um tamanho elevado e custo reduzido para sua utilização, fatos que poderiam satisfazer estas necessidades. Porém, a velocidade de acesso aos dados é reduzida. Assim, a utilização de uma memória com maior rapidez de acesso aos dados tornou-se necessária.

Surgida para preencher a lacuna formada entre a CPU (*Central Processing Unit*) e a memória principal, a memória cache é caracterizada por apresentar elevada velocidade no tempo de acesso a dados [Stacpoole e Jamil, 2000]. A cache, geralmente, é utilizada para armazenar as instruções e dados utilizados recentemente pela CPU [Stacpoole e Jamil, 2000]. Desta forma, a cache é utilizada pela CPU para encontrar um endereço de que necessita durante sua execução. Apesar de ser um modelo de acesso rápido, a utilização da memória cache em dispositivos é limitada devido ao elevado custo envolvido na utilização da mesma [Stacpoole e Jamil, 2000]. Com isso, as memórias cache passam a serem consideradas itens especiais de memória utilizados em um sistema computacional [Stacpoole e Jamil, 2000].

A memória cache apresenta vantagem em sua utilização, pois esta utiliza os princípios de localidade [Junior, 2007] [Stacpoole e Jamil, 2000] para encontrar os endereços de que a CPU necessita. A aplicação de um conceito de localidade aumenta a eficiência em tempo de execução de uma aplicação. Este fato ocorre, pois, as referências à memória, em um intervalo

de tempo da execução, não ocorrem de forma aleatória [Junior, 2007], mas sim sequencial. As aplicações do princípio de localidade se baseiam em dois componentes [Stacpoole e Jamil, 2000] que são:

a) espacial, ou seja, um item acessado tende a possuir um endereço sequencial, ou vizinho, de um endereço de item acessado no instante anterior [Junior, 2007] [Stacpoole e Jamil, 2000] ou;
b) temporal, ou seja, o acesso remete a um endereço recentemente acessado [Junior, 2007], como por exemplo, em uma instrução de laço de repetição [Stacpoole e Jamil, 2000].

A utilização de princípios de localidade possibilita que o desempenho da execução, relacionado à manipulação da memória, apresente uma elevação [Stacpoole e Jamil, 2000]. Esta elevação ocorre devido à possibilidade de manter, em uma memória mais rápida, um conjunto de endereços de uma memória mais lenta por um intervalo de tempo [Junior, 2007]. Caso o endereço já esteja na cache, a CPU pode utilizá-lo de imediato. Caso contrário, uma busca a endereços vizinhos, aos que estão na cache, é realizado evitando assim uma busca sequencial na memória principal. Assim, uma busca por endereços de uma memória lenta, ocorre com um conjunto de endereços inseridos em uma memória de velocidade elevada [Junior, 2007].

A memória cache, assim como os registradores, é interna ao processador [Stacpoole e Jamil, 2000], o que confere altas velocidades de uso e acesso à memória. Cache é um nome genérico dado ao nível da hierarquia de memória situado após os registradores [Patterson e Hennessy, 1996]. Uma hierarquia de memória divide os tipos de memórias em níveis e é gerada através de uma comparação entre as seguintes características [Stacpoole e Jamil, 2000]:

a) tempo de acesso, ou seja, tempo gasto pela CPU para acessar os dados de um dado nível;
b) tamanho da memória, que se refere ao tamanho que uma memória pode apresentar em um determinado nível ;
c) custo por *byte,* sendo a relação entre tamanho da memória e custo para sua utilização;

d) largura da banda de transferências, ou seja, o tempo gasto para movimentar dados entre os níveis da hierarquia;
e) unidade de transferência, é o tamanho em que um dado, ou um bloco de dados, podem ser transferidos simultaneamente.

A Figura 2.1 [Stacpoole e Jamil, 2000] apresenta a hierarquia em que as memórias podem ser divididas. Neste modelo, os tipos de memória que estão no topo apresentam elevada velocidade no acesso a dados, tamanho reduzido, custo elevado por *byte*, uma elevada largura de banda e possuem unidades de transferências reduzidas, se comparadas às memórias que estão mais na base da hierarquia [Stacpoole e Jamil, 2000]. Além de analisar estes dados, é importante mencionar que de 20% a 30% do tempo normal de execução de um programa é relacionado a acessos a memória [Stacpoole e Jamil, 2000].

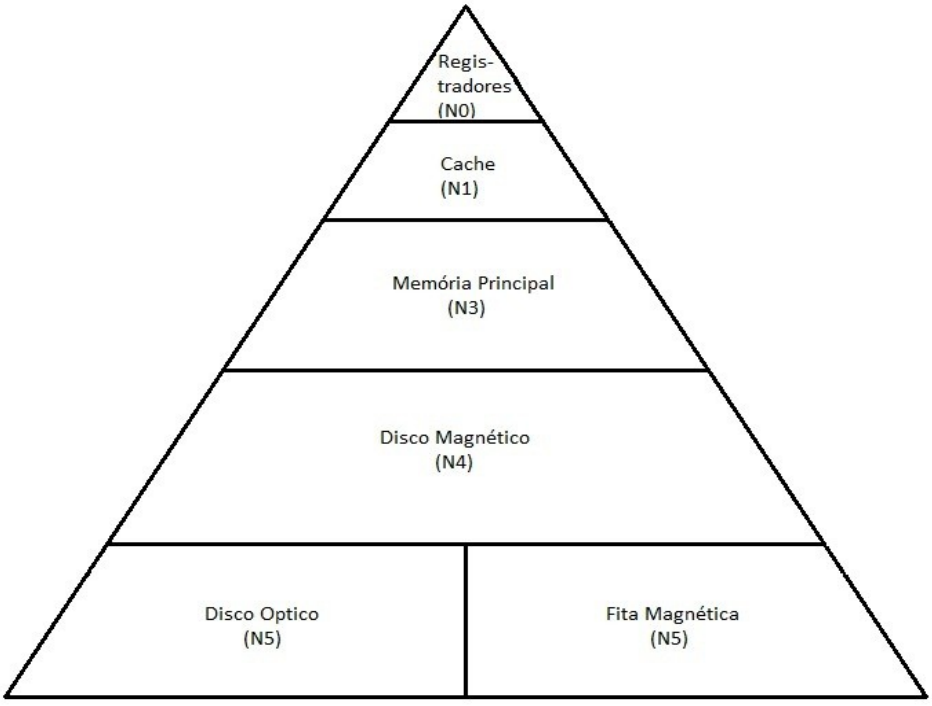

Figura 2.1: Hierarquia típica das memórias [Stacpoole e Jamil, 2000].

Na definição de uma cache, além de definir informações como tamanho e níveis utilizados, é necessário uma definição de como será realizado o mapeamento de blocos da memória

principal para a cache. Além disso, quando a memória cache é utilizada, é necessário que a coerência nos dados seja garantida. Nas sessões seguintes serão discutidos esses assuntos, além de outros fatores que são utilizados para gerar uma diferenciação entre as memórias cache.

2.1 Mapeamentos na Memória

Devido ao seu tamanho e a tecnologia, a memória cache possibilita que o processador encontre geralmente em um único ciclo de execução um dado de que necessita. Nessa situação ocorre um cache *hit*, ou seja, o dado presente em um endereço solicitado pode ser encontrado com um dos blocos mapeados na cache [Junior, 2007]. Quando este endereço solicitado não pode ser obtido de forma direta, através dos blocos mapeados na cache, ocorre uma situação de cache *miss* [Junior, 2007].

Em uma situação de cache *miss*, um novo bloco deverá ser mapeado da memória principal para a cache, sendo em seguida refeita a busca, na cache, pelo processador, para que ocorra um cache *hit* [Junior, 2007]. Esta situação demanda o gasto de muitos ciclos de execução para ser finalizada, uma vez que ocorre uma cesso a uma memória mais lenta. Caso houver espaço físico livre na cache, este bloco é apenas inserido, no final de pilha de blocos. Caso contrário, um bloco deverá ser escolhido, segundo algum critério, e ser eliminado.

A memória cache geralmente é dividida em blocos, ou linhas, dependendo da nomenclatura utilizada pelo autor, sendo associado a cada bloco um conjunto de informações, que formam uma estrutura denominada diretório [Junior, 2007]. A forma que é utilizada para realizar um mapeamento dos blocos da memória principal para a memória cache é denominada mapeamento. Os principais tipos de mapeamento são: mapeamento direto, mapeamento totalmente associativo e mapeamento associativo por conjunto.

No mapeamento direto, cada bloco da memória principal é mapeado em uma linha da memória cache. Associado a este bloco, na cache, existe uma *tag*, que contém informações sobre um endereço da memória principal, este conjunto de informações permite identificar se a informação desejada esta ou não na cache [Junior, 2007]. Este modelo de mapeamento é o mais simples de ser implementado, com custos reduzidos, porém apresenta a limitação de um

mapeamento 1x1. Isto que dizer que cada bloco da memória principal carregado na cache ocupa uma linha da cache, assim se um dado da memória principal ocupar mais de um bloco, a troca de linhas na memória cache será constante, reduzindo o desempenho da utilização da mesma [Junior, 2007].

No mapeamento totalmente associativo, cada bloco da memória principal pode ser mapeado para qualquer linha da memória cache. Neste modelo, o tamanho da *tag* destinada a armazenar dados para verificação da ocorrência de cache *hit* ou *miss* é o mesmo que o do mapeamento direto. Essa característica aumenta a flexibilidade da cache e aumenta seu desempenho [Junior, 2007]. Assim, uma mesma linha da cache pode apresentar um maior número de associações com blocos da memória principal. Para ser encontrado um bloco, uma busca sequencial deve ser realizada em todas as linhas da cache [Junior, 2007]. Sendo esta a principal desvantagem do mapeamento totalmente associativo, devido à complexidade de se desenvolver uma cache que suporte comparação sobre todas as suas linhas de forma eficiente [Junior, 2007].

O mapeamento associativo por conjunto reúne as principais vantagens do mapeamento direto e totalmente associativo em um único modelo [Junior, 2007]. Neste mapeamento a cache é reorganizada, criando um conjunto de linhas, sendo cada uma identificada por uma *tag* diferente. Assim, um conjunto apresenta um número variado de associações a blocos da memória principal, porém, internamente a este conjunto, cada associação é representada por uma *tag*, que permite identificar se a informação desejada está ou não na cache. Assim, em uma busca, a cache seleciona um conjunto e, verificando a *tag,* consegue identificar se a informação desejada está ou não naquele conjunto mapeado. As vantagens desse modelo é a eliminação de uma busca sequencial por toda a cache, como no mapeamento totalmente associativo, e a eliminação da constante troca de valores das linhas da cache presente em um mapeamento direto.

Desta forma, dependendo do modelo de mapeamento utilizado, podem ocorrer acréscimos de custos, complexidade e redução ou ganho de desempenho. A adoção de um modelo de cache eficiente também é necessário, para possibilitar que o ES desenvolvido apresente o melhor desempenho possível.

2.2 Políticas de Substituição de Blocos na Cache

Na ocorrência de um cache *miss* é necessário inserir um novo bloco de informações da memória principal na memória cache. Esse bloco possui a informação necessária à continuação da execução do processador. Caso não existam mais linhas livres na cache, alguma das linhas deverá ter seu valor substituído, pelos dados do novo bloco a ser carregado.

Algoritmos que realizam essa troca de blocos na cache são denominados, algoritmos de substituição, e estes são baseados em alguma metodologia, ou seja, política, para escolher qual o melhor bloco a ser substituído. As políticas utilizadas são: FIFO, LRU, LFU e aleatória. As duas políticas mais utilizadas para a substituição de blocos na memória cache são a escolha aleatória e a política LRU (*Last Recently Used*) [Patterson e Hennessy, 1996]. A escolha aleatória é simples de ser implementada e entendida. Quando um bloco precisa ser substituído na cache, é escolhido um bloco de forma aleatória e este é eliminado. O problema dessa política é a possibilidade de eliminar um bloco com constante utilização, causando uma elevação em número de cache *miss* [Junior, 2007].

Este problema é, em parte, solucionado pela adoção da política LRU. Como o nome indica, o bloco escolhido para ser eliminado é aquele que a mais tempo não é acessado. Esta política apresenta maior complexidade para ser desenvolvida, acrescenta a necessidade de realizar registro do tempo de acessos a um bloco, e demanda mais tempo para sua realização, devido ao processo de busca do bloco. Apesar disso, este modelo de substituição de blocos evita a ocorrência de um número elevado de caches *miss*. O método LFU (*Last Frequently Used*) é semelhante à política LRU, porém, este método analisa dados relacionados a frequência de atualização do blocos, em vez do tempo de acesso, como no LRU.

O método FIFO (*First-In First-Out*) é representado por eliminar o bloco que foi primeiramente inserido na cache. É uma política simples de ser implementada e apresenta resultados geralmente satisfatórios.

2.3 Políticas de Atualização da Memória Cache

Política de atualização é a forma como ocorre a atualização de informações entre a memória cache e a memória principal, garantindo uma coerência entre os dados dessas memórias. As políticas de atualização irão indicar em qual instante será realizada a atualização da memória principal. As políticas de atualização são:

a) *write through*: ocorre simultaneamente a escrita na cache e na memória principal. Sua principal desvantagem é o tempo de escrita mais lento, uma vez que, deve ocorrer em ambas as memórias [Junior, 2007];
b) *write back*: a escrita ocorre apenas na cache, sendo atualizado na memória principal apenas quando o bloco for removido da cache [Junior, 2007]. Como desvantagem está a elevada movimentação de dados entre memória principal e cache, quando este bloco for escrito na memória principal [Junior, 2007].

Do mesmo modo que as políticas de substituição de blocos na cache e o modelo de mapeamento de blocos, as políticas de atualização também interferem no desempenho final de um dispositivo. Assim, é necessário um entendimento das características referentes à cache, antes de sua utilização, para garantir que as melhores opções sejam utilizadas obtendo o maior desempenho possível ES pretendido.

2.4 Protocolos de Coerência de Cache – Modelo Rastreamento

Para garantir que seja mantida uma coerência nos dados das caches, um protocolo de coerência de cache deve ser utilizado. Para protocolos do tipo rastreamento, são utilizados os modelos de protocolos MSI [Baskett et al., 1988], MESI [Papamarcos e Patel, 1984], MOESI [Sweazey e Smith, 1986] e DRAGON [Thacker et al., 1988].

No protocolo MSI (*Modified, Shared, Invalid*), os dados da cache podem apresentar três estados de análise. Estes estados são:

a) modificado (Modified - M):

b) compartilhado (Shared – S) e;

c) inválido (Invalid – I).

O estado inválido indica que o bloco não deve ser utilizado por apresentar dados não coerentes. O estado compartilhado indica que mais de uma cache contém uma cópia válida do mesmo bloco. E o estado modificado é utilizado para indicar que um bloco sofreu alteração pelo processador e a cópia na memória principal está desatualizada [Junior, 2009].

Um exemplo do diagrama de estados deste protocolo pode ser observado na Figura 2.2 [Junior, 2009]. Neste diagrama está associada, a cada transição, uma condição para que ela ocorra e uma consequência, obtendo o formato causa/consequência [Junior, 2009]. As causas e consequências estão relacionadas a leituras e escritas do processador (PrRd, PrWr), escritas, leituras e leituras exclusivas do barramento (BusRd, BusWr, BusRdx) e a operações da própria cache (Flush) [Junior, 2009]. O método *Flush* consiste em utilizar alguma política de atualização da memória cache.

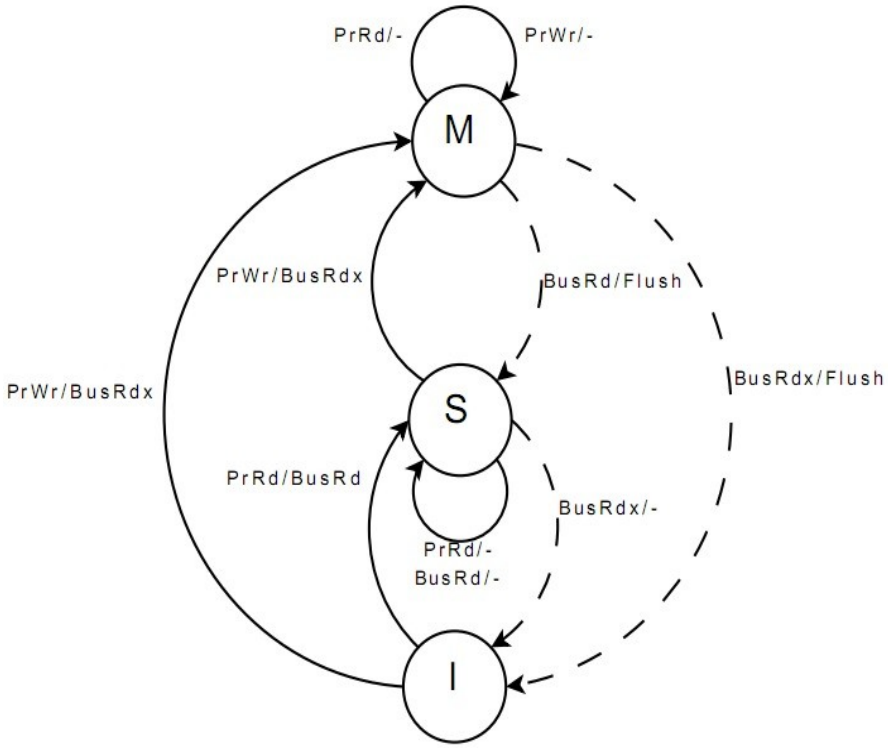

Figura 2.2: Protocolo de coerência de cache MSI [Junior, 2009].

As leituras exclusivas do barramento sempre obrigam a cache a realizar a atualização, na memória principal, dos dados que ela possui (Flush), invalidando as demais cópias. Se uma posição está em estado inválido e o processador realiza uma leitura desta posição, devido a uma leitura que o barramento irá realizar, para obter o dado em outra memória, este endereço fica em estado compartilhado. Caso o processador realize uma escrita sobre um dado inválido, uma leitura especial será feita no barramento, para invalidar as demais cópias, e o endereço local fica em estado modificado.

Caso um endereço esteja em estado compartilhado, leituras de processadores ou do barramento não causarão uma alteração no seu estado. Uma alteração só ocorre se houver uma escrita por um processador, obrigando a invalidação das demais cópias, e este endereço alterará seu estado para modificado. Quando um endereço está com estado modificado, uma alteração de estado só vai ocorrer se uma leitura especial do barramento ocorrer.

Um problema deste modelo é quando ocorrem leituras e escritas consecutivas, como numa operação de incremento [Junior, 2009]. Por não saber se existem cópias do endereço, após

realizar a leitura que altera o estado para compartilhado, a escrita deve gerar uma leitura exclusiva, mudando o estado de compartilhado para inválido [Junior, 2009]. O protocolo MESI (*Modified, Exclusive, Shared, Invalid*) elimina este problema, inserindo sinais que indicam a existência de cópias do bloco em outras caches (BusRd(S)), ou que indicam que não existem outras cópias (BusRd((S\)) [Junior, 2009].

Além desses sinais, um novo estado foi criado e inserido nas possibilidades de definição de um bloco. Este estado é denominado exclusivo (*Exclusive* – E), que representa a não existência de outras cópias do bloco analisado. Assim, um estado pode passar de inválido para exclusivo com uma leitura feita pelo processador caso não existir outras cópias destes blocos nas demais caches [Junior, 2009]. Um exemplo do diagrama do protocolo MESI é demonstrado na Figura 2.3 [Junior, 2009]. As transições entre os estados são semelhantes às mencionadas no protocolo MSI.

Uma deficiência no protocolo MESI ocorre quando um processador esta constantemente alterando uma posição de memória e um segundo processador está lendo esta mesma posição [Junior, 2009]. Nesta situação, constantemente ocorre uma escrita das alterações feitas na cache do processador na memória principal, para que o segundo processador possa obter o dado atualizado. Assim, ocorre uma constante manipulação de uma memória mais lenta, aderindo uma perda de desempenho na execução do sistema [Junior, 2009].

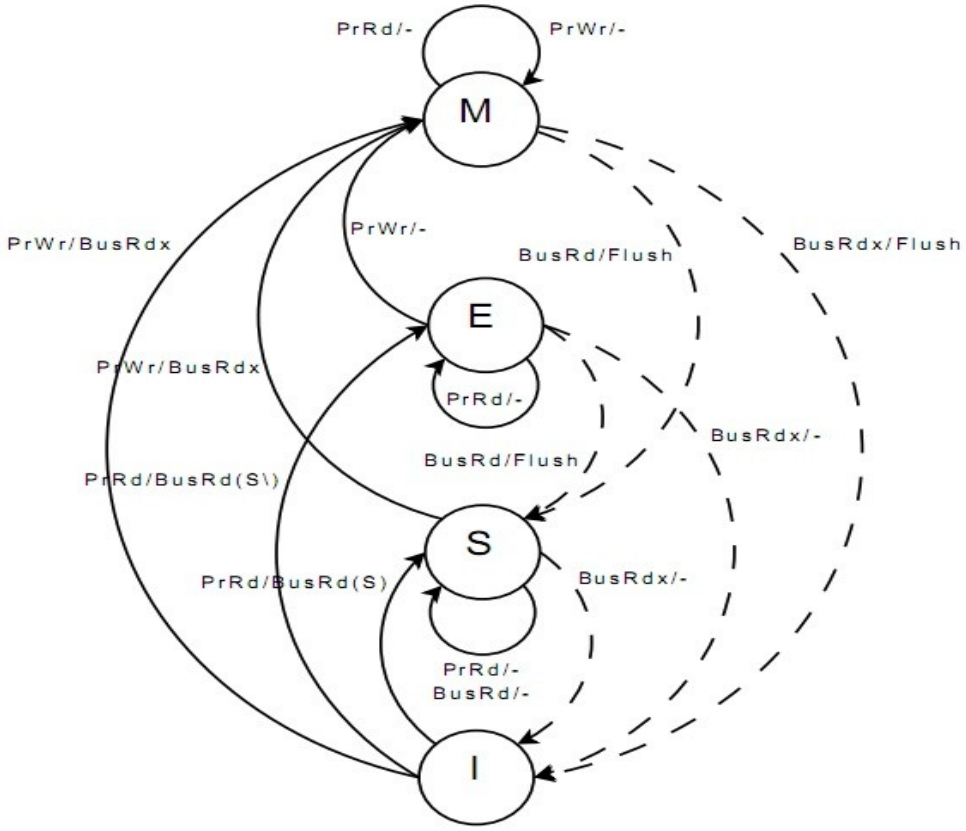

Figura 2.3: Protocolo de coerência de cache MESI [Junior, 2009].

Para corrigir a deficiência do protocolo MESI, um novo estado denominado *owned*, foi anexado ao protocolo MESI, obtendo o protocolo MOESI (*Modified, Owned, Exclusive, Shared, Invalid*). Um diagrama de estados do protocolo MOESI está representado na Figura 2.4 [Junior, 2009].

O estado o*wned* é utilizado para indicar que o bloco está modificado em relação à memória principal e não a outras caches [Junior, 2009]. Desta forma quando um bloco é solicitado via barramento e este se apresente no estado modificado, o estado deste bloco passa a apresentar o estado *owned* e toda vez que este bloco foi solicitado, este deve ser obtido de uma cache que o apresente neste estado. Apenas uma leitura exclusiva irá fazer com que este bloco seja escrito na memória principal, reduzindo a movimentação de dados no sistema.

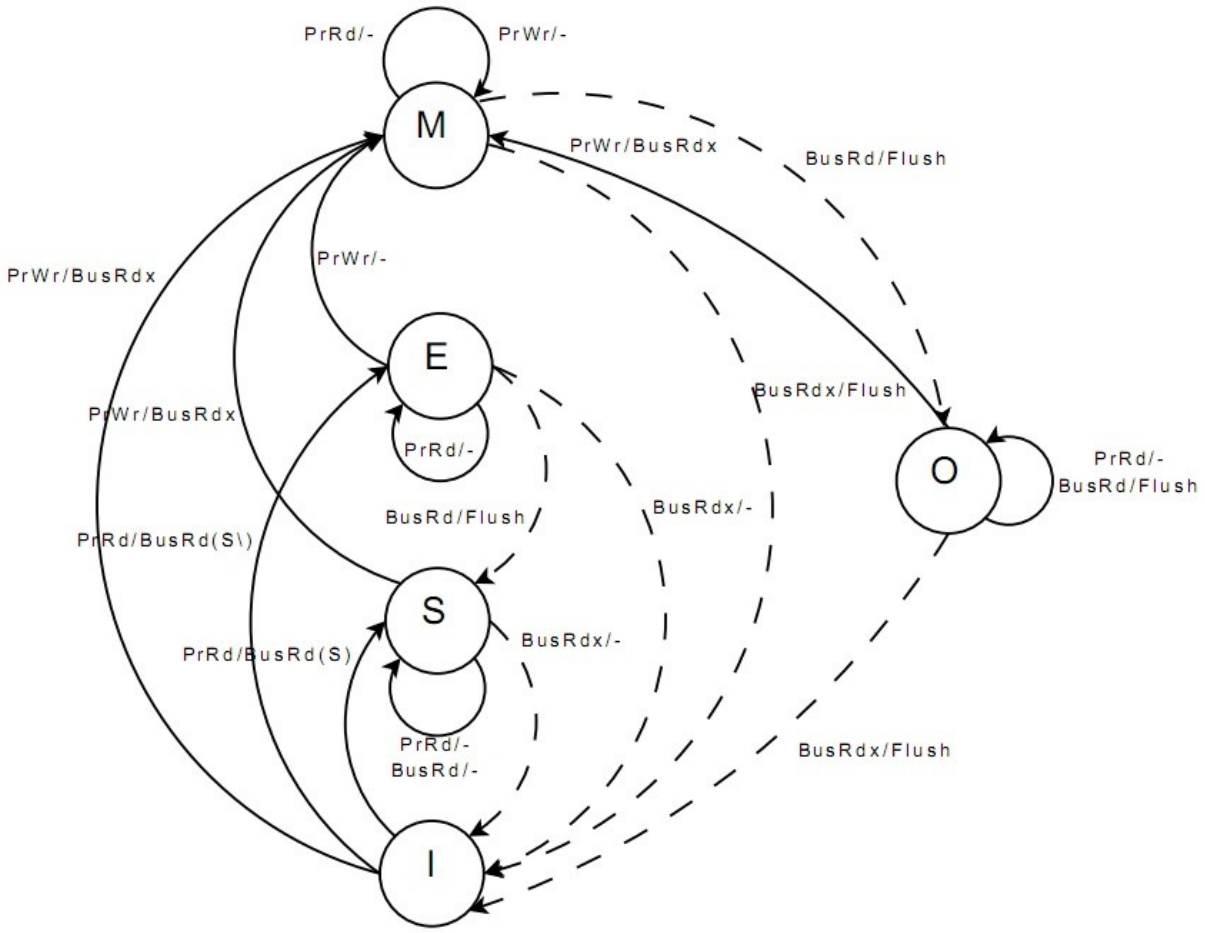

Figura 2.4: Protocolo de coerência de cache MOESI [Junior, 2009].

O protocolo DRAGON, cujo diagrama de estados é apresentado na Figura 2.5 [Junior, 2009], apresenta quatro estados (*Exclusive, Shared Clear, Shared Modified, Modified*). Os estados *Exclusive* e *Modified* são semelhantes aos dos protocolos anteriores [Junior, 2009]. O estado *Shared Clear* indica que o bloco está em duas ou mais caches, podendo a memória principal estar, ou não, atualizada [Junior, 2009]. E o estado *Shared Modified* indica que o bloco está presente em mais de uma cache, que a memória principal não está atualizada e que a cache nesse estado deve fornecer este bloco [Junior, 2009]. O estado inválido não é utilizado, pois se considera que os blocos da cache sempre estão preenchidos após a inicialização [Junior, 2009].

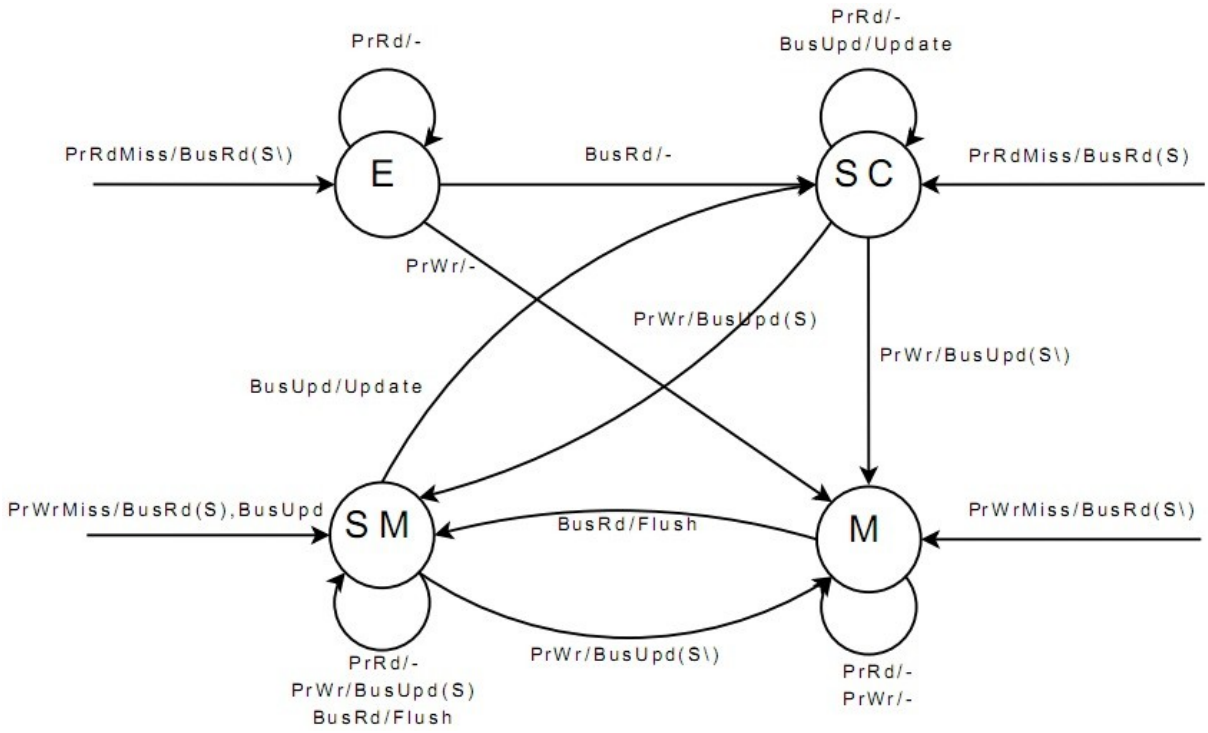

Figura 2.5: Protocolo de coerência de cache DRAGON [Junior, 2009].

O protocolo DRAGON utiliza outra operação de barramento, denominada *update*, responsável pelo ganho de desempenho deste protocolo [Junior, 2009]. Esta operação permite que apenas palavras sejam manipuladas pelo barramento e não blocos, diminuindo, desta forma, o tráfego de dados em determinadas situações [Junior, 2009]. Essa redução é possível uma vez que, caches no estado *shared clear*, mantém os dados atualizados sem a necessidade de transferir toda a linha entre as caches e/ou memória principal [Junior, 2009].

Dentre esses protocolos os mais utilizados são o MESI e o MOESI [Junior, 2009]. Ambos os protocolos apresentam desempenho elevados, não variando muito entre eles. Para uma implementação, o protocolo MOESI exige uma maior complexidade no desenvolvimento, devido ao fato da existência do estado *owned*. Desta forma, optou-se por utilizar e implementar o protocolo MESI, para prover uma coerência na utilização das caches no VIPRO-MP, devido ao alto desempenho e a menor complexidade de implementação.

2.5 Protocolos de Coerência de Cache – Modelo Diretório

O modelo de coerência de cache diretório, pode utilizar os mesmos modelos de coerência de cache citados acima. No modelo diretório existe um módulo responsável, pela gerencia de todas as informações que existe na cache. No modelo rastreamento, porém, o controle sobre a manipulação de caches é realizado pelo monitoramento das movimentações de dados do barramento.

No diretório estão presentes informações sobre quais dados da memória compartilhada estão sendo manipulados e em qual(is) cache(s) esses dados se encontram [Patterson e Hennessy, 1996]. Um modelo simples de diretório, contém pelo menos a informação se o processador apresenta uma cópia do dado, e a informação de estado desse dados, ou seja, se existe uma cópia desse dado em outras caches [Covacevice et al., 2007]. Esse estado é referente de acordo com o modelo de coerência adotado. Um exemplo de diretório pode ser visto na Figura 2.6 [Covacevice et al., 2007].

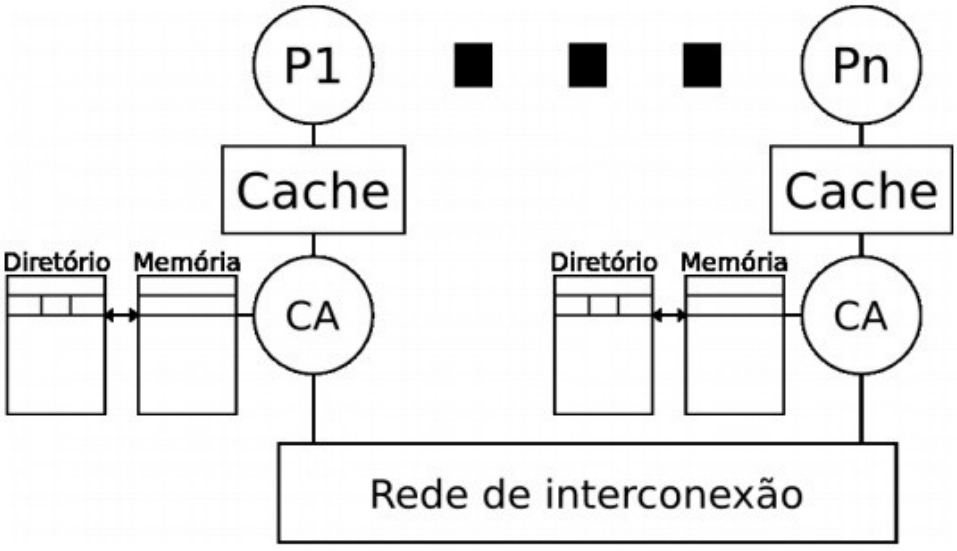

Figura: 2.6 Coerência de cache com modelo Diretório [Covacevice et al., 2007].

No modelo diretório, existe uma memória privada, que é mapeada no diretório. Essa memória sofre acessos tanto do processador local, como de outros processadores, via barramento [Patterson e Hennessy, 1996]. Desta forma, um nodo é formado por um ou mais

processadores, um diretório e uma memória privada. Assim é necessário um meio de comunicação eficiente entre os diversos nodos, como exemplo uma rede *intra-chip*.

Todas as operações sobre dados da memória cache passam pelo diretório. Como o diretório possui informações sobre os dados que existem na memória cache e o estado que esse dado possui, é facilitada uma operação de busca, em outras cache, quando ocorre um *cache miss*. Desta forma, o diretório é quem realizada o controle da coerência dos dados manipulado nas memórias cache.

A principal vantagem deste modelo é a escalabilidade [Patterson e Hennessy, 1996]. Para a inclusão de novos processadores/nodos no sistema, apenas ira ser alterado as informações do número de processadores presentes na arquitetura, no momento que é feita uma busca por informações no diretório. E como principal desvantagem desse modelo tem-se o aumento do tempo de resposta a uma solicitação do processador para manipular determinado dado, uma vez que todas as operações que ocorrem na cache devem ser centralizadas no diretório [Covacevice et al., 2007].

O modelo diretório é utilizado, geralmente, para arquiteturas com grande quantidade de processadores. Desta forma é necessário um meio e comunicação eficiente entre os processadores, para elevar o desempenho [Patterson e Hennessy, 1996]. Da mesma forma, pode ser utilizado um diretório para conter informações de mais de um processador. Com isso, conforme o número de núcleos conectados num mesmo nodo, o diretório tem seu desempenho prejudicado.

Para a realização deste trabalho, o modelo diretório foi incluído em um ambiente com barramento centralizador. Desta forma, para testar o seu desempenho, elevou-se o número de processadores presentes no nodo, uma vez que a estrutura utilizada não permite que mais de um nodo seja criado. Assim, existe apenas um diretório que centraliza todas as operações sobre as memórias caches, mesmo variando o número de processadores. Isso evidencia uma perda de desempenho com a elevação do número de processadores presentes na arquitetura.

Capítulo 3

Sistemas Embarcados

A necessidade da utilização de sistemas computacionais de pequeno porte para atividades específicas, impulsionou o desenvolvimento de Sistema Embarcado. ES passam a integrar em um único sistema, componentes como memória, barramento, dentre outros componentes de *hardware*, além disso, surge a necessidade de desenvolver *software* que facilitam a utilização destes dispositivos pelos usuários. Assim, estes sistemas passaram a ter grande utilização e importância, tanto no ramo de pesquisa como para consumidores, passando a ser denominados SoC (*System-on-Chip*) [Fisher et al., 2005].

Considerada como a área de maior importância na tecnologia de informação moderna [Marwedel, 2006], Sistemas Embarcados passam a ser utilizados para as mais variadas funcionalidades. Utilizados de eletrodomésticos, consoles de vídeo-game, a sistemas de controle de vôo e tráfego aéreo, os ES passam a ser geridos por avanços tecnológicos e necessidades mercadológicas. Estimado em torno de 16 bilhões de dispositivos ao redor do mundo [Fisher et al., 2005], o número de ES aumenta constantemente e faz com que 95% dos processadores produzidos atualmente [Fisher et al., 2005] sejam para Sistemas Embarcados. Somado a isso, o *time-to-market*, desenvolvimento de produtos para períodos específicos de comercialização, como Natal, obriga que produtos com qualidade sejam desenvolvidos de forma rápida e com custos reduzidos.

Com a evolução da tecnologia para criação de circuitos integrados, surgida na década de 60 [Junior, 2009], o desenvolvimento de sistemas computacionais de tamanho reduzido passou a envolver um grande número de pesquisadores [Wolf, 2001] pelo mundo. Este fato se deve, entre outras características, ao fato de uma tecnologia se tornar obsoleta de forma rápida, como é apresentado na Figura 3.1 [Wolf, 2001] e a necessidade dos ES apresentassem

mais funcionalidades [Garcia, 2008]. Como exemplo, os telefones celulares, que no início eram destinados apenas a ligações telefônicas, e atualmente apresentam diversas funcionalidades como acesso a Internet e reprodução de áudio e vídeo.

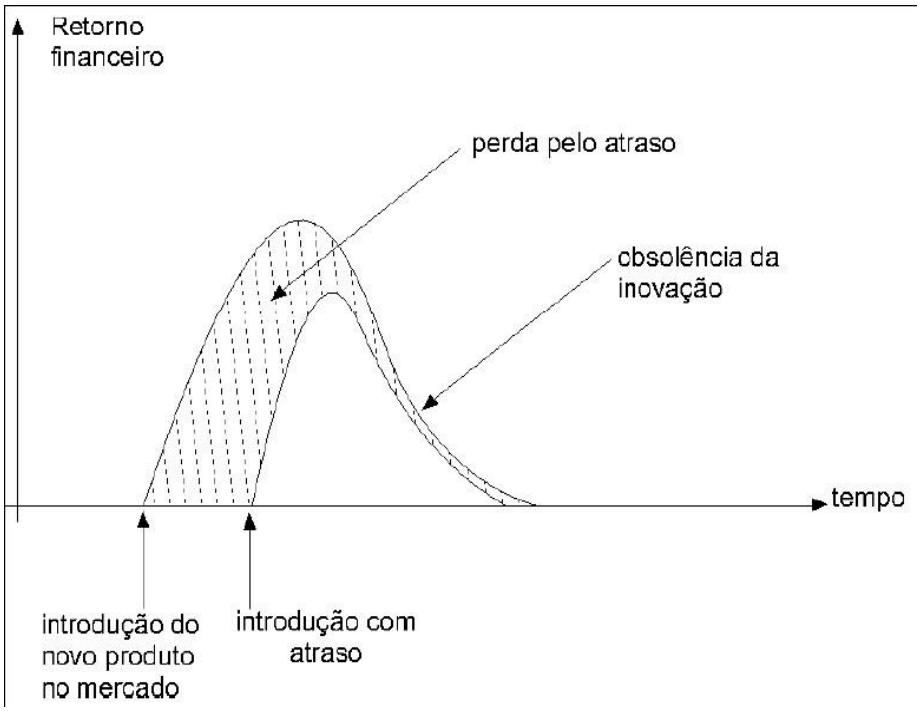

Figura 3.1: Relação entre tempo e retorno financeiro, relacionados com atrasos no desenvolvimento de um produto [Wolf, 2001].

Juntamente ao *hardware* a ser desenvolvido, *softwares* específicos, para o Sistema Embarcado, tornaram-se necessários. Estes *softwares* possibilitam que o usuário tenha maior facilidade em utilizar as variadas funcionalidades presentes no ES. Exemplos práticos são os celulares e *Smartphones*, onde a cada nova geração lançada, novas funcionalidades são desenvolvidas e anexadas aos dispositivos que passam a fornecer mais recursos aos usuários.

A integração dos módulos de *hardware* e de *software* dedicados ao *hardware,* além de questões financeiras, fez com que algumas características se tornassem fundamentais no processo de desenvolvimento de um ES. Dentre eles:

a) consumo de potência: em um ES, sua fonte de alimentação provém geralmente de baterias, portanto, quanto maior a duração da "carga" desta fonte de energia maior será a

eficiência do sistema. Para reduzir o consumo de energia, menor deverá ser a movimentação de dados. Esta redução torna mais rápida uma resposta que deverá ser fornecida. Este tópico tem sido fonte de constantes pesquisas aos desenvolvedores;

b) desempenho: quanto maior o número de funcionalidades presentes, maior deve ser a eficiência do sistema garantindo alto processamento, atendendo todas as solicitações e fornecendo respostas rápidas e precisas;

c) peso e tamanho: derivados de computadores de propósito geral, os ES apresentam uma redução estrutural por executar atividades dedicadas [Fisher et al., 2005]. Com a inclusão de novas funcionalidades, novos módulos de *hardware* devem ser anexados e uma nova organização deve ser desenvolvida. Isso garante conforto e praticidade a quem for utilizar este Sistema Embarcado, como celulares, por exemplo;

d) segurança: muitos sistemas embarcados são utilizados em dispositivos que, de forma direta ou indireta, envolvem o bem estar e a segurança de pessoas. Dentre estes dispositivos podem ser citados sistemas de controle de tráfego aéreo, freios ABS, robôs utilizados em cirurgias, entre outros;

e) tolerância à falhas: um ES, quando na ocorrência de uma falha, deve ser capaz de continuar a executar as demais atividades para as quais é programado, informando aos usuários o problema ocorrido. Quando um erro ocorrer no sistema, não deve acarretar em parada completa da execução do mesmo, para que medidas de recuperação possam ser tomadas por parte dos usuários;

f) tempo de projeto: atualmente o desenvolvimento de Sistemas Embarcados é voltado ao mercado consumidor buscando inovações tanto em tecnologia envolvida como em *design*, atendendo requisitos de *time-to-market;*

g) custo: voltado ao mercado consumidor, um ES deve ser desenvolvido de forma eficiente no âmbito financeiro, uma vez que o custo do seu desenvolvimento é imposto no produto final.

Para garantir o atendimento dessas características, novas tecnologias passaram a ser desenvolvidas. Dentre essas inovações se destacam a mudança na tecnologia de processadores como utilização da tecnologia VLSI (*Very Large Scale Integration*), que em 1970 possibilitou a utilização de uma completa CPU em um único chip [Wolf, 2001]. Além disso, sistemas com

mais de um núcleo passaram a ser criados, juntamente com novos meios de intercomunicação entre estes núcleos, aumentando assim a eficiência e desempenho de um ES.

3.1 Processadores Embarcados

Desenvolvidos a partir de 1972, quando a Intel [Intel, 2010a] lança seu primeiro modelo de sucesso [Junior, 2009], os microprocessadores passaram a representar a base na tomada de decisão de um projeto de Sistemas Embarcados. Atualmente, a maior parte dos processadores embarcados utiliza a tecnologia RISC (*Reduced Instruction Set Computer*) [Patterson et al., 2007], ao invés da tecnologia CISC (*Complex Instruction Set Computer*).

No modelo CISC, o processador apresenta um elevado conjunto de instruções sintetizadas em *hardware*. Muitas dessas instruções acabam não sendo utilizadas, ou são utilizadas em poucos casos, elevando apenas a complexidade do desenvolvimento do *hardware*. Em contra partida, numa arquitetura RISC, o processador apresenta apenas um número mínimo de instruções, as instruções básicas. Neste modelo, a complexidade é retirada do processador e inserida no compilador, uma vez que as instruções mais complexas devem ser convertidas em instruções básicas pelo compilador, e repassadas ao processador para execução.

No desenvolvimento de ES, inúmeras arquiteturas passaram a ser desenvolvidas, tendo maior destaque os modelos ARM (*Advanced RISC Machine*) [ARM, 2010], SPARC (*Scalable Processor Architecture*) [Lamboia, 2008], MIPS (*Microprocessor without Interlocked Pipe Stages*) [MIPS, 2010] [Lamboia, 2008] e PowerPC (*Power Optimization With Enhanced RISC – Performance Computing*) [Marsala e Kanawati, 1994].

Processadores e micro-controladores embarcados estão em transição de controladores de 8/16 *bits* para arquiteturas 32 *bits* [Junior, 2009], proporcionando, assim, o surgimento de novas soluções, com novas características de desempenho e potência.

Baseado nos projetos RISC I e RISC II da Universidade de Berkeley, a arquitetura SPARC foi desenvolvida pela Sun Microsystems [Sun, 2010] entre os anos de 1984 e 1987 [Dettmer, 1990]. Baseado em arquitetura RISC de 32 *bits,* o SPARC apresenta *pipeline* de quatro estágios. A *pipeline* é constituída por um conjunto de partes, onde cada parte é responsável por manipular um conjunto de recursos do sistema e estas partes somadas, completam a

execução de uma instrução. A utilização comercial do SPARC é livre, com isso surgem aperfeiçoamentos por parte das empresas que utilizam esta arquitetura, apresentando um bom desempenho e pouca dissipação de potência [Junior, 2009] [Dettmer, 1990]. Sua utilização é presente em *workstations* da Sun, bem como servidores da Sun e outras empresas.

A arquitetura MIPS utiliza o conceito RISC, sendo criada baseada no conceito VLSI, onde as instruções apresentam tamanho elevado e é característica dos computadores de propósito geral, em 1982 na Universidade de Stanford [Junior, 2009] [Hennessy et al., 1982]. Inicialmente projetado para instruções 32 *bits,* atualmente suporta instruções de 64 *bits*. O MIPS eliminou o processo de execução sequencial de *pipelines*, onde toda a *pipeline* era executada para, apenas em seguida, outra começar sua execução.

Cada parte de uma *pipeline* consome um conjunto específico de recursos do sistema. Finalizada essa parte esses recursos não são utilizados até a execução de uma outra *pipeline*. O MIPS buscou remover esses períodos de tempo em que recursos do processador ficavam ociosos, e inseriu também, o conceito de interrupções, tratamento de perda de páginas e suporte a *overflow* conseguindo um elevado ganho de desempenho [Hennessy et al., 1982]. Assim como a arquitetura SPARC, o MIPS apresenta bom desempenho e reduzido gasto de potência [Junior, 2009]. Essa tecnologia está sendo utilizada em gravadores e reprodutores de DVD (*Digital Vídeo Disc*), câmeras digitais, aparelhos celulares, GPSs (*Global Positioning System*), televisores, consoles de vídeo-game, entre outros [MIPS, 2010]. Além desses dispositivos, a tecnologia MIPS esta sendo utilizado para a área de ensino, como exemplo tem-se a carteira digital [Mammana et al., 2009], que é voltada a alunos de ensino médio e tem por objetivo ser utilizada em todo o país.

A arquitetura ARM é uma arquitetura de 32 *bits,* desenvolvida entre os anos de 1983 e 1985 na Acorn Computer Limited, Cambridge, Inglaterra [Silvestre e Bachiega, 2007]. Baseado na arquitetura RISC, os processadores ARM apresentam simplicidade em seu desenvolvimento, redução de tamanho e baixo consumo de potência [Junior, 2009]. Assim como a SPARC, a ARM não produz *chips,* deixando a cargo de empresas interessadas a produção em silício. Desta forma é possível encontrar versões destinadas a diversas funcionalidades [Silvestre e Bachiega, 2007]. Dentre os produtos que utilizam a tecnologia

ARM estão televisores, câmeras digitais, consoles de vídeo-game, celulares, receptores de satélite, controladores de áudio, entre outros [ARM, 2010] [Silvestre e Bachiega, 2007].

Em 1991, as empresas IBM, Apple e Motorola juntaram esforços para desenvolver um novo microprocessador denominado PowerPC [Marsala e Kanawati, 1994]. Baseado na arquitetura RISC, o PowerPC foi projetado para permitir mudanças na sua implementação com facilidade, tornando a arquitetura flexível [Lamboia, 2008]. Apresenta instruções de 32 *bits* e 64 *bits,* pode ser executado em diferentes sistemas operacionais e apresenta instruções complexas para um modelo de arquitetura RISC [Marsala e Kanawati, 1994] [Lamboia, 2008]. É utilizado em computadores Macintosh da Apple, consoles de vídeo-game, telefones celulares, servidores, entre outros dispositivos [Lamboia, 2008].

Com o surgimento do conceito RISC, uma simplificação no desenvolvimento de processadores foi atingida. Essa simplificação buscou tornar a execução de uma aplicação mais eficiente e com uma redução no consumo de potência. Porém, com o surgimento de aplicações mais complexas, novos conceitos tecnológicos precisaram ser incorporados a estes projetos buscando fornecer esta garantia. A necessidade de um ES apresentar inúmeras funcionalidades num mesmo dispositivo, mantendo uma redução do consumo de potência com alto desempenho é outra característica importante. Na tentativa de atender a estes requisitos, conceitos como processamento paralelo, plataformas multiprocessadas e redes de interconexão passaram a ser desenvolvidas e anexadas a ES.

3.2 Histórico da Evolução do Processamento Paralelo

A constante evolução apresentada pela área do conhecimento em computação proporcionou o surgimento constante de novas tecnologias. A utilização de *pipelines*, aplicação da lei de Moore (para desenvolver *chip* com elevado número de transistores, possibilitando a elevação da frequência de operação e aumento de lógica disponível), o desenvolvimento das arquiteturas superescalares e a adoção de *threads* são exemplos de inovações desenvolvidas para promover uma constante evolução na área de processadores. Essas tecnologias e conceitos auxiliaram o desenvolvimento de processadores de alto

desempenho, envolvendo todas as possibilidades de arquiteturas, sistemas e *chip* sejam multi-núcleo ou multi-processo.

3.2.1 Pipeline

Em 1972, a Intel lança seu primeiro microprocessador de sucesso, o Intel 4004 [Wolf, 2001]. Desde este acontecimento, a área computacional vem apresentando constante evolução no desenvolvimento de CPUs. Neste período, uma das importantes inovações tecnológicas a ser inserida em processadores foi a utilização de *pipelines.* Melhorias nessas estruturas possibilitaram o desenvolvimento de arquiteturas mais eficientes e com um maior desempenho.

Pipelining é uma técnica de implementação onde múltiplas instruções são sobrepostas em execução, buscando tornar a CPU mais rápida [Patterson e Hennessy, 1996]. O início das pesquisas sobre a utilização de *pipelines* ocorreu na década de 50, sendo deixado de lado entre os anos 1970 e 1980, onde o foco de estudos tecnológicos eram conceitos como a evolução da tecnologia CISC, sendo retomada sua importância após 1980 [Pizzol, 2002]. Com as arquiteturas RISC, a utilização de *pipelines* voltou a ser foco de pesquisas, pois, para reduzir a estrutura apresentada pelas instruções, novos conceitos e estruturas de *pipelines* deveriam ser pesquisados [Pizzol, 2002].

A Figura 3.2 [Pizzol, 2002] apresenta um modelo de *pipeline* com cinco estágios. Quando for executada, uma *pipeline* é dividida em pequenas partes, onde cada uma destas partes é executada em uma fração de tempo, sendo cada uma destas partes denominada de estágio [Pizzol, 2002]. Neste exemplo, o primeiro estágio (BI) busca a instrução, o segundo (DI) decodifica a instrução buscada, sendo o terceiro estágio (EX) responsável pela execução da mesma [Pizzol, 2002]. Se for necessário um acesso à memória, é no estágio quatro (MEM) que este processo ocorre [Pizzol, 2002]. A finalização da *pipeline* ocorre no estágio cinco (AR), com a atualização dos registradores do sistema [Pizzol, 2002].

Inicialmente, a execução de instruções era realizada de forma sequencial, onde uma *pipeline* era iniciada e finalizada, sem interrupções. Porém, na arquitetura MIPS, esse conceito foi repensado [Hennessy et al., 1982]. Durante a execução de uma *pipeline*, cada estágio consome um ciclo e determinados recursos do sistema. Quando o estágio é finalizado,

os recursos que foram utilizados são liberados. Na arquitetura MIPS, uma *pipeline* tem início com um ciclo de atraso em relação a *pipeline* anterior, assim em cada ciclo é utilizado o máximo de estágios possíveis, como mostrado na Figura 3.3 [Pizzol, 2002], desta forma o tempo final para executar um conjunto de *pipelines* é reduzido consideravelmente [Hennessy et al., 1982]. A Figura 3.3 [Pizzol, 2002] mostra esse processo de execução, onde três *pipelines* são iniciadas sem a necessidade do término da primeira.

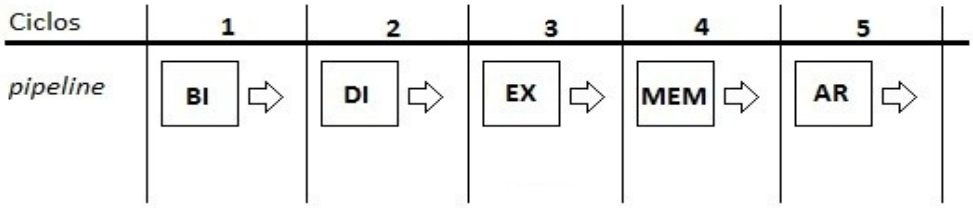

Figura 3.2: Modelo de *pipeline* com cinco estágios [Pizzol, 2002].

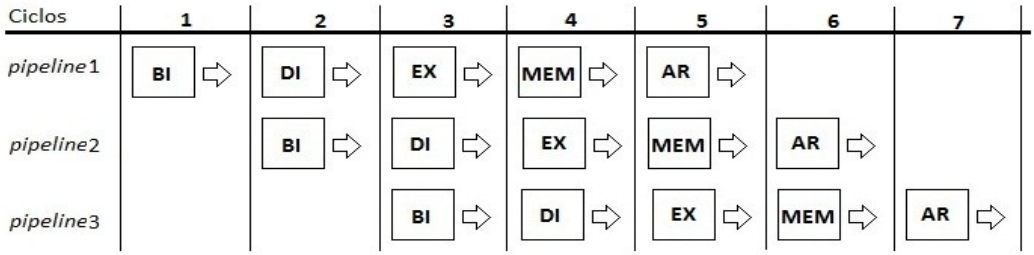

Figura 3.3: Modelo de *pipeline* com cinco estágios com três *pipelines* em execução [Pizzol, 2002].

Essa forma de encadeamento das *pipelines* possibilitou ganho de desempenho se comparado à execução sequencial de *pipelines*. Isso ocorre devido ao número total de ciclos gastos para executar um conjunto de *pipelines* encadeadas ser menor se comparada com a execução sequencial. Porém alguns problemas acabam ocorrendo quando esta técnica é utilizada. A resolução destes problemas acabam por tornar a utilização otimizada de *pipelines* mais complexas. Dentre esses problemas estão:

a) dependência de controle, perceptível quando ocorre um desvio de execução, necessitando saber qual o caminho certo a ser tomado;

b) conflito de leitura, onde uma instrução pode solicitar um dado proveniente de uma leitura anterior;
c) conflitos de escrita, presente quando um dado precisa ser escrito, mas pode estar desatualizado na memória do processador;
d) conflitos de endereços, assim como no conflito de leitura, pode solicitar um dado proveniente de uma instrução anterior;
e) dependência de dados que ocorre quando existe uma dependência entre *pipelines* consecutivas [Pizzol, 2002].

3.2.2 Lei de Moore

A evolução do *hardware* que compunha os processadores contribuiu para uma evolução acentuada nas arquiteturas computacionais. Essa evolução segue os conceitos preditos por Gordon Moore, co-fundador da Intel, citadas em 1965, conhecida como lei de Moore. Ela estimava que a cada dois anos o número de transistores presentes em um *chip* seria duas vezes maior, sem implicar no tamanho do mesmo, aumentando consequentemente a velocidade de execução dos mesmos [Gaudin, 2010]. O transistor é o componente eletrônico responsável pela passagem de corrente elétrica em um processador. O chaveamento realizado por um conjunto de transistores, forma as portas lógicas que compõem o processador. A Figura 3.4 [Junior, 2009] apresenta a evolução apresentada pelos processadores ao longo dos anos, tanto no número de transistores como na velocidade alcançada em execução.

A indústria de processadores, desde o surgimento da lei de Moore, passou a seguir seus preceitos, buscando constantemente reduzir o tamanho e aumentar o número dos transistores a serem utilizados em processadores. Como prova disso tem-se a comparação entre o microprocessador Intel 4004 lançado no início da década de 1970 que apresentava pouco mais de 2000 transistores, com a família de processadores Penryn da Intel, lançada em 2007, que apresenta 820 milhões de transistores [Gaudin, 2010].

Atualmente os transistores estão chegando à redução limite de seu tamanho. Consequentemente alguns problemas acabam surgindo, como por exemplo, elevação da dissipação de calor. Este fato ocorre devido ao aumento da circulação de energia nos transistores e, com isso, surge à necessidade de novas técnicas de resfriamento [Garcia, 2008].

Além disso, a potência média dissipada aumenta com a elevação do número de transistores presentes num *chip*. Isso ocorre uma vez que, além de possuir uma dissipação estática, perceptível quando não ocorre chaveamento, apresenta, também, uma dissipação dinâmica necessária para executar uma aplicação [Garcia, 2008]. Estudos comprovam que, para processadores com transistores de 70nm de tamanho, a potência estática dissipada ultrapassa a potência dinâmica necessária [Garcia, 2008]. Desta forma, o consumo de potência acaba sendo definido pelo número de transistores, e não pelo gasto para executar uma aplicação.

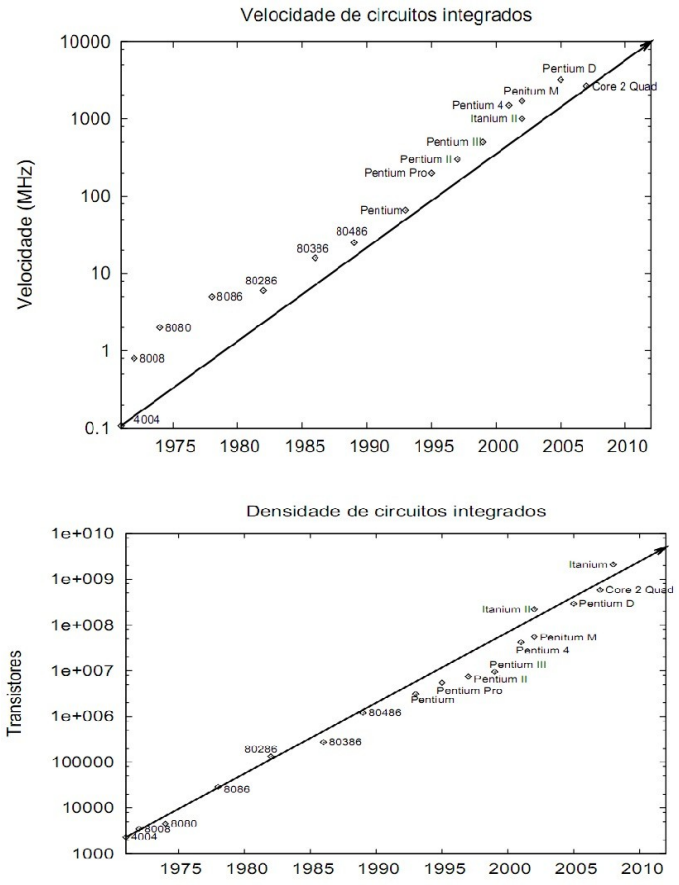

Figura 3.4: Densidade e velocidade de circuitos integrados segundo a lei de Moore [Junior, 2009].

Dentre as características essenciais para ES, a energia dissipada para a execução de uma aplicação é uma das mais importantes, uma vez que esses sistemas geralmente são alimentados por baterias. Portanto, dados como os anteriores não podem ser ignorados.

Buscando solucionar este problema, novos conceitos relacionados com a execução de aplicações surgem e passam a ser incorporados ao desenvolvimento de sistemas embarcados. Dentre as soluções mais exploradas estão os conceitos de paralelismo, onde o objetivo é, utilizando processadores de menores frequências, realizar todas as tarefas necessárias da forma mais eficiente e rápida possível.

3.2.3 Arquiteturas superescalares

O termo escalar denota o conceito de possibilitar a fragmentação de uma instrução. Nesta fragmentação, cada parte é executada separadamente, consumindo um determinado tempo, garantindo coerência, bem como, a possibilidade de executar instruções simultaneamente, utilizando o conceito de processamento paralelo [Pizzol, 2002]. Sendo uma adaptação das arquiteturas com *pipelines*, as arquiteturas superescalares foram desenvolvidas buscando aumentar o ganho de desempenho e eliminar os problemas apresentados na manipulação de *pipelines*.

Desenvolvidos a partir de 1980, as arquiteturas superescalares surgiram como forma de executar múltiplas instruções de múltiplas *pipelines* em um mesmo período de tempo, buscando assim a eliminação da execução de apenas uma instrução por ciclo [Smith e Sohi, 1995]. Para garantir que os conceitos de paralelismo pudessem ser atendidos, segundo SMITH e SOHI [Smith e Sohi, 1995], as arquiteturas superescalares passaram a implementar as características de:

a) busca simultânea de múltiplas instruções, tentando identificar e buscar futuras instruções de desvio;
b) métodos de determinação de dependências reais entre valores de registradores e formas de relacionar esses valores quando for necessário;
c) métodos de inicialização de múltiplas instruções de forma paralela;
d) execução paralela de múltiplas instruções com funções de acesso a uma hierarquia de memória;
e) métodos para realizar um interligamento de valores de dados através de instruções de *load* e *store*, permitindo uma interação com a hierarquia de memória presente na arquitetura;

f) métodos para garantir que a execução do processo ocorra de forma correta.

A inclusão destas funcionalidades nas arquiteturas superescalares garantiu que os problemas iniciais da otimização de *pipelines* fossem resolvidos, conseguindo um ganho significativo de desempenho [Pizzol, 2002]. Atualmente o uso de arquiteturas superescalares é o padrão tecnológico adotado pela maioria dos fabricantes de microprocessadores [Pizzol, 2002]. Assim como houve aumento de desempenho, aumentou também a complexidade no desenvolvimento destas arquiteturas [Garcia, 2008]. Isso ocorre porque o desenvolvimento de mecanismos gerenciadores de dependências tende a se agravar com o aumento do número de instruções executadas em paralelo, limitando a arquitetura [Garcia, 2008] [Pizzol, 2002]. Outras formas, que visam proporcionar maior ganho de desempenho, passaram a ser desenvolvidas utilizando conceitos, como *threads*, na tentativa de buscar um paralelismo de mais alto nível [Garcia, 2008].

3.2.4 Paralelismo de *threads*

A utilização de *threads*, diferente das alternativas anteriores é uma solução voltada ao *software*, sobre a execução de uma aplicação. O conceito de *threads* surgiu em 1979 com desenvolvimento do sistema operacional Toth, possibilitando que o espaço de endereçamento de um processo pudesse ser acessado por várias linhas de execução [Maia, 1998]. No ano seguinte, na Universidade de Carnegie Mellon, foi realizada a primeira diferenciação entre os conceitos de processos e *threads*, com o desenvolvimento do sistema operacional Mach [Maia, 1998]. Desde este acontecimento, a evolução do conceito de *thread* é constante, sendo esta técnica de paralelismo utilizada em inúmeros processadores modernos [Xinmin et al., 2003].

No momento que as *threads* passaram a ser utilizadas, era utilizada uma *thread* por processo. Porém, com a programação multi-processo, existia a possibilidade de criar e executar um número variado de *threads,* onde suas respectivas execuções aconteciam de forma concorrente [Maia, 1998]. Para que todos os processos fossem executados ocorria uma divisão do tempo de uso da CPU, entre todas as *threads* onde apenas uma executava num determinado instante, ficando as demais bloqueadas nesse período de tempo.

Esta configuração trouxe problemas para execuções paralelas. Cada *thread* aloca recursos específicos elevando o montante total de recursos utilizados. Além disso, ocorre a dificuldade na comunicação entre processos, uma vez que, cada processo tem seus próprios recursos. Com isso, são necessários mecanismos complexos que permitam:

a) uma interação e coerência entre os dados, uma vez que cada *thread* possui seu próprio espaço de endereçamento. Além disso, cada *thread* possui recursos de *hardware* e *software* e um PC (*Program Counter* – Contador de Programas) próprio que indica qual a próxima instrução a ser executada;
b) uma livre movimentação de dados entre as estruturas de memórias utilizadas pelos processos [Maia, 1998].

A Figura 3.5 [Maia, 1998] exemplifica um modelo de sistema onde existe apenas uma única *thread* responsável por executar um determinado processo, denominado ambiente *monothread.* Como exemplo de utilização de ambientes *monothread,* pode ser citado o sistema operacional MS-DOS (*MicroSoft Disc Operating System* – Sistema Operacional em Disco da MicroSoft).

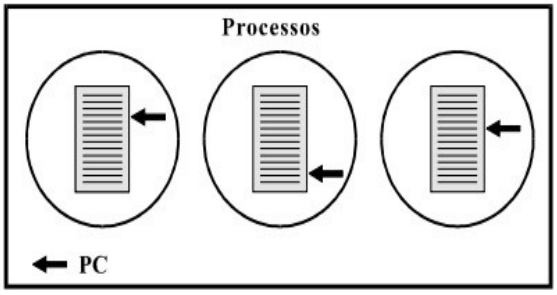

Figura 3.5: Uma organização *monothread* [Maia, 1998].

Para solucionar os problemas apresentados nos ambientes *monothread*, passaram a desenvolver ambientes *multithread*, onde varias *threads* são criadas para a execução de um mesmo processo [Maia, 1998]. Nos ambientes *multithread*, cada *thread* apresenta um conjunto de registradores, contexto de *hardware* e PC individual. O espaço de endereçamento e os recursos de *software* são compartilhados entre todas as *threads* que executam um mesmo

processo. Desta forma, o acesso a dados é mais rápido, possibilitando a execução de processos de forma paralela quando existem mais de um processador na arquitetura [Maia, 1998].

Desta forma as muitas *threads* que executam um determinado processo, são vistas pelo processador como um único processo, como pode ser observado na Figura 3.6 [Maia, 1998]. Desta forma, o desempenho da aplicação é mais elevado, com maior rapidez de acesso a dados, possibilidade da realização de atividades paralelamente, quando não houver uma dependência de dados [Maia, 1998]. Além disso, existe a possibilidade de executar mais processos, uma vez que o total de recursos alocados para cada processo é reduzido [Maia, 1998]. Sistemas como Solaris [Sun, 2010] e Windows 2000, são exemplos da utilização de *multithread*.

O elevado número de processos que devem ser suportados em uma arquitetura fez com que o conceito de paralelismo fosse amplamente estudado. O conceito de *multithread* passou a ser utilizada juntamente com arquiteturas superescalares, para poder obter o maior rendimento possível das vantagens apresentadas por ambas as arquiteturas [Levy et al., 1996]. A combinação destas duas tecnologias possibilitou que os problemas para criação de arquiteturas superescalares fossem reduzidos [Levy et al., 1996]. O impacto da utilização de ambientes *monothread* foi minimizado e o desempenho final das arquiteturas *multithread* foi elevado [Levy et al., 1996]. Segundo estudos apresentados houve a possibilidade da execução de 5.4 instruções por ciclo de relógio do processador, representando uma elevação de desempenho de 37% comparada a arquiteturas similares [Levy et al., 1996].

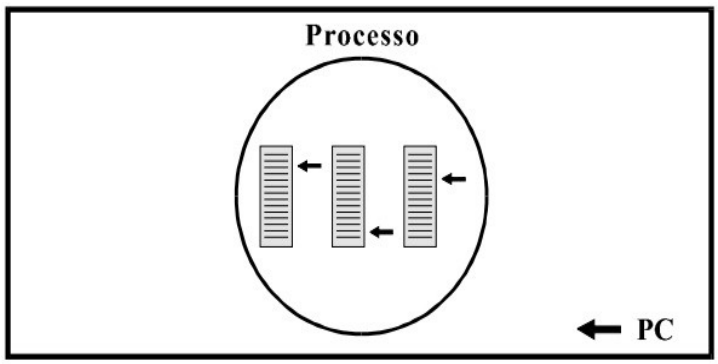

Figura 3.6: Uma organização *multithread* [Maia, 1998].

Além da utilização de ambientes *multithread* em arquiteturas superescalares, passou a ser desenvolvido um novo conceito envolvendo *threads*, denominado *Hyper-Threading* [Xinmin et al., 2003]. Este conceito foi introduzido pela Intel em 2002, utilizando o processador Intel Pentium IV como base, onde fazendo uso de um ambiente *multithread,* em um único processador foi criada uma visão da existência de dois processadores [Xinmin et al., 2003]. Utilizando este conceito é possível executar aplicações de forma paralela ou executar uma aplicação em dois processadores [Xinmin et al., 2003]. Desta forma, o conceito de *thread* passou de um nível de *software* para um nível de *hardware*.

A simulação de um segundo processador possibilita a redução do tempo de execução de uma determinada aplicação. Caso esta aplicação for desenvolvida com suporte ao uso de *threads* os dados serão compartilhados, e esta aplicação será executada por dois processadores simultaneamente [Xinmin et al., 2003]. Como exemplos de aplicações que apresentam suporte a esta tecnologia, estão aplicações de áudio e vídeo [Xinmin et al., 2003]. Além da simulação paralela de uma mesma aplicação, o *Hyper-Threading* possibilita que duas aplicações executem de forma paralela, uma vez que existe a simulação de um segundo processador [Xinmin et al., 2003].

O conceito de *Hyper-Threading* serviu de base para o desenvolvimento da tecnologia Dual Core lançada pela Intel em 2005 [Intel, 2010a]. Na Figura 3.7 [Intel, 2010a] é possível visualizar como é a organização de uma arquitetura *Hyper-Threading*. Neste modelo, apenas os registradores são divididos entre cada processador lógico, ficando todos os demais recursos compartilhados. A utilização dessa tecnologia possibilitou o desenvolvimento de processadores com elevado desempenho e velocidade, aproveitando modelos antigos de frequências não tão elevadas.

Desta forma a utilização de *threads* possibilitou o desenvolvimento de novas tecnologias, reutilizando conceitos e técnicas já desenvolvidas. Porém, o número de aplicações a serem executadas simultaneamente é cada vez mais elevado, mesmo utilizando inúmeras *threads*, a partir de certo ponto os recursos disponibilizados por um processador acabam se esgotando. Além disso, o fato do tempo gasto para o processador poder realizar uma troca de execução entre processos diferentes, consomem escalas cada vez maiores do tempo de processamento, inviabilizando a utilização de apenas um núcleo físico [Garcia, 2008]. Desta forma, soluções

que apresentam mais de um processador começaram a surgir, possibilitando que inúmeras aplicações possam ser executadas paralelamente, elevando o desempenho final de uma arquitetura.

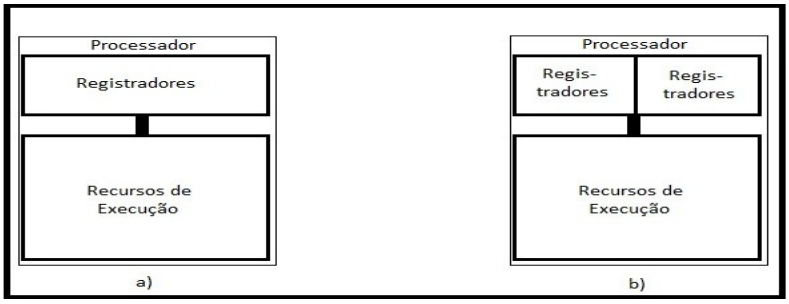

Figura 3.7: Um processador sem *Hyper-Threading* (a) e um processador com *Hyper-Threading* (b) [Intel, 2010a].

3.2.5 Arquiteturas Multiprocessadas

O termo arquiteturas multi-núcleos surgiu em 1989, quando Gelsinger e colaboradores [Gelsinger et al., 2000], publicaram um artigo onde previam a existência de arquiteturas com mais de um processador em meados do ano 2000. Essa previsão se baseava nos conceitos da possibilidade e existência de um grande número de aplicações sendo executadas numa arquitetura, a uma melhor organização de recursos como memória e um possível limite da minimização dos componentes que integram um processador [Gelsinger et al., 2000]. Atualmente o número de arquiteturas para computadores de propósito geral que apresentam múltiplos processadores, sendo denominadas *multicores*, são elevadas, conseguindo obter elevado desempenho com baixo custo.

Na área de ES, os conceitos de arquiteturas multi-núcleos passou a ser utilizada por Jerraya em 2004 onde, para designar uma arquitetura com múltiplos núcleos em um chip, utilizou o termo MPSoC [Jerraya, 2004]. Este conceito passa a ser utilizado para representar uma estrutura que apresente um variado número de processadores. Onde cada um com, ou sem, uma memória local, interligados via uma barramento, podem ainda apresentar módulos de memória compartilhados e módulos de leitura e escrita em periféricos.

O termo multiprocessado está ligado ao fato de poder compartilhar o espaço de endereçamento [Garcia, 2008]. Desta forma, com endereços compartilhados, todos os processadores podem realizar atividades sobre uma mesma aplicação, possibilitando um ganho significativo de desempenho. O compartilhamento de endereços é possível através da criação de memórias globais. Essas memórias são módulos específicos para armazenar dados, que podem ser acessados a qualquer momento por qualquer processador. Existe ainda a possibilitando de todos os processadores acessarem a memória local de cada processador.

Para ambos os casos, é necessário o desenvolvimento de métodos que garantam que os dados manipulados por um processador não gerem dados inconsistentes em outros processadores. Esses métodos são responsáveis por atribuir uma maior complexidade no desenvolvimento desses sistemas, pois, quanto mais processadores envolvidos, um maior controle deverá ser exercido, com isso, o sistema pode se tornar mais lento [Garcia, 2008]. Métodos que visam exercer esta função são chamados de método de coerência de memória. Alguns desses métodos foram apresentados no Capítulo 2 deste trabalho.

As arquiteturas multiprocessadas são definidas segundo o tipo de compartilhamento de memória que apresentam [Garcia, 2008]. Os tipos de compartilhamento de memória podem ser aquelas que provêem um acesso uniforme (UMA – *Uniform Memory Access*) ou podem apresentar acesso não uniforme (NUMA – *Non Uniform Memory Access*) [Junior, 2009].

Um terceiro modelo é denominado COMA (*Cache Only Memory Access*). O modelo COMA defende a utilização de apenas uma memória cache, de elevado tamanho, em cada processador [Garcia, 2008]. Desta forma ocorreria uma descentralização no acesso aos dados, elevando o desempenho da arquitetura [Garcia, 2008], porém, devido ao elevado custo deste modelo de memória, o mesmo se torna inviável.

O modelo de compartilhamento de memória com acesso uniforme (UMA) é representado na Figura 3.8, onde P representa um processador e Barramento é o sistema que realiza a movimentação de dados. Na arquitetura com memória UMA existe um variado número de processadores, onde cada um apresenta uma pequena memória cache. Além disso, existe uma memória global, que é acessada por todos os processadores via uma rede de interconexão [Junior, 2009] [Garcia, 2008]. Essa memória global é controlada por um dispositivo que promove um acesso serializado aos dados [Junior, 2009], gerando uma fila que armazena

todas as solicitações de dados e as atende de acordo com a prioridade, ou ordem de chegada de cada solicitação. Desta forma, quanto mais processadores presentes na arquitetura utilizando esta memória global, maior será o tempo de espera, por parte do processador, para ler ou escrever um determinado dado. Esta tecnologia esta presente nos processadores Core 2 Duo [Intel, 2010b] da Intel, onde dois núcleos são encapsulados, acessando a memória de maneira compartilhada [Garcia, 2008].

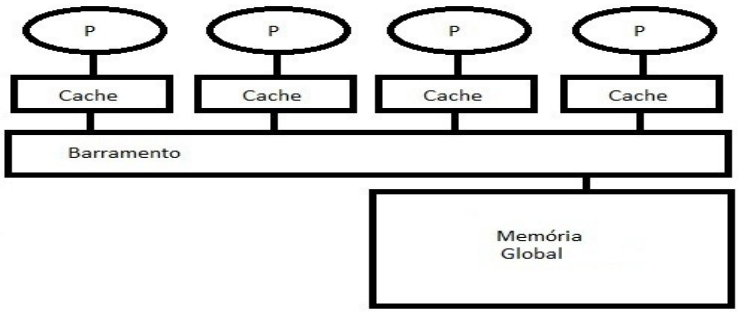

Figura 3.8: Modelo de memória compartilhada UMA.

No modelo NUMA, cada processador apresenta, além da cache, um módulo de memória local, maior e mais lento que a cache. A memória global não é utilizada, como no modelo UMA, com isso, cada processador armazena em sua memória local os dados necessários a sua execução. A Figura 3.9 representa uma modelo conceitual de arquitetura com memória compartilhada segundo o padrão NUMA, onde P é o processador e o barramento é o responsável por fazer a comunicação entre os processadores e suas memórias. Neste modelo, não é mais o meio de interconexão que realiza o acesso direto aos dados. Em cada processador deve ser incluído um controlador de memória, responsável por fornecer os dados requeridos via barramento por outro processador [Junior, 2009] [Garcia, 2008].

A existência da cache e também de uma memória local acarreta um aumento da complexidade no desenvolvimento de sistemas multiprocessados [Junior, 2009] [Garcia, 2008]. Quando existe uma manipulação local e um armazenamento global, podem ocorrer inconsistências decorrentes do fato de um dado presente na memória local ser modificado e não atualizado na memória global antes da realização da leitura por outro processador. Desta

forma é necessário a elaboração de métodos que garantam que dados desatualizados não sejam utilizados por processadores [Junior, 2009] [Garcia, 2008].

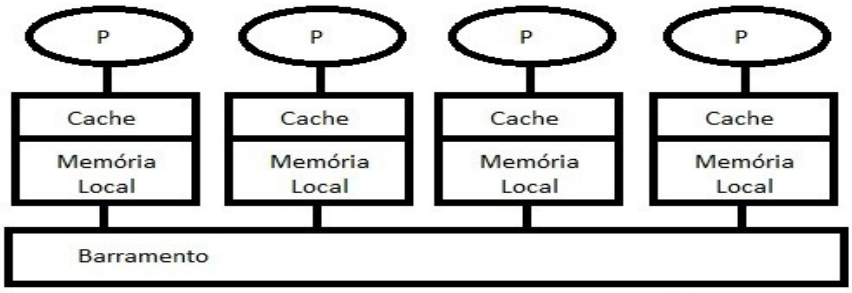

Figura 3.9: Modelo de memória compartilhada NUMA.

Além da coerência de cache, o acesso a memória, via meios de inter-conexão, também eleva a complexidade de uma arquitetura multiprocessada, dependendo do modelo adotado. Esta rede de inter-conexão pode ser um barramento, que são linhas de interligam os componentes, permitindo a movimentação de informações, ou ainda pode ser uma estrutura mais complexa semelhante as redes utilizadas pela Internet. Devido a constante movimentação de dados, é necessário que uma estrutura adequada seja utilizada evitando que o processador fique muito tempo ocioso esperando que os dados necessários sejam repassados a ele. O desenvolvimento de meios eficientes de inter-conexão vem sendo algo de constantes estudos na área de ES, pois, sua eficiência ou não, irá determinar o desempenho apresentado pela arquitetura [Garcia et al., 2010].

3.3 Redes de Interconexão

O surgimento das arquiteturas multiprocessadas possibilitou um acentuado nível de paralelismo na execução de aplicações em uma arquitetura. Ao mesmo tempo, anexando mais processadores numa mesma arquitetura, espaços de endereçamento compartilhados precisaram ser incluídos. Para realizar a movimentação de dados entre todos os processadores que formam uma arquitetura multiprocessada, redes de interconexão escaláveis e eficientes passaram à ser desenvolvidas.

A necessidade de desenvolver uma estrutura de comunicação eficiente, além de possibilitar a comunicação entre processadores, possibilita que outros módulos possam ser anexados à arquitetura, como periféricos e módulos de memória. Além disso, a arquitetura deve possibilitar que mais processadores sejam anexados, sem a necessidade de mudar toda a estrutura da rede de comunicação já utilizada [Kumar et al., 2003]. Desta forma, uma rede de comunicação presente em uma arquitetura multiprocessada deve apresentar características de estruturação, escalabilidade, reusabilidade e alto desempenho [Kumar et al., 2003].

Relacionado à estruturação, uma rede de interconexão deve possuir tamanho reduzido, apresentando boas propriedades relacionadas a plano de desenvolvimento e análises de desempenho e consumo de potência [Kumar et al., 2003]. A escalabilidade indica que a arquitetura deve possibilitar utilizar um número variado de processadores, não sendo o número de processadores uma limitação da arquitetura [Kumar et al., 2003]. Reusabilidade permite que, quando um novo projeto for desenvolvido, o programador responsável necessite apenas fornecer as especificações da nova arquitetura e o número de processadores. Desta forma, não há a necessidade de criar uma nova rede de interconexão, pois esta apresentará suporte a uma nova configuração [Kumar et al., 2003]. Além de que, um alto desempenho é obtido possibilitando que um número elevado de processadores utilize esse sistema de interconexão de forma paralela [Kumar et al., 2003]. Somado a estas características, o consumo de potência não pode ser desconsiderado [Ost, 2004]. Dependendo do modelo de interconexão adotado, para realizar uma movimentação de dados, dependendo da distância percorrida e dos caminhos adotados, poderá acarretar em um consumo maior ou menor de energia.

Na tentativa de desenvolver redes de interconexão eficientes, vários grupos de pesquisadores, passaram a desenvolver variados modelos de redes *intra-chip*, tendo como base as redes utilizadas em computadores de propósito geral [Kumar et al., 2003]. Dentre os sistemas que foram projetados tem destaque as ligações ponto-a-ponto e multi-ponto, o sistema de barramento compartilhado e, mais recentemente, as NoCs [Ost, 2004] [Junior, 2010]. Esses modelos serão discutidos nas sessões subsequentes.

3.3.1 Ligação ponto-a-ponto

Nas arquiteturas que possuem este tipo de ligação, existe uma comunicação direta entre dois processadores. Analisando a estrutura de *hardware.* Esta ligação é um fio que conecta dois processadores, também chamado de fio dedicado (*dediated wire*) [Ost, 2004]. Este tipo de estrutura de comunicação é o mais simples a ser utilizado em uma arquitetura multiprocessada, sendo um exemplo deste modelo, demonstrado na Figura 3.10.

Sua principal vantagem está no fato de cada ligação ocorrer apenas entre dois processadores. Desta forma, por não existir uma interligação direta entre todos os processadores, múltiplas transações podem ocorrer de forma paralela, uma em cada par de processadores [Junior, 2010]. Além disso, o fato de um canal ser independente dos demais possibilita que melhorias no fluxo de movimentação de informações sejam realizadas em determinado canal, sem a necessidade de replicar esta ação aos demais canais presentes na arquitetura [Junior, 2010]. Porém, este fato pode gerar uma complexidade mais elevada para sua correta execução e desenvolvimento.

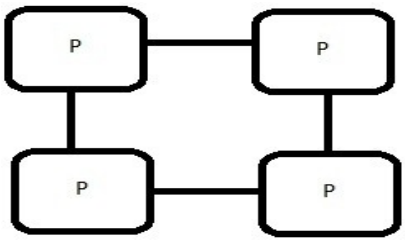

Figura 3.10: Modelo de intercomunicação ponto-a-ponto.

Por ser um único fio ou conjunto de fios, que interliga(m) dois processadores, a movimentação de informações percorre um espaço reduzido acarretando em menor consumo de energia. Diminui, também, o espaço de tempo entre a realização de uma solicitação de dados e o recebimento deste dado solicitado, processo definido como latência de comunicação, e possibilita maiores frequências de *clock* [Zeferino e Susin, 2003]. Elevando a frequência, o tempo gasto para a realização de um *clock* é reduzido, uma vez que essa relação é inversamente proporcional. Desta forma em um mesmo intervalo de tempo, com uma frequência maior podem ser realizados mais *clock*, e, portanto, mais instruções.

Um dos problemas deste modelo é a necessidade de, para cada nova ligação, em um determinado processador, um novo conjunto de portas de comunicação precisa ser inserido no processador [Kumar et al., 2003], necessitando alterar a estrutura básica do mesmo. Além disso, quando um processador possuir mais de uma ligação, este passa a atuar como um roteador, onde deve definir qual caminho tomar para obter um dado, uma vez que terá um número elevado de caminhos possíveis [Kumar et al., 2003]. Somado a isto, apresentará tempo impreciso para o cumprimento de uma solicitação, pois não sabe exatamente onde o dado se encontra nem qual a distância necessária para obter o dado [Kumar et al., 2003]. Além do que, quanto mais tempo gastar, maior será a probabilidade de perda de qualidade do sinal [Kumar et al., 2003]. Como muitas arquiteturas podem apresentar um número elevado de processadores, o montante de linhas de comunicação acabaria por aumentar o tamanho do dispositivo. Como os processadores utilizados neste tipo de arquitetura sofrem alterações físicas, a reusabilidade dos mesmos é prejudicada, implicando em elevação nos custos de produção, uma vez que se tornaria uma produção específica [Ost, 2004] [Junior, 2010].

Apesar de, inicialmente, apresentar vantagens em sua aplicação, o modelo de ligação ponto-a-ponto aplicado em arquiteturas multiprocessadas é inviável [Kumar et al., 2003]. Fato decorrente da reusabilidade não ser possível. Devido a isto, outras formas de ligação foram desenvolvidas, eliminando problemas apresentados pelo modelo ponto-a-ponto. Dentre estes modelos tem-se o modelo de ligação multi-ponto ou de barramento.

3.3.2 Ligação multi-ponto ou barramento

Uma ligação multi-ponto, comumente denominada barramento, é um canal de comunicação que se conecta com todos os processadores e com os demais módulos presentes do sistema. Desta forma, não ocorre mais uma ligação direta entre os processadores, sendo necessário à passagem pelo barramento, para obter alguma informação desejada. Com a aplicação deste conceito, é facilitada a inclusão de novos módulos de *hardware* na arquitetura, como módulos de memória e dispositivos de entrada e saída. A Figura 1.1 [Garcia, 2008] apresenta um modelo de arquitetura com dois processadores conectados a uma memória compartilhada e um DMA, sendo a comunicação entre esses componentes, realizada por um barramento.

A fácil inserção de novos componentes em uma arquitetura que utiliza barramento torna o sistema amplamente reutilizável [Junior, 2010]. O baixo custo envolvido no desenvolvimento de sistemas com barramento, uma vez que o reaproveitamento é possível, além do fato de protocolos de controle de fluxo estarem bem desenvolvidos [Junior, 2010], faz com que, este modelo de comunicação, seja o mais utilizado em SoCs atualmente [Kumar et al., 2003]. Protocolos de controle, normalmente denominados árbitros do barramento, são necessários para garantir que todos os processadores conectados ao barramento possam utilizá-lo da melhor forma possível.

A Figura 3.11 [Ost, 2004] demonstra um exemplo de arquitetura multiprocessada com utilização de barramento gerenciado por um árbitro centralizado. Neste modelo de gerenciamento do barramento, o árbitro controla toda a movimentação de dados no barramento. Este executa uma solicitação por vez e empilha, em uma fila de requisições, as demais solicitações para que estas possam ser executadas posteriormente. Este modelo de comunicação, entre módulos de *hardware* e processadores, com um árbitro centralizado controlando a movimentação de dados, esta presente na atual arquitetura do VIPRO-MP.

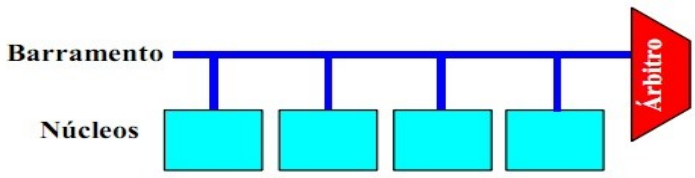

Figura 3.11: Arquitetura multiprocessada e barramento com árbitro centralizado [Ost, 2004].

A utilização de um sistema de comunicação com barramento e árbitro centralizado, apresenta muitas vantagens quando comparado a uma arquitetura com ligação ponto-a-ponto. Além da reusabilidade e baixo custo já citados, cada processador necessita apenas uma ligação, ou seja, um canal, para poder se comunicar com todo o sistema. Além disso, o sistema se torna escalável, até certo ponto, podendo inserir mais processadores sem precisar alterar a estrutura da arquitetura. Somando a isto, a complexidade em buscar rotas, que nas arquiteturas ponto-a-ponto eram exercidas pelo processador, fica agora a cargo do árbitro do

barramento. Isso contribui para tornar este meio de comunicação simples, porém eficiente para ser utilizado em arquiteturas multiprocessadas.

Apesar dos benefícios da utilização do sistema de barramento com árbitro centralizado, a partir de um determinado número de processadores ligados a este barramento, a escalabilidade acaba se tornando um problema [Kumar et al., 2003]. Com o número elevado de processadores e com a atuação do árbitro do barramento de forma sequencial, o tempo que um processador fica esperando que uma requisição seja atendida é elevado [Garcia et al., 2010]. Desta forma, o processador fica muito tempo ocioso. Além disso, o maior tamanho dos fios que fazem a ligação entre os componentes, promove uma elevação no consumo de potência [Ost, 2004] que acaba por prejudicar o desempenho final da arquitetura.

Desta forma, algumas formas de lidar com o acesso e manipulação sequencial das requisições feitas ao árbitro do barramento foram propostas. Dentre estas propostas estão uma estruturação do barramento em hierarquias [IBM, 1999] e a utilização de múltiplos barramentos [Chen e Sheu, 1991].

O conceito de múltiplos barramentos pode ser visto na Figura 3.12 [Junior, 2010]. Este modelo consiste em utilizar vários barramentos em uma mesma arquitetura. A forma como os processadores irão se conectar a estes barramentos depende da arquitetura implementada [Junior, 2010]. Esta variação pode fazer com que cada processador tenha um canal com cada barramento, Figura 3.12 (a), possibilitando que um número variado de transações aconteça simultaneamente. Também existe a possibilidade de conectar apenas alguns processadores a um determinado barramento, Figura 3.12 (b), possibilitando que uma aplicação seja executada independentemente das demais numa mesma arquitetura. Estas estruturações possibilitam aumentar a capacidade de comunicação e aumentam a largura de banda da arquitetura [Junior, 2010]. Como não existe apenas um árbitro de barramento, nas sim um conjunto deles, possibilita que mais de uma requisição seja atendida em um determinado intervalo de tempo.

As arquiteturas que utilizam um sistema hierárquico de barramento, também utilizam múltiplos barramentos, porém inserem a possibilidade de uma comunicação entre estes barramentos [Junior, 2010]. Neste modelo de barramento, não existe um árbitro [Ost, 2004], mas um circuito ponte (*bridge*), responsável por realizar a comunicação entre dois ou mais barramentos [Junior, 2010]. Desta forma, um processador se conecta unicamente a um

barramento, possibilitando uma estruturação adequada, por parte do desenvolvedor do sistema, de acordo com as funcionalidades que cada processador deverá desempenhar [Junior, 2010].

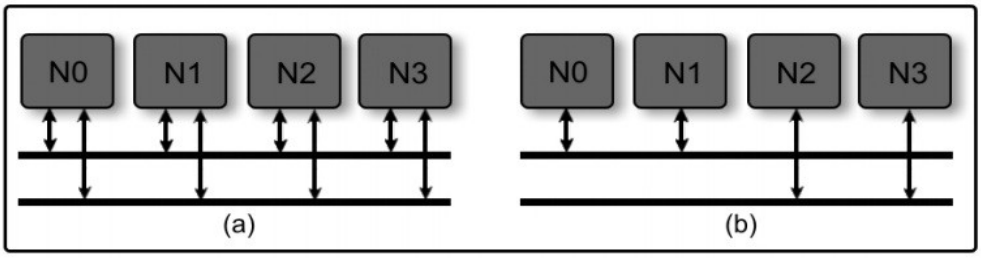

Figura 3.12: Arquitetura de comunicação com múltiplos barramentos: núcleos completamente conectados (a) e núcleos parcialmente conectados (b) [Junior, 2010].

Uma estruturação é possível e necessária quando a frequência dos processadores presentes na arquitetura é variada. Desta forma, é possível juntar em um barramento de alta velocidade os processadores de altas frequência e em outro barramento, que apresente menor velocidade, os processadores de baixas frequências [Junior, 2010]. Aplicando este conceito é possível criar certo grau de paralelismo na arquitetura. Para isso os processadores podem executar suas atividades em diferentes barramentos, sendo possível que todos compartilhem as mesmas informações, pois todo o barramento está conectado.

Na Figura 3.13 [Junior, 2010] é representado um exemplo de arquitetura com um sistema hierárquico de barramento. Apesar das vantagens este modelo também apresenta alguns problemas de desempenho. A grande perda de desempenho ocorre quando é necessário realizar a movimentação de informações entre os barramentos, pois, nesta situação ambos os barramentos ficam bloqueados até o término da requisição [Junior, 2010]. Caso ocorra uma diferenciação entre frequências de processadores e velocidades de barramentos, esta situação torna-se ainda mais grave, devido ao fato da necessidade de uma forma de sincronização entre barramentos e processadores. Além disso, uma transação entre subunidades do sistema de barramentos ocupa diversos recursos do sistema e exige *hardware* adicional, elevando o custo da arquitetura [Zeferino e Susin, 2003].

Desta forma, até mesmo as arquiteturas com hierarquia de barramento podem apresentar desvantagens quando utilizadas em arquiteturas multiprocessadas. Na tentativa de resolver os problemas apresentados pela utilização de barramentos nessas arquiteturas, novos conceitos de estruturas de comunicação passaram a ser desenvolvidas. Esses conceitos se referem à utilização de redes semelhantes às redes utilizadas na Internet, sendo denominadas NoCs ou redes *intra-chip* [Kumar et al., 2003].

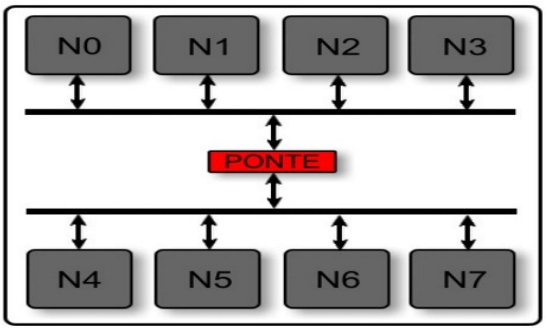

Figura 3.13: Arquitetura de comunicação com hierarquia de barramentos [Junior, 2010].

3.3.3 Redes NoC

Devido ao elevado número de canais e à falta de reusabilidade e escalabilidade em sistemas de comunicação ponto-a-ponto e escalabilidade reduzida em sistemas de comunicação com barramento, novos conceitos precisaram ser estudados e adaptados a um sistema SoC. Assim, da mesma forma que o paralelismo proporcionou o desenvolvimento dos ES multiprocessados, a necessidade da realização de atividades paralelas, incentiva o desenvolvimento de novos meios eficientes de comunicação [Junior, 2010]. Dentro desta necessidade, sendo baseados nos avanços tecnológicos presentes nas redes que formam a Internet [Kumar et al., 2003], como também em redes de intercomunicação de multiprocessadores [Junior, 2010], novos meios de comunicação *intra-chip* passam a ser desenvolvidos.

O advento da utilização de redes interligadas a computadores de uso geral, para simulação de atividades complexas [Junior, 2010], proporcionou uma evolução muito rápida das tecnologias de comunicação. São utilizadas, tanto em redes locais como em WANs (*Wide*

Area Network – rede de longa distância) [Patterson e Hennessy, 1996], como em redes de processamento paralelo [Ost, 2004], onde executam atividades específicas. Dentre os serviços suportados, que tornam essas redes de comunicação eficientes [Ost, 2004] [Junior, 2010] [Rijpkema et al., 2003], podem ser citados:

a) desempenho paralelo;
b) garantia de recebimento;
c) dados recebidos em ordem;
d) tolerância a falhas;
e) redução de custos;
f) largura de banda.

Como forma para garantir que todas essas características sejam atendidas, a utilização de roteadores e protocolos de controle e movimentação tornou-se fundamental [Rijpkema et al., 2003] [Patterson e Hennessy, 1996]. Os benefícios apresentados com a utilização de redes, baseadas em protocolos e roteadores, fizeram surgir um novo conceito na área de ES referente aos meios de comunicação entre processadores e módulos de *hardware.* Estas estruturas passam a ser conhecidas como NoCs (*Network-on-Chip* – redes em *chip* ou redes i*ntra-chip*), e emergem como solução para os problemas de intercomunicação presentes em arquiteturas multiprocessadas [Rijpkema et al., 2003].

O fato deste modelo de comunicação ser referenciado como solução deve-se a características como:

a) maior redução e eficiente nos gasto de energia [Rijpkema et al., 2003] [Benini e De Micheli, 2001];
b) apresenta alta escalabilidade [Ost, 2004] [Rijpkema et al., 2003];
c) linhas de comunicação compartilhadas [Rijpkema et al., 2003], fato que reduz o número total de linhas e possibilita uma melhor utilização das mesmas;
d) confiabilidade [Ost, 2004], possibilitada pela atuação de protocolos;
e) redução de custos [Benini e De Micheli, 2001];
f) elevado desempenho proveniente de uma estrutura com menor latência e maior vazão de dados [Benini e De Micheli, 2001].

Para garantir que estas características sejam atendidas, as NoCs devem prover serviços como os previstos por Rijpkema et al. [Rijpkema et al., 2003], sendo este:

a) integridade dos dados, evitando que os dados sejam corrompidos no processo de transmissão de uma fonte a um receptor;
b) entrega dos dados sem perdas, ou seja, deve ser evitado que os dados sejam perdidos no processo de transmissão;
c) dados entregues em ordem, ou seja, como existe mais de um caminho possível, é necessários garantias de os dados cheguem ao destino na mesma ordem que estão dispostos na fonte;
d) taxa de transferência e latência, relacionados com o tempo gasto para uma movimentação de dados.

Com a elevação dos requisitos necessários as redes computacionais de propósito geral, estas apresentam propostas que visam satisfazer toda esta complexidade, propostas estas, que podem ser repassadas a redes *intra-chip*, porém com certas diferenças [Rijpkema et al., 2003]. A primeira diferença se relaciona com a quantidade de recursos manipulados e a complexidade [Rijpkema et al., 2003] envolvida na tomada de decisões sobre caminhos, por parte dos roteadores, muito menores em NoCs se comparado a estruturas de propósito geral. Um SoC geralmente é desenvolvido para uma atividade específica, a NoC desenvolvida pode ser mapeada para um padrão de comunicação, otimizando caminhos críticos na troca de mensagens durante a execução [Junior, 2010].

Uma segunda diferença referencia o tamanho dos canais de comunicação utilizados [Ost, 2004] [Rijpkema et al., 2003]. Como a estrutura de redes é inserida num único *chip,* as distâncias entre os módulos e processadores é reduzida, se comparado a uma rede de propósito geral, desta forma é possível uma melhor sincronização entre os roteadores utilizados na NoC [Rijpkema et al., 2003]. Com a redução das distâncias, deve ser garantida uma redução na latência na arquitetura, reduzindo assim o número de ciclos necessários para que uma dada requisição seja atendida, reduzindo, desta forma, o consumo de energia do sistema [Junior, 2010]. Esta redução da distância entre os componentes e os roteadores, aliada a um número menor de componentes presentes em um SoC, se comparado a uma rede de

propósito geral, faz com que, protocolos específicos sejam desenvolvidos [Ost, 2004]. Este fato contribui para um aumento na complexidade da utilização destas arquiteturas.

Uma NoC pode ser caracterizada de acordo com:

a) serviço prestado [Ost, 2004], referente a alocação de recursos durante a execução;
b) protocolos, ou seja, sistemas de comunicação [Ost, 2004] [Junior, 2010];
c) pelas topologias que apresentam [Ost, 2004] [Junior, 2010].

Os serviços podem ser classificados como serviços garantidos e serviços de melhor esforço [Rijpkema et al., 2003]. Serviços garantidos requerem a reserva de recursos para situações críticas, como garantia na taxa de transferência [Ost, 2004], enquanto os serviços de melhor esforço não alocam recursos específicos, mas com isso não garantem taxa de transferência e latência [Ost, 2004].

Os protocolos definem como será realizada a transferência de dados pela rede [Junior, 2010]. Da mesma forma que nos protocolos de comunicação de redes de propósito geral, a comunicação, em rede *intra-chip*, é realizada através de mensagens [Benini e De Micheli, 2002]. Uma mensagem é composta por três partes, sendo elas, cabeçalho, carga útil e sufixo [Ost, 2004]. O cabeçalho contém informações gerais a respeito da origem, tamanho, podendo ou não conter dados referentes a métodos de segurança presentes na mensagem. A carga útil contém os dados em si, e o sufixo é utilizado para indicar o término da mensagem.

Por apresentar tamanhos variados, uma mensagem pode ser dividida em pacotes [Ost, 2004]. Cada pacote apresenta uma estrutura semelhante à estrutura da mensagem [Ost, 2004], porém a sua carga útil representa apenas uma fração da mensagem original. Desta forma o cabeçalho de um pacote deverá apresentar métodos e/ou informações para garantir que o receptor da mensagem possa extrair os dados de diversos pacotes e montar a mensagem na ordem correta, obtendo a mensagem original.

Um protocolo é utilizado para permitir a troca de mensagens entre todos os componentes da rede [Ost, 2004]. Desta forma, é necessário que todos os componentes que formam o sistema estejam interligados da melhor forma possível. Esta interligação, em uma estrutura *intra-chip*, é realizada através de um conjunto de roteadores e canais de comunicação, os fios de um sistema [Ost, 2004]. O conjunto formado pelos roteadores e os canais de comunicação geram uma topologia de NoC [Ost, 2004], e variando a forma como é realizada a ligação entre

roteadores e demais módulos é possível obter diferentes topologias de rede. De acordo com a topologia adotada, as redes de interconexão podem ser classificadas em duas categorias, redes diretas, ou estáticas, e redes indiretas ou dinâmicas [Junior, 2010].

O conjunto formado por canais de comunicação e pelos roteadores tem influência direta sobre a estrutura final apresentada por um SoC. Quando uma NoC é desenvolvida, requisitos como diâmetro da rede, conectividade, largura de banda e latência devem ser considerados, pois estes fatores influenciarão o desempenho final do sistema [Kumar et al., 2003]. O diâmetro da rede indicará o tamanho que os canais de comunicação apresentarão [Kumar et al., 2003] [Junior, 2010]. A conectividade aborda quantas ligações é possível realizar em um roteador [Kumar et al., 2003]. A largura de banda indica qual será a movimentação de *bits* por segundo suportada pela rede [Kumar et al., 2003] e a latência é o tempo necessário para uma mensagem ir da origem ao destino [Kumar et al., 2003]. A escolha de fatores como número de roteadores e quantas ligações cada um irá fazer tem ligação direta com o custo final do SoC, uma vez que quanto mais componentes, mais elevado fica o valor do produto final.

Nas topologias de redes ditas diretas, cada processador é ligado, através de uma ligação ponto-a-ponto, com um roteador, e os roteadores podem apresentar um número variado de ligações com outros roteadores [Junior, 2010]. O canal utilizado para ligar dois roteadores pode ser monodirecional, possibilitando fluxo de dados apenas em um sentido, ou bidirecional, apresentando movimentação de dados em ambos os sentidos, ou seja, receber e enviar dados pelo mesmo canal de comunicação [Junior, 2010]. Comparado a uma estrutura de barramento, a escalabilidade desta organização ocorre do fato da inserção de um novo nodo, conjunto formado pelo roteador e processador [Junior, 2010]. Isto ocorre, pois, aumenta a largura de banda total do sistema, uma vez que adiciona mais canais, ao contrário do sistema de barramento que divide os canais já existentes [Junior, 2010].

As topologias de redes diretas ou estáticas recebem este nome devido, as ligações serem realizadas uma vez e não poderem ser refeitas, tendo assim uma conexão dedicada [Ost, 2004]. Essa topologia apresenta ainda uma divisão, dependendo do diâmetro que apresenta, ou seja, da distância entre os roteadores [Junior, 2010]. Caso a distância é uniforme, as redes diretas são denominadas redes regulares, e irregulares, quando essa distância é variada. Na

Figura 3.14 [Junior, 2010] é apresentado alguns modelos de redes de interconexão diretas, onde os blocos representam um nodo da rede.

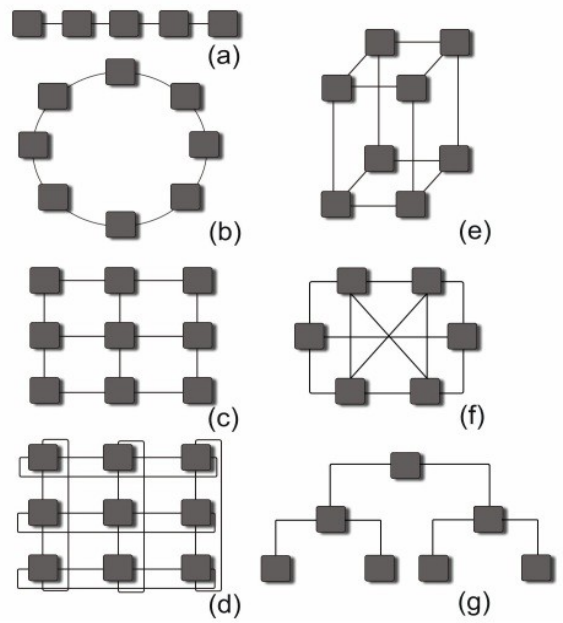

Figura 3.14: Topologias de redes de interconexão diretas: Rede Linear (a), Rede Anel (b), Rede Grelha (c), Rede Toróide (d) Rede Cubo de Grau 3 (e), Rede Completamente Conectada (f) e Rede em Árvore (g) [Junior, 2010].

A rede linear Figura 3.14 (a) é o modelo mais simples, porém, implica em dificuldades quando se deseja buscar informações entre os extremos da arquitetura. Este problema é solucionado pela Rede Anel, Figura 3.14 (b), uma vez que a distância entre todos os nodos é constante. Na Rede Grelha, Figura 3.14 (c), cada nodo se conecta aos nodos vizinhos gerando uma distância maior entre as extremidades e forçando um número maior de ligações nos nodos centrais, o que acarreta maior complexidade na hora da escolha de qual caminho adotar. Para eliminar o problema da distância entre as extremidades da rede grelha, foi desenvolvida a Rede Toróide, Figura 3.14 (d), que conecta os nodos extremos diretamente. Uma Rede Hipercubo apresentam um nodo interligando *n* outros dependendo do grau da rede [Junior, 2010], no exemplo da Figura 3.14 (e), o grau da rede é três, desta forma é obtido uma Rede Hipercubo de grau três, onde cada nodo se conecta a outros três nodos [Junior, 2010]. A rede apresentada na Figura 3.14 (f) é um exemplo de Rede Totalmente Conexa, ou *full-*

connect, desta forma é possível fazer uma ligação direta entre todos os nodos, como uma rede ponto-a-ponto, porém esta forma apresenta a mesma desvantagem do modelo ponto-a-ponto, ou seja, o custo elevado para ser desenvolvida. A rede da Figura 3.14 (g) é uma rede em forma de árvore binária, apresentando os mesmos conceitos envolvidos na definição das mesmas.

Nas redes indiretas ou dinâmicas, os roteadores passam a ser representados por chaves denominados *switches* ou *crossbar switches* [Junior, 2010]. Estes elementos são responsáveis pelo fornecimento do caminho para ligar dois nodos distintos [Junior, 2010], ou seja, as chaves possuem mecanismos que a possibilitam uma mesma chave realizar ligação a mais de um nodo. Desta forma as ligações são dinâmicas, mudando de acordo com o caminho desejado, ao contrário das redes estáticas onde as ligações são fixas. As topologias indiretas podem ser divididas em três categorias [Junior, 2010], sendo elas *Crossbar*, Estágio Único e Multiestágio.

Um exemplo de rede *Crossbar* é apresentado na Figura 3.15 [Junior, 2010]. Neste modelo existe uma conexão entre todas as entradas com todas as saídas, sendo um modelo de comunicação rápida [Junior, 2010]. Para garantir isto, existe um *switch* para cada ligação entre dois módulos, elevando o custo e reduzindo a escalabilidade do sistema, pois para cada novo módulo anexado, um novo *switch* deve ser incluído e ligado a cada processador. Por exemplo, na Figura 3.15 existem quatro processadores e quatro módulos de memória, cada processador apresenta um *switch* com cada módulo de memória totalizando 16 *switch* [Junior, 2010]. Caso um novo processador ou módulo de memória fosse incluído, seriam necessários mais 4 *switch*.

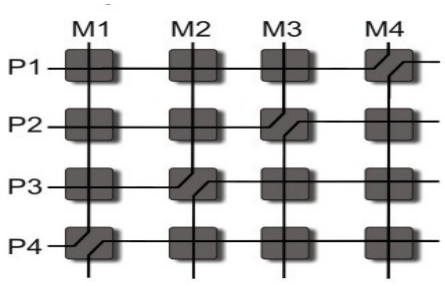

Figura 3.15: Rede *Crossbar* 4x4 [Junior, 2010].

Em redes de Estágio Único, existe uma entrada em cada *switch,* e o dado fica circulando pela rede até encontrar um caminho que ligue a origem ao destino [Junior, 2010]. Nas redes multiestágio, como a apresentada na Figura 3.16 [Junior, 2010], existem várias entradas e várias saídas, ficando a cargo de um roteador escolher o caminho a ser tomado. Desta forma é possível que uma ligação entre origem e destino apresente caminhos diferentes.

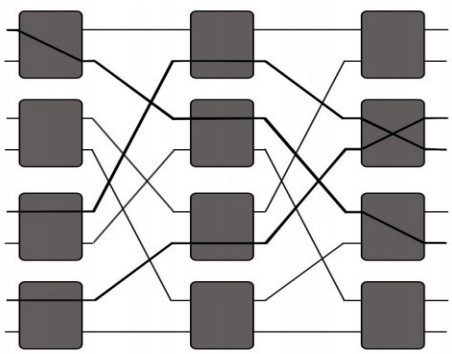

Figura 3.16: Rede Multiestágio [Junior, 2010].

Para a adoção de um modelo de rede *intra-chip*, o desenvolvedor do sistema deve levar em consideração fatores como:

a) relação custo/benefício [Ost, 2004];
b) número de núcleos e módulos de memória que se pretende utilizar na rede [Ost, 2004];
c) quantas portas de entrada e de saída um roteador irá apresentar [Ost, 2004], pois isto influência, tanto na complexidade, como no custo do dispositivo;
d) escalabilidade e confiabilidade [Ost, 2004];
e) a aplicação a qual a rede deverá executar [Ost, 2004];
f) a complexidade envolvida para a adoção de protocolos, para obter a melhor forma de movimentação dos dados;
g) reusabilidade;
h) escolha na forma de transferência de pacotes [Ost, 2004] [Rijpkema et al., 2003] e mecanismos de controle de fluxo [Junior, 2010];

i) características do canal a ser adotado como largura, comprimento, latência e frequência suportadas [Junior, 2010];
j) escolha de roteadores e algoritmos de roteamento adequados para o objetivo proposto [Junior, 2010].

Desta forma, os principais fatores envolvidos na escolha de uma rede *intra-chip* está no conhecimento das mesmas e qual a funcionalidade que o sistema irá desempenhar. Dentre alguns dos modelos de redes NoCs utilizados atualmente podem ser citados as redes indiretas em FPGAs (*Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo) [Kilts, 2007]. As Redes de árvores são utilizadas no modelo SPIN (*Scalable Programmable Interconnection Network*) [Adriahantenaina, 2003], com alterações que possibilitam que todos os processadores se conectem aos nodos folhas [Junior, 2010]. A rede Octagon [Karim et al., 2002] apresenta a topologia anel com oito nodos, porém, cada nodo apresenta três canais de comunicação. A rede Spidergon [Coppola et al., 2004] é uma evolução da rede Octagon, apresentando desenvolvimento comercial, sendo uma rede regular escalável e com topologia direta [Junior, 2010], ou seja, com ligação ponto-a-ponto entre os nodos. A rede SoCIN (SoC *Interconnection Network*) [Zeferino e Susin, 2003] apresenta a topologia grelha, possui algoritmos para evitar conflitos na movimentação de dados e é de fácil implementação [Junior, 2010], mesmas características da rede Nostrum [Millberg et al., 2004].

A tecnologia de intercomunicação evoluiu consideravelmente nos últimos anos, possibilitando o desenvolvimento de redes *intra-chip* com elevado desempenho. A adoção de um modelo de NoC em um SoC possibilita alto grau de paralelismo, rapidez na resolução de uma atividade, aumento de desempenho e redução de parâmetros como tamanho da arquitetura e tempo de projeto, uma vez que esta solução pode ser altamente escalável. Porém, o desenvolvedor da arquitetura deve ter conhecimento de qual a melhor topologia a ser adotada, uma vez que uma escolha equivocada pode resultar em perda de desempenho e elevação do custo do sistema. Um conhecimento das funcionalidades do sistema e das topologias de rede auxilia na tomada desta decisão.

Capítulo 4

Protótipo Virtual Utilizado: VIPRO-MP

O VIPRO-MP (*Virtual Prototype for Multiprocessor Architectures*) é uma plataforma virtual projetada para a simulação de arquiteturas multiprocessadas [Garcia, 2008]. A utilização, neste trabalho, do VIPRO-MP deve-se ao fato de ser um ambiente livre de simulação, com código aberto, desenvolvido em meio acadêmico. Este simulador possibilita a simulação de ambientes multiprocessados, e a inclusão de novos módulos de *hardware* é facilitada. Além disso, é possível alterar o tamanho das caches, dados de configuração dos processadores e latência da memória. Somando a isto, é possível realizar estimativas do consumo de potência, número de ciclos gastos, potência dissipada por ciclo, além de outros dados que fornecem um completo conjunto de informações referentes à simulação de uma aplicação.

No VIPRO-MP, os dados não são armazenados em memória cache, devido a não adoção de protocolos de coerência de cache no simulador [Garcia et al., 2010]. Além disso, o sistema de intercomunicação adotado é um barramento centralizado. Este modelo apresenta um árbitro, responsável por controlar o acesso a memória, gerando assim uma forma serializada de acesso à memória compartilhada. Com isso, ocorre uma limitação do número de processadores que podem ser utilizados em uma arquitetura, devido a problemas de desempenho. Este modelo de utilização da memória, onde a memória privada utiliza a cache, mas a memória compartilhada não, é utilizado em plataformas embarcadas como o Nexperia [Goossens, 2005] ou mesmo o processador Cell [IBM, 2010]. Nas seções subsequentes serão discutidas as ferramentas utilizadas por esta plataforma virtual e as motivações da utilização de plataformas virtuais no desenvolvimento de Sistemas Embarcados.

4.1 Ferramentas Utilizadas Pelo Simulador

O VIPRO-MP é uma arquitetura que permite a execução de aplicações implementadas em linguagem C, e compiladas para a geração de código para o conjunto de instruções PISA (*Portable Instruction Set Architecture*), semelhante a arquitetura MIPS [Garcia, 2008]. A utilização da arquitetura MIPS ocorreu devido a grande utilização da mesma em sistemas embarcados atuais [Garcia, 2008]. A plataforma virtual foi desenvolvida para ambiente GNU/Linux. Os compiladores utilizados são o GCC 3.4 e o G++ 3.4 [Garcia, 2008]. Um Makefile é utilizado para atender os requisitos das ferramentas envolvidas e a execução do VIPRO-MP é baseada na manipulação de *threads* [Garcia, 2008].

O VIPRO-MP foi desenvolvido baseado na ferramenta SimpleScalar [Burger e Austin, 1997]. Esta ferramenta constitui um grupo de simuladores de processadores superescalares, compiladores e depuradores. Foi iniciado em 1997 na Universidade de Wisconsin-Madison, e apresenta suporte as arquiteturas PISA, ARM e X86. Dentre os oito simuladores do SimpleScalar, o sim-outorder é o adotado na plataforma VIPRO-MP. Isto se deve ao fato deste simulador, em relação aos outros, apresentar previsão de desvios, renomeação de registradores, execução fora de ordem (*out-of-order*, despacho de instruções fora da ordem que aparecem no código, possibilitando sobrepor as restrições de dependência) e apresenta a possibilidade de configuração dos seus parâmetros [Garcia, 2008]. Dentre eles estão o tamanho da fila de busca de instruções, o tamanho das caches de dados e instruções, a fila de *load/store* e a quantidade de unidades funcionais [Garcia, 2008].

O VIPRO-MP apresenta suporte a interrupções anexadas ao sim-outorder, possibilitando que módulos de *hardware,* como o DMA (*Direct Memory Access*), pudesse ser anexado à plataforma [Garcia et al., 2010], uma vez que este módulo necessita de sinais de interrupção para correto funcionamento. Em 2008 o SimpleScalar foi reescrito em SystemC, para incluir um suporte a simulação de arquiteturas multiprocessadas, uma vez que a limitação inicial deste simulador era o suportar de apenas arquiteturas monoprocessadas [Garcia, 2008]. SystemC é uma linguagem proposta para especificar sistemas de *hardware* e *software* desenvolvida em C/C++ [Garcia, 2008]. Desta forma, o SystemC é utilizado para modelar

módulos de *hardware* utilizados no VIPRO-MP, como memória compartilhada, barramento e DMA [Garcia et al., 2010].

A modelagem em alto nível de abstração, em C++, de módulos de *hardware* facilita a manipulação da plataforma pelo usuário [Garcia, 2008]. Como exemplo desta facilidade, tem-se a comunicação entre memória/barramento, barramento/processador, barramento/DMA. Esta comunicação é realizada por canais de transferência no nível de transações - TLM (*Transaction Level Messages*) modelados em SystemC [Garcia, 2008] [Garcia et al., 2010], e não em níveis de sinais. Ou seja, a troca de informações entre esses módulos podem ser simulados com chamadas de funções que operam de forma semelhando a sinais em *hardware.*

Para as metodologias tradicionais, após um módulo ser desenvolvido em C/C++, analisado, testado e refinado é modelado manualmente em HDL (*Hardware Description Language*), simulado e sintetizado, Como pode ser observado na Figura 4.1 (a) [Garcia, 2008]. Porém, este modelo pode gerar inconsistências e qualquer erro resulta em uma recriação total do módulo [Garcia, 2008]. O SystemC passou a criar uma metodologia própria de elaboração de projetos de ES, como apresentado na Figura 4.1 (b). Este fato é decorrente da utilizando de uma mesma linguagem para simular um sistema que possibilita um refinamento gradual, facilitando adaptações e detecção de erros e a utilização de um mesmo conjunto de testes ao longo de todo o desenvolvimento do sistema [Garcia, 2008].

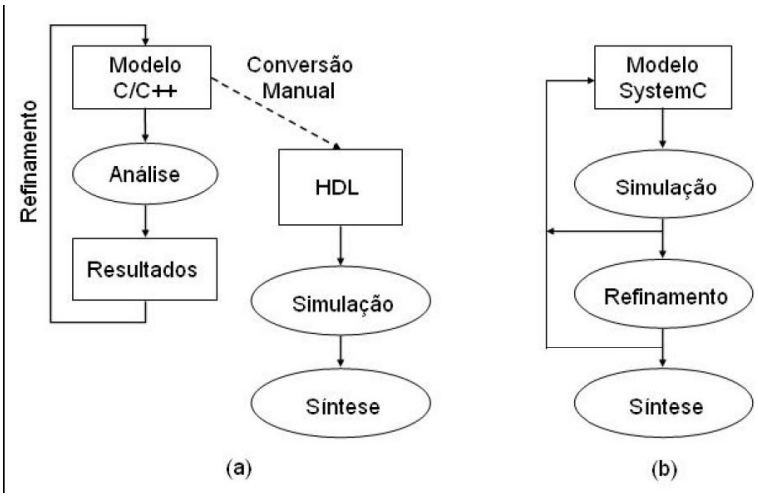

Figura 4.1: Duas metodologias de projeto. (a) metodologia tradicional; (b) metodologia SystemC [Garcia, 2008]

No desenvolvimento do VIPRO-MP, para o cálculo de potência, foi utilizado o *framework* Sim-Wattch [Brooks et al., 2000], um simulador desenvolvido pela Universidade de Princeton no ano de 2000, estendendo o simulador sim-outorder [Garcia, 2008]. O Sim-Wattch utiliza a biblioteca CACTI, que através de formulações, fornece uma estimativa de consumo de potência de alguns elementos de *hardware* [Garcia, 2008]. Dentre as medições realizadas pela CACTI estão a medida do consumo de potência dinâmica, energia consumida por ciclo, modelagem de área, análise de potência estática e consumo de potência de acesso a memórias como RAM (*Random Access Memory*) e SRAM (*Static Random Access Memory*) [Garcia, 2008]. Utilizando o Sim-Wattch é possível obter um conjunto de informações completas sobre o consumo de potência envolvida na execução de uma aplicação, proporcionando um estudo para validar se o desempenho da arquitetura é satisfatório.

Atualmente, o VIPRO-MP, apresenta uma divisão entre memórias, como pode ser observado na Figura 1.1, onde existe uma memória compartilhada, uma memória cache e uma memória interna em cada processador. Como mencionado, não ocorre uma utilização da cache pela memória compartilhada, apenas as memórias locais dos processadores simulam o acesso à cache. Mesmo nas memórias locais, a manipulação da cache é feita apenas como uma simulação de acesso. Desta forma, o primeiro passo a ser adotado é uma alteração na atual estrutura da cache para que esta possa ser utilizada e manipulada de forma coerente para que fosse possível a inclusão de protocolos de coerência de cache.

4.2 Porque da Utilização de Protótipos Virtuais no Auxilio de Desenvolvimento de ES

Por apresentar uma ampla variedade de componentes em um mesmo dispositivo, o projeto e desenvolvimento de um ES apresentam alta complexidade. Na tentativa de elaboração de um projeto com qualidade, uma especificação deve ser utilizada. Porém, não existe uma padronização para essa área, sendo a divisão em macro níveis, apresentados na Figura 4.2 [Wolf, 2001], uma das formas mais viáveis de dividir as fases de desenvolvimento de um Sistema Embarcado.

Os requisitos mencionados na Figura 4.2 fazem referência às características que devem ser priorizadas no desenvolvimento, por exemplo, desempenho e potência. A especificação é utilizada para representar as funcionalidades e objetivos do sistema em uma linguagem de alto nível, como a UML (*Unified Modeling Language*). Os componentes de *hardware* e *software* que serão utilizados são especificados e documentados na fase da arquitetura. Na fase dos componentes são desenvolvidos, separadamente, os módulos de *hardware* e *software* sendo a integração desses componentes a última parte do projeto.

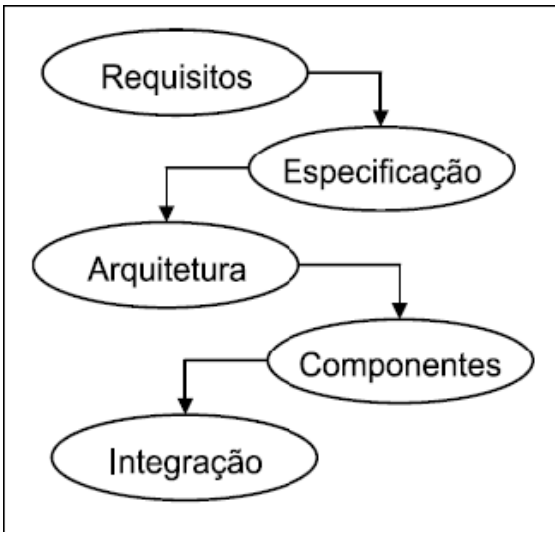

Figura 4.2: Níveis de abstração de um projeto de ES [Wolf, 2001].

Essa ordem sequencial, no entanto, pode gerar uma perda de tempo muito elevada, caso algum erro ocorra. Pois, se um erro ocorrer, todo o sistema fica na espera da solução deste problema, para poder continuar seu desenvolvimento. Na tentativa de eliminar esses atrasos, a utilização de plataformas virtuais passaram a ser utilizadas Com a utilização deste recurso, o projetista pode, através de simulações, encontrar potenciais erros e os corrigir, antes mesmo de dar início ao projeto físico do ES [Jerraya, 2004].

Com a simulação em plataformas virtuais, o projetista pode realizar testes sobre as mais variadas configurações possíveis para um Sistema Embarcado. Existe a possibilidade da escolha de barramentos, memórias, tamanhos de cache, número de processadores, entre outros componentes. A possibilidade da simulação faz com que uma configuração que melhor

execute a aplicação desejada seja encontrada, sem envolver custos adicionais de tempo e de recursos. Com isso, ocorre uma validação de *hardware* e *software* necessários a aplicação, de forma simultânea. Desta forma, o tempo do projeto é reduzido, evita-se que erros de projeto ocorram,

Com a simulação é possível escolher os melhores módulos de *hardware* e obter dados de consumo de energia, potência e desempenho, antes mesmo do ES ser finalizado [Garcia, 2008]. Atualmente, é possível ter uma economia de até 40% no tempo total de desenvolvimento de um ES com o uso de simulações [Zivojnovic, 2007]. E, analisando projetos de Sistemas Embarcados é possível afirmar que um terço do total dos projetos não atingem 50% da performance esperada nem das funcionalidades que o sistema deveria suportar e que mais de 13% do total de projetos são cancelados [Krasner, 2003]. Todos os benefícios que as simulações disponibilizam a um projeto de Sistema Embarcado fizeram com que as plataformas virtuais se tornassem uma técnica muito utilizada.

Capítulo 5

Modelagem e Implementação de Protocolos de Coerência de Cache no VIPRO-MP

Para possibilitar o uso da memória cache e de protocolos de coerência, foram necessárias alterações na estrutura do modelo de memória cache que o simulador VIPRO-MP apresentava. Após estas alterações, foi desenvolvido um modelo de protocolo de coerência de memória cache do tipo rastreamento e um modelo do tipo diretório. Executando uma aplicação JPEG paralela [Garcia, 2008] com o uso destes protocolos, foi realizada uma comparação com a execução do simulador sem a utilização da cache de dados. Os módulos implementados e os resultados obtidos estão descritos nas seções subseqüentes.

5.1 Reestruturação na Memória Cache

A versão original de memória cache do VIPRO-MP realiza apenas simulações de acesso a esta memória. Assim era apenas calculada a latência no acesso a um determinado dado, em qualquer nível de cache. Não ocorre uma real manipulação sobre os dados na memória cache.

Para que a cache pudesse ser utilizada, foram desenvolvidos e inseridos métodos para a real manipulação dos dados. Ao invés dos dados serem manipulados na memória interna dos processadores, esses valores passam a ser inseridos na cache, desviando o fluxo de movimentação de dados do simulador, elevando o desempenho obtido. Desviando esse fluxo de movimentação, a memória interna dos processadores, que era utilizada anteriormente desaparece, e a manipulação de dados passa a ocorrer na memória cache.

Foram adotados os seguintes critérios na configuração da memória cache:

a) mapeamento direto de dados da memória compartilhada;
b) política de substituição de paginas FIFO e;
c) política de atualização *write back*.

Em um sistema multiprocessado, é necessário que a cache tenha dados coerentes. Para isso, uma variável para controlar o estado de coerência de um bloco presente na cache foi inserida nesta memória. O protocolo utilizado para garantir a coerência nos dados é o protocolo MESI. Uma manipulação sobre um endereço sempre é seguido da atualização de seu estado de coerência, garantindo que o sistema permaneça coerente.

5.2 Protocolo de Coerência de Cache – Modelo Rastreamento

O primeiro protocolo de coerência implementado foi o modelo rastreamento. Neste modelo os módulos implementados foram:

a) Controlador da Memória cache;
b) Snoop;
c) Barramento das caches ou Barramento local.

Na Figura 5.1, os componentes em cinza, representam os módulos citados acima, implementados e inseridos no simulador. Nas subseções seguintes é apresentado cada um desses módulos.

5.2.1 Controlador da memória cache

Após as alterações na cache, foi desenvolvido e inserido no simulador um módulo de *hardware* responsável por controlar as atividades relacionadas à utilização da memória cache. Este dispositivo recebe solicitações do processador para manipulação de dados, solicita a um módulo responsável a busca de dados em outras memórias cache, busca dados na memória compartilhada e controla diretamente a movimentação e manipulação de dados na cache. Este dispositivo foi inserido diretamente no simulador SimpleScalar. Isto ocorreu devido à facilidade de acesso aos demais componentes do sistema, registradores, ao barramento e a

memória cache. O fluxo de execução do controlador da cache, no momento de uma requisição realizada pelo processador é demonstrado na Figura 5.2.

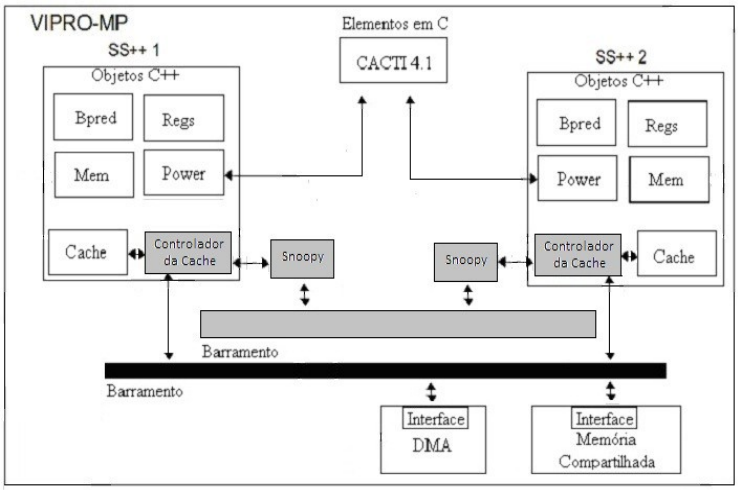

Figura 5.1: VIPRO-MP com módulos do protocolo de coerência rastreamento.

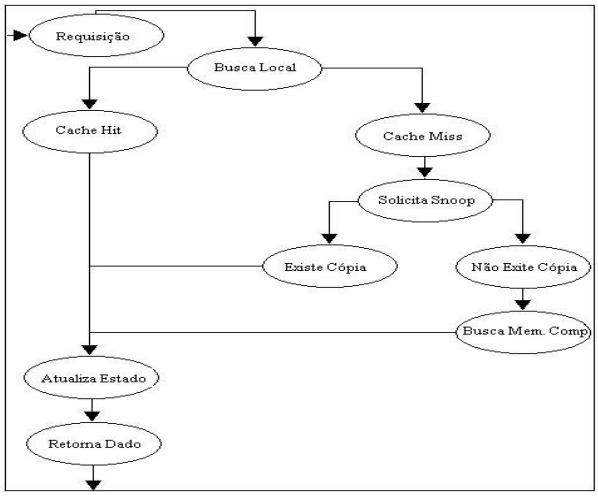

Figura 5.2: Fluxo de execução do controlador da cache.

No momento em que um dado é requisitado pelo processador, um pedido é feito ao controlador da cache. Neste momento o processador fica em estado de espera até o momento em que esta requisição seja atendida e finalizada. O controlador realiza uma busca pelo

endereço solicitado na memória cache local. Caso o endereço esteja na cache e seja válido, a operação solicitada é realizada, com a atualização dos dados conforme o protocolo MESI. Caso contrário, o controlador da cache tentará encontrar uma cópia válida deste endereço nas cache dos demais processadores via requisição ao Snoop. Se for encontrado um dado válido, este é retornado ao controlador que atualiza a cache e retorna o dado ao processador. Caso não exista alguma cópia válida, o controlador irá buscar o dado com endereço solicitado na memória compartilhada. Depois de inserido na cache, este dado é atualizado e retornado ao processador.

O controlador também recebe solicitações de verificação se a cache local possui um endereço válido vinda de outros controladores. Em uma situação como esta, o controlador da cache, interrompe a sua execução atual, verifica se existe uma cópia válida do endereço solicitado na cache local, e retorna o dado do endereço solicitado ou uma notificação da não existência do endereço na cache local. Assim, todo acesso a cache deve ser realizada via controlador da cache.

5.2.2 Rastreador de movimentação de dados - Snoop

Este módulo foi denominado Snoop, sendo sua função ligar o barramento local de cache aos controladores de cache. Quando solicitado, o Snoop envia, via barramento, uma solicitação a todos os demais Snoops, presentes na arquitetura, na tentativa de encontrar uma cópia válida do dado solicitado. Quando uma solicitação de busca local é realizada a um Snoop, este interrompe a execução do controlador da cache ao qual está ligado, solicita o dado, espera o controlador atualizar o dado na cache e retorná-lo. Após ter recebido o dado do controlador o Snoop que apresenta o dado válido, fornece, via barramento, o dado ao Snoop que gerou a requisição.

Toda movimentação de dados entre processadores, obrigatoriamente, deve passar por Snoops. Um Snoop apresenta portas TLM de conexão com o barramento local e com o seu respectivo controlador de cache. Este sistema de comunicação é idêntico ao praticado pelo processador com o barramento do sistema. Assim, independente da quantidade de processadores presentes na arquitetura, o Snoop apresenta uma estrutura que não precisa ser alterada.

5.2.3 Barramento local

O barramento local foi desenvolvido baseado no barramento utilizado pelo VIPRO-MP para comunicação entre processadores e memória compartilhada. É utilizado especificamente para a movimentação de dados entre as caches dos processadores presentes na arquitetura. Utiliza portas TLM, e não apresenta um árbitro centralizador. Para garantir a integridade dos dados, quando uma solicitação de movimentação é solicitada, o barramento local fica bloqueado e apenas o Snoop do processador que está realizando a movimentação de dados pode utilizá-lo. Este bloqueio evita que atrasos de leitura ou escrita gerem dados inconsistentes. Quando a movimentação de dados é finalizada, o barramento é liberado, e qualquer Snoop pode utilizá-lo novamente.

5.3 Protocolo de Coerência de Cache – Modelo Diretório

Neste modelo, um único módulo apresenta informações de quais dados estão presentes nas caches e em quais processadores elas se encontram. Para o desenvolvimento desse modelo de coerência de cache os seguintes módulos foram implementados:

a) Controlador da cache e;
b) Diretório.

Na Figura 5.3, os componentes em cinza, representam os módulos citados acima, implementados e inseridos no simulador . Nas subseções seguintes é apresentado cada um desses módulos.

5.3.1 Controlador da memória cache

O controlador da cache utilizado para o modelo diretório é o mesmo modelo utilizado no modelo rastreamento, com o mesmo funcionamento. Toda vez que um dado é necessário ao processador, uma requisição é gerada ao controlador da cache. Caso este dado esteja presente na memória cache local e esteja com estado válido, este dado é fornecido ao processador e atualizado no diretório, caso necessário. Em caso de cache miss, é realizada uma busca no diretório. Se o dado não estiver em outra cache do sistema, este dado é buscado na memória compartilhada, inserido na cache local, atualizado no diretório e retornado ao processador.

Não realizando alterações na estrutura do controlador da cache e na cache, quando uma comparação for realizada, a diferença no resultados ocorre devido aos demais componentes que fazem parte da estrutura, no caso deste trabalho, o Snoop e o Diretório. Da mesma forma a complexidade em cada projeto é definida de acordo com a inclusão desses novos componentes.

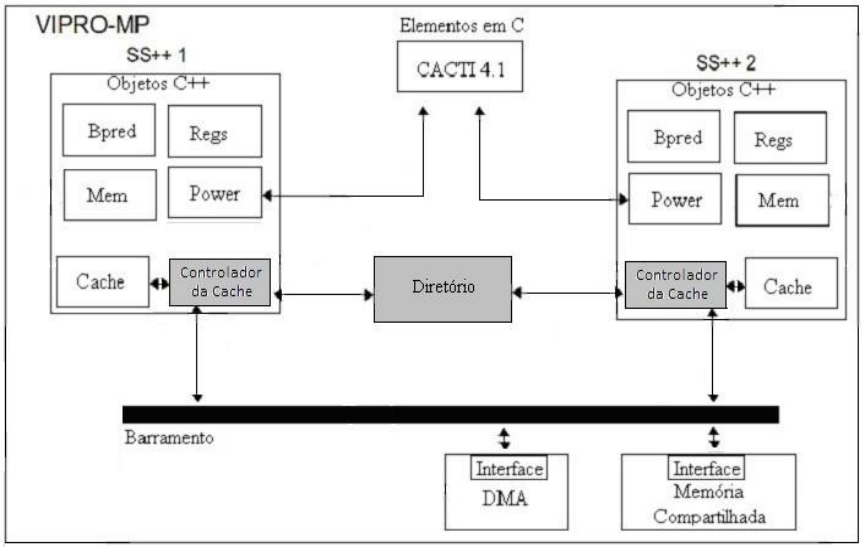

Figura 5.3: VIPRO-MP com módulos do protocolo de coerência diretório.

5.3.2 Diretório

O diretório é responsável por armazenar informações dos endereços da cache que estão mapeados nos diversos processadores do sistema. Para o desenvolvimento do diretório foi criada uma tabela *hash*, onde cada campo é uma estrutura que apresenta o dado, o estado atual deste dado, segundo o protocolo de coerência MESI e em qual processador esta localizado.

A tabela *hash* auxilia em uma operação de busca, fornecendo uma maior rapidez para se encontrar o dado solicitado. Esta é dividida em cinco conjuntos com os dados sendo inseridos em cada conjunto, seguindo a regra adotada. Esta regra aplica uma divisão do endereço do dado por cinco, sendo o resto inteiro desta divisão o conjunto no qual o dado será inserido. Caso um dado tenha seu estado alterado para inválido, este dado é eliminado do diretório,

realizando um *flush* na cache em que se apresente, nos demais casos, este dado é apenas atualizado.

Este módulo apresenta uma estrutura responsável por armazenar as requisições dos processadores aos quais está conectado. Uma *thread* fica responsável pela realização destas solicitações. Quando uma solicitação é finalizada uma nova passa a ser atendida, retornando ao controlador da cache que realizou a solicitação finalizada:

a) o dado necessário, ou;

b) a indicação da não existência deste dado em outra cache ou;

c) um sinal de atualização da informação referente a um determinado dado.

No caso de fornecer um dado a um controlador de cache, o diretório envia uma solicitação de cópia do dado ao controlador que apresenta o dado válido em sua cache, espera que este dado seja retornado e fornece o dado ao controlador solicitante, alterando a informações do estado do dado caso necessário em sua tabela.

Assim como o Snoop, o diretório se comunica com os controladores de caches através de canais TLM, sendo uma ligação para cada controlador. Esta funcionalidade se assemelha ao barramento que conecta o controlador a memória compartilhada. Porém, não apresenta um árbitro centralizador, sendo as requisições atendidas de acordo com ordem de chegada. Este fato reduz a escalabilidade do sistema a partir de um número de processadores utilizado, sendo um gargalo no sistema. Quanto mais processadores, maior o tempo para que uma requisição seja atendida, elevando o tempo ocioso do processador e sobrecarregando o diretório. Apesar dessa limitação, este modelo é vantajoso para arquiteturas que apresentam poucos processadores, devido a sua complexidade de implementação reduzida.

5.4 Resultados das Simulações

Para validar o sistema foi realizada uma simulação com um algoritmo JPEG paralelo [Garcia, 2008]. Como pode ser observado na Figura 5.4 [Garcia, 2008], esse algoritmo é dividido em duas partes, na primeira parte cada processador é responsável por realizar a transformada DCT (*Discrete Cosine Transform*) de N/P blocos JPEG de uma imagem e reescrever os resultados na memória compartilhada (onde N é o número total de blocos desta

imagem e P é o número total de processadores presentes na simulação) [Garcia, 2008]. Quando todos os processadores terminarem de executar esta atividade, um processador *master* passa a realizar a parte final do algoritmo. Esta parte final é a compressão JPEG (*entropy encode*), sendo realizada de forma sequencial e gerando o arquivo de saída [Garcia, 2008].

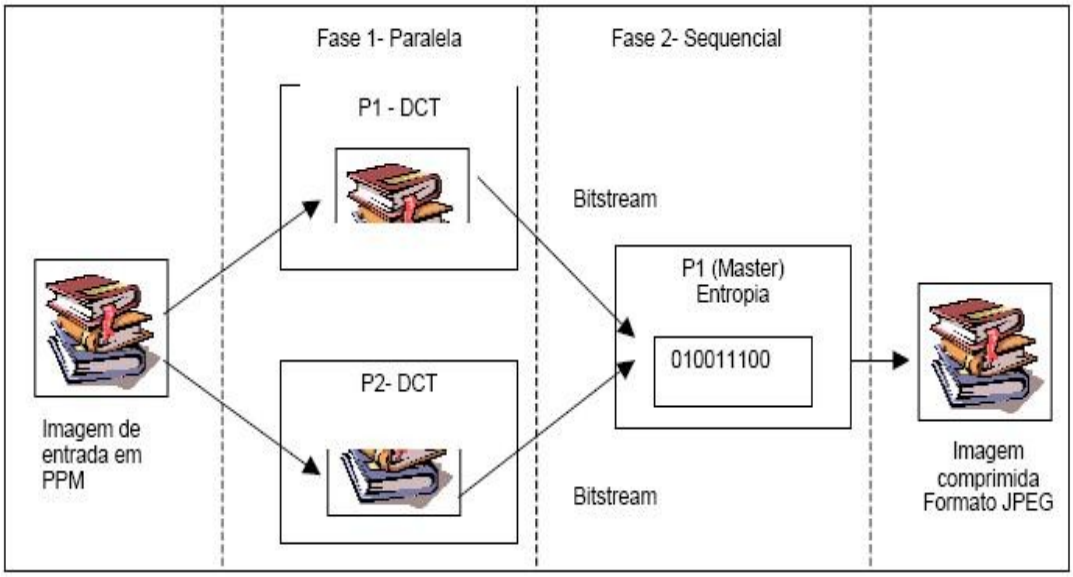

Figura 5.4: Algoritmo de compressão JPEG paralelo [Garcia, 2008].

Na primeira simulação, não é utilizada a cache dos dados presentes na memória compartilhada. Desta forma, toda vez que um dado é requerido pelo processador uma busca deste dado é realizada na memória compartilhada, sendo em seguida fornecido para manipulação na memória local do processador. Com isso ocorrem, de forma constante, acessos a uma memória com maior latência. A latência é o termo utilizado para indicar qual memória é mais rápida em uma comparação, quanto maior a latência, mais lenta é a memória. Neste âmbito a memória compartilhada apresenta latência de 18 ciclos, ou seja, são necessários 18 ciclos de execução para que dado possa ser fornecido, o que seria obtido pela memória cache em apenas 1 ciclo.

Na segunda simulação, ocorre a inserção dos dados presentes na memória compartilhada na cache. Com isso o acesso a memória compartilhada é reduzida, diminuindo o tempo

necessário na obtenção de um determinado dado, quando ocorre um cache *hit*. Para ambas as simulações é utilizado uma imagem de 256x256 *pixels*. Este tamanho é utilizado para facilitar a divisão dos blocos JPEG entre os processadores na primeira fase do algoritmo JPEG paralelo. As arquiteturas utilizadas apresentam um, dois, quatro e oito processadores.

Com relação a frequência utilizada pelo processador, deve ser levado em conta a possibilidade da execução de atividades paralelas e a vazão de processamento oferecida [Garcia, 2008]. Isso indica que um processador de 2 Ghz pode apresentar desempenho, teoricamente, similar que quatro de 500 Mhz, uma vez que esse conjunto possibilita a execução de mais atividades em paralelo que apenas um núcleo. A utilização de mais processadores com frequências menores esta sendo utilizada por processadores de propósito geral atuais como os Core 2, da Intel e os Athlon X2 da AMD [Garcia, 2008]. Segundo GIVARGIS [Givargis et al., 2002] o consumo de potência em circuitos integrados CMOS (*complementary metal-oxide-semiconductor*), pode ser modelado pela equação 5.1:

$$P = \frac{C \, x \, A \, x \, F \, x \, V^2}{2} \tag{5.1}$$

onde, C, é a capacitância média de chaveamento, A, é um termo de 0.0 a 1.0 que determina a atividade média de chaveamento, F, é a frequência de relógio aplicada no circuito e V, é a tensão aplicada para que ocorra o chaveamento das portas lógicas. Com isso e analisando a Figura 5.5 [Simunic et al., 2001], percebe-se que ocorre uma redução da voltagem com a redução da frequência, tendo impacto direto sobre a potência gerada. A redução de voltagem contribui positivamente para uma arquitetura pois implica em menos gasto de energia. Porém a redução de frequência implica numa redução de velocidade de execução do processador, o que gera uma elevação no tempo para a finalização de uma atividade, reduzindo o desempenho desta arquitetura. Assim a voltagem e a frequência apresentam uma relação aproximadamente linear. Porém, como apresentado na fórmula 5.1, a voltagem tem uma relação quadrática em relação a potência.

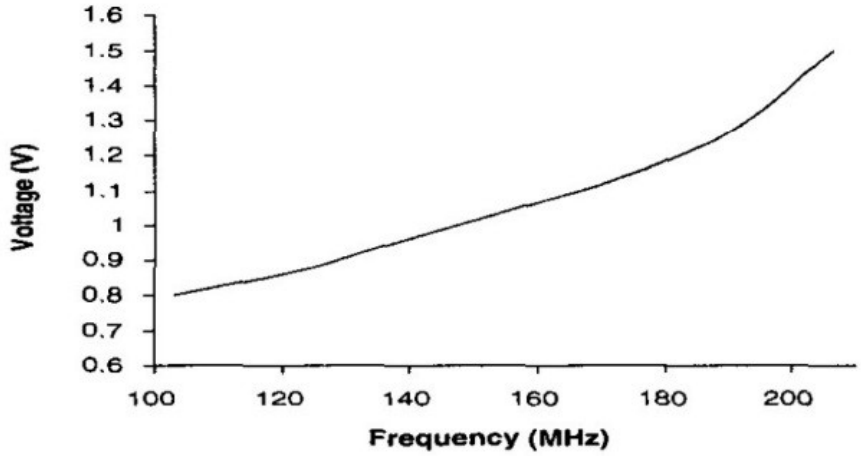

Figura 5.5: Frequência vs Voltagem em um StrongARM SA-110 [Simunic et al., 2001].

Analisando a fórmula 5.2 tem-se que a energia consumida é fornecida pela potência dissipada em um determinado tempo. Assim, reduzindo a potência, poderá ocorrer uma redução da energia gasta. Porém, para este fato se confirmar é necessário que o tempo gasto não sofra uma alteração considerável, sendo este tempo relacionado com a frequência utilizada pelos processadores. Assim, energia, potência e frequência se relacionam para gerar o desempenho final de uma arquitetura.

$$Energia = Potência \times Tempo \tag{5.2}$$

Com relação a tamanho da memória cache, tem-se que quanto maior a quantia de dados manipulados, maior deverá ser a capacidade de armazenamento da memória cache, evitando que ocorra elevado incidência de *cache miss*. Desta forma, caso ocorra um aumento do número de processadores, o montante total de dados manipulados por cada processador será reduzido, com isso a cache também pode apresentar uma redução, sem elevar a ocorrência de *cache miss*. Considerando todos esses fatores, e analisando a Tabela 5.1 [Garcia, 2008], foi adotada as configurações de frequência, tamanho de cache de dados e cache de instruções apresentadas na Tabela 5.2. A diminuição das frequências dos núcleos e do tamanho das caches buscou que, a soma das partes sempre resultasse no valor total utilizado para a arquitetura com apenas um processador.

Tabela 5.1: Dados de configuração e execução de ambientes com um, dois e quatro núcleos [Garcia, 2008].

Configuração		1 Núcleo	2 Núcleos		4 Núcleos			
			#1	#2	#1	#2	#3	#4
Frequência	MHz	1200	600	600	300	300	300	300
Escala de Voltagem	-	1	0,5	0,5	0,25	0,25	0,25	0,25
Cache de dados	Kb	512	256	256	128	128	128	128
Cache de Instruções	Kb	256	128	128	64	64	64	64
Execução								
Número de Ciclos	-	16.130.586	10.903.904	7.605.377	10.303.912	4.891.482	5.987.316	6.989.351
Contenção do Barramento	ciclos	0	558.098	591.943	1.081.213	558.547	1.654.657	2.659.138
Potência Média por Ciclo	watt	611,27	53,3414	70,3985	6,0182	8,4541	6,9062	5,9139
Tempo de Execução	Seg.	0,0134	0,0182		0,0343			
Energia	w*t	8,1910	2,2520		0,9361			

Tabela 5.2: Frequência dos processadores e tamanho das memórias caches utilizadas nas simulações.

	1 Núcleo	2 Núcleos	4 Núcleos	8 Núcleos
Frequência (MHz)	1200	600	300	150
Cache de dados (Kb)	4096	2048	1024	512
Cache de instruções (Kb)	8192	4096	2048	1024

A seguir são apresentados os resultados das simulações conforme indicado anteriormente, utilizando primeiramente o protocolo de coerência rastreamento e em seguida o modelo diretório.

5.4.1 Coerência de cache com modelo rastreamento - Snoop

Como o desempenho da aplicação é relacionado com o número de ciclos gastos, na Tabela 5.3 são apresentados o total de ciclos gastos para realizar uma transferência de um dado. Quando da ocorrência de um *cache hit* é necessário apenas um ciclo de execução para fornecer o dado ao processador. Para uma situação de cache, onde tem-se a verificação da existência de cópias válidas e/ou o fornecimento dessa cópia, é necessário um número maior de ciclos de execução, devido a operações de busca, inserção, atualização e invalidação ou atualização. Os valores fornecidos, são teóricos, pois apenas com a implementação física é possível obter com precisão esses valores.

A elevação do número de ciclos pela quantidade de processadores é devido ao número de elementos e a distância entre eles, este fato acaba por fazer que a movimentação de informações e dados consuma mais ciclos de execução. A distância mencionada se refere a distribuição física dos componentes no *hardware* a ser desenvolvido. Essa medida deve ser calculada de forma a garantir um correto funcionamento de toda a arquitetura.

Numa ocorrência de *cache miss* e acesso a memória compartilhada, deve ser somado ao total de ciclos a latência da memória compartilhada. No modelo simulada a latência dessa memória é de 18 ciclos. Toda vez que um processador solicitar um dado ao controlador da cache, um certo número de ciclos de execução irá ser gasto para atender essa requisição, de acordo com as informações da Tabela 5.3. Os resultados das simulações do algoritmo JPEG paralelo sem cache dos dados da memória compartilhada e com cache dos dados utilizando o protocolo de coerência modelo rastreamento estão presentes na Tabela 5.4, que apresenta o número total de ciclos de execução gastos para finalizar a aplicação.

Tabela 5.3: Ciclos de execução do protocolo de coerência Rastreamento.

	Ciclos			
	1 Núcleo	**2 Núcleos**	**4 Núcleos**	**8 Núcleos**
Cache *Hit*	1	1	1	1
Cache *miss* (verificar existência de cópias válidas)	4	6	8	10
Cache *miss* (fornecer cópia válida)	4	6	8	10
Cache *miss* (acessar memória compartilhada)	22 (4+18)	24 (6+18)	26 (8+18)	28 (10+18)

Tabela 5.4: Ciclos gastos para execução do algoritmo JPEG paralelo, sem cache da memória compartilhada e com protocolo de coerência Rastreamento.

	Ciclos			
	1 Núcleo	**2 Núcleos**	**4 Núcleos**	**8 Núcleos**
Sem cache da memória compartilhada	11244275	6971236	4819811	4056665
Com cache da memória compartilhada (Solução Proposta)	5220750	2986698	1853810	1250071
Redução (%)	53,57	57,16	61,54	69,18

O número de ciclos utilizado para realizar esta comparação se refere ao montante de ciclos gastos pelo processador *master* para finalizar a compressão da imagem. Isto se deve ao fato deste processador finalizar sua execução por último. Analisando o total de ciclos gastos é possível perceber uma redução que varia de 53,57% para arquiteturas com um processador, 57,16% para arquiteturas com dois processadores, 61,54% para quatro processadores presentes na arquitetura sendo a redução de 69,18% para arquiteturas com oito processadores. Considerando que, em cada arquitetura utilizada, o gasto de energia por ciclo sofre uma redução quadrática devido a diminuição, pela metade, da frequência de cada processador, esta redução de ciclos irá implicar em uma redução, considerável, do consumo de energia da arquitetura. Além disso, cada ciclo corresponde a um tempo gasto para execução, com isso o tempo de finalização da aplicação também é reduzido. O conjunto formado por todos esses dados comprovam os benefícios que a utilização da memória cache pode prover em um ES.

5.4.2 Coerência de cache com modelo diretório

O total de ciclos gastos para realizar uma transferência de um dado é apresentado na Tabela 5.5. Quando ocorre um *cache hit* é necessário apenas um ciclos de execução para fornecer o dado solicitado ao processador, devido a latência da memória cache. Na ocorrência de um *cache miss* são necessários três ciclos de execução, número teórico, devido as operações de busca, inserção e atualização/remoção do diretório. Como existe apenas um elemento centralizador, mesmo elevando o número de processadores, o total de ciclos necessários não é alterado. Caso ocorra um acesso a memória compartilhada, tem-se um acréscimo de 18 ciclos no total, referente a latência dessa memória. . Os resultados das simulações do algoritmo JPEG paralelo sem cache dos dados da memória compartilhada e com cache dos dados utilizando o protocolo de coerência modelo diretório estão presentes na Tabela 5.6.

Da mesma forma que os dados apresentados na Tabela 5.4, o número de ciclos apresentados na Tabela 5.6 se refere ao total de ciclos gastos pelo processador *master* para finalizar a compressão JPEG da imagem. Analisando o total de ciclos tem-se uma redução de 74,64% para a arquitetura com um processador, 71,65% para a arquitetura com dois processadores, com quatro processadores obteve-se uma redução de 62,17% e com oito

processadores 33,43% de redução do número de ciclos foi obtido. Da mesma forma que nas simulações com o modelo Snoop apresentados na tabela 5.4, para garantir a coerência nos dados da cache, esta redução implica em redução da energia gasta para a execução da aplicação e do tempo necessário a finalização da mesma.

Tabela 5.5: Ciclos de execução do protocolo de coerência Diretório.

	Ciclos			
	1 Núcleo	**2 Núcleos**	**4 Núcleos**	**8 Núcleos**
Cache *Hit*	1	1	1	1
Cache *miss* (verificar existência de cópias válidas)	3	3	3	3
Cache *miss* (fornecer cópia válida)	3	3	3	3
Cache *miss* (acessar memória compartilhada)	21 (3+18)	21 (3+18)	21 (3+18)	21 (3+18)

Tabela 5.6: Ciclos gastos para execução do algoritmo JPEG paralelo, sem cache da memória compartilhada e com protocolo de coerência Diretório.

	Ciclos			
	1 Núcleo	**2 Núcleos**	**4 Núcleos**	**8 Núcleos**
Sem cache da memória compartilhada	11244275	6971236	4819811	4056665
Com cache da memória compartilhada (Solução Proposta)	2851670	1976219	1823571	2700555
Redução (%)	74,64	71,65	62,17	33,43

5.4.3 Comparativo entre os modelos rastreamento e diretório

Um comparativo entre o total de ciclos gastos para executar a aplicação JPEG paralela pode ser observado no gráfico apresentado na Figura 5.6. Analisando o traçado gerado, é possível obter uma comparação entre os modelos de coerência rastreamento e diretório.

Devido ao fato do modelo diretório apresentar uma única estrutura que centraliza todas as operações, quando ocorre um aumento do número de processadores, o tempo necessário para que uma requisição seja atendida também é elevado. Esse tempo de espera eleva o tempo que

o processador fica ocioso a espera do retorno do dado, eleva o número de ciclos gastos. Com isso, mesmo que o número de ciclos por instrução seja menor que no modelo rastreamento, o fato do diretório centralizar todas as operações acaba elevando o montante total de ciclos para poder atender todas as solicitações, quando se eleva o número de núcleos presentes na arquitetura. No modelo rastreamento, o barramento que centraliza as operações de busca só fica bloqueado quando ocorre uma transferência de dados. Uma operações de busca não gera o bloqueio do mesmo, sendo assim, varias operações de busca podem ocorrer em paralelo. Desta forma o modelo rastreamento apresenta um desempenho constante mesmo elevando o número de processadores. Por outro lado o modelo diretório perde desempenho quando ocorre um aumento do número de processadores.

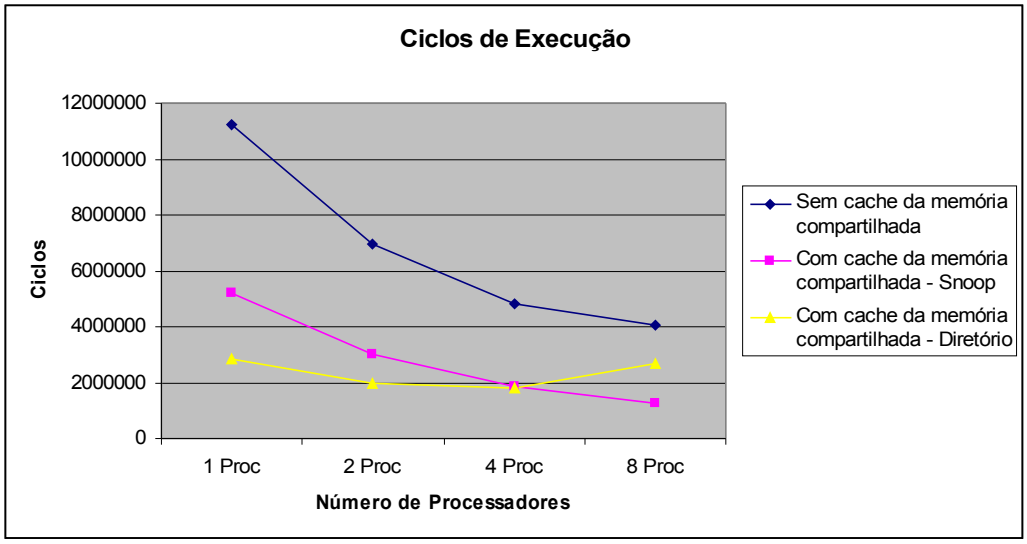

Figura 5.6: Ciclos de execução para a aplicação JPEG paralela.

Para um arquitetura com um processador o modelo diretório obteve um desempenho superior de 21,07%, em relação ao modelo rastreamento, devido ao número de componentes acessados. No modelo rastreamento, toda requisição do processador envolve acesso controlador da cache e, caso ocorra cache *miss*, ao Snoop e ao barramento, mesmo que a arquitetura apresente apenas um processador, pois as atividades dos módulos são projetadas para suportar arquiteturas com *n* processadores. Desta forma esses acessos contribuem para

elevar o tempo em que uma requisição é finalizada, elevando o montante final de ciclos de execução. Para a arquitetura monoprocessada que apresenta o modelo diretório, ocorre acessos apenas ao diretório caso ocorra um cache *miss*. Isso proporciona uma resposta ao processador em menos ciclos de execução necessários.

Na a Figura 5.7, são apresentados os porcentuais de redução do número de ciclos gastos para executar a aplicação JPEG paralela. Com os dados obtidos, tem-se que o modelo rastreamento apresenta uma redução de ciclos constante mesmo elevando o número de processadores presentes na arquitetura. Por outro lado, o modelo diretório apresenta uma redução menor, conforme o número de processadores aumente na arquitetura.

Para as arquiteturas com um e dois processadores o modelo diretório apresentou uma redução do número de ciclos mais elevada que o modelo rastreamento. No entanto, este comportamento não se mantém e a partir de arquiteturas com quatro processadores o desempenho do modelo rastreamento é superior. Assim, para arquiteturas com um e dois processadores,o modelo diretório se mostrou mais eficiente. Para arquiteturas com quatro e oito processadores, o modelo rastreamento apresentou um desempenho mais elevado. Estes fatos comprovam que a forma de como ocorre o acesso aos dados deve ser considerado em um projeto de ES, evitando que uma escolha resulte em perda de desempenho.

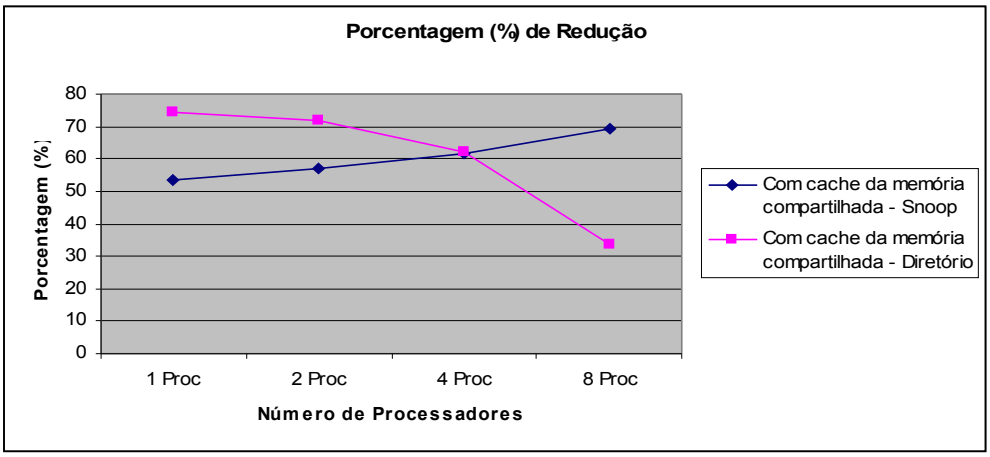

Figura 5.7: Porcentagem de redução para a aplicação JPEG paralela.

Capítulo 6

Conclusões e Trabalhos Futuros

ES apresentam constante evolução nas tecnologias adotadas na sua elaboração. Geralmente destinados ao mercado consumidor, um Sistema Embarcado deve ser elaborado em curto espaço de tempo, com elevado desempenho, custos reduzidos e evitando a ocorrência de erros de projeto. Buscando evitar potenciais erros e projeto, protótipos virtuais passa a ser ferramentas indispensáveis em projetos de ES.

Através da simulação é possível realizar testes e obter resultados precisos das mais variadas configurações de *hardware* a ser adotada em Sistemas Embarcados. Modelando os componentes de *hardware* através de *software* é possível estimar com maior margem de segurança as melhores configurações através da realização de testes que comprovam a eficiência ou não de determinado módulo em relação a um similar em determinada arquitetura.

O VIPRO-MP possibilita uma rápida exploração de diferentes arquiteturas, além da facilidade de inclusão de novos módulos. O fato de todos os componentes do simulador serem modelados em uma mesma linguagem, possibilita que novos módulos de *hardware* possam ser criados, anexados e simulados nesse ambiente de simulação.

Através dos módulos de coerência de cache implementados e anexados ao simulador foi possível analisar todo o impacto positivo, em termos de desempenho, que o uso da memória cache pode prover a uma arquitetura. Apesar da complexidade envolvida, o uso da cache pode prover um ganho de desempenho de até 74,64% em uma aplicação, como observado para a simulação do algoritmo JPEG paralelo e modelo de coerência de cache diretório.

Apesar de ambos os modelos de coerência de cache garantirem o uso correto da memória cache, ocorre diferenças de desempenho, de 21,07%, de um modelo para o outro, devido a forma que ocorre o acesso aos dados das memórias cache. Desta forma, através de simulações, foi possível analisar qual modelo é mais adequado para uma determinada arquitetura. Caso esses dados não fossem obtidos através da simulação, ambos os modelos necessitariam ser projetados, sintetizados e testados, elevando, assim, o tempo e custo de um projeto envolvido.

Em termos de complexidade, o modelo rastreamento é o mais complexo e, portanto, o mais trabalhoso para ser desenvolvido e testado, devido ao fato de apresentar mais elementos envolvidos e a possibilidade de suportar atividades paralelas. Em termos de desempenho, o modelo diretório se mostrou mais eficiente para arquiteturas com um e dois processadores, decrescendo seu desempenho com o aumento do número de processadores envolvidos. Sendo o modelo rastreamento mais eficiente para arquiteturas com quatro e oito processadores. Este fato evidencia que o diretório pode vir a se tornar um gargalo do sistema, da mesma forma que o barramento com árbitro centralizador é atualmente para o VIPRO-MP [Garcia et al, 2010]. Este fato ocorre devido a configuração do simulador que permite apenas a inclusão de apenas um nodo de diretório. Desta forma uma alteração na estrutura de comunicação do simulador, com a inclusão das redes NoC, torna possível, como trabalhos futuros, a inclusão de um sistema de diretórios distribuídos. Com isso o real desempenho de um diretório em sistemas muitiprocessados pode ser testado e validado.

Como continuidade para este trabalho, melhorias no simulador, com a inclusão de uma rede de inter-conexão NoC, devem ser realizadas. Este conceito, com o estudo apresentado neste trabalho, fornecerão uma maior variedade de componentes para simulação e aumentarão o escopo de arquitetura que podem ser simuladas pelo VIPRO-MP. Assim, quanto mais ampla a variedade de componentes presentes em um simulador, maior será o espaço de projeto que esta plataforma virtual pode fornecer.

Referências Bibliográficas

[Adriahantenaina, 2003] ADRIAHANTENAINA, A. et al. SPIN: a Scalable, Packet Switched, on-Chip Micro-Network. In: *Conference on Design, Automation and Test*, 2003. Proceedings of the Conference on Design, Automation and Test, Munich, pag. 70-73.

[ARM, 2010] ARM. Disponível em: http://www.arm.com. Acessado em: 13/jul/2010.

[Baskett et al., 1988] BASKETT F., JERMOLUK, T. A., SOLOMOM, D. The 4D-MP graphics superworkstation: Computing + graphics = 40 MIPS + 40 MFLOPS and 100,000 lighted polygons per second. *COMPCON*, pag 468–471, 1988.

[Benini e De Micheli, 2001] BENINI. L.; DE MICHELI. G. Powering Networks on Chip. In: *International Symposium on System Synthesis*, 2001. Proceedings of the International Symposium on System Synthesis, [s.n.], pag. 33–38.

[Benini e De Micheli, 2002] BENINI, L.; DE MICHELI,G. Networks on Chips: a New SoC paradigm. *Computer*, vol. 35, no 1, pag. 70-78, Janeiro, 2002.

[Brooks et al., 2000] BROOKS, D.; TIWARI, V.; MARTONOSI, M. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In: *International Symposium on Computer Architecture, ISCA*, 2000. Proceedings of International Symposium on Computer Architecture, Vancouver, B.C, pag. 83-94.

[Burger e Austin, 1997] BURGER, D.; AUSTIN, T. M. *The SimpleScalar Tool Set: Version 2.0*. Madison: University of Wisconsin, 1997. Relatório técnico no.1342.

[Chen e Sheu, 1991] CHEN, W.T.; SHEU, J.P. Performance Analysis of Multiple Bus Inerconnection Networks with Hierarchical Requesting Model. *IEEE Transactions on Computers*, vol. 40, no. 7, pag. 834- 842, 1991.

[Coppola et al., 2004] COPPOLA, M. et al. Spidergon: a Novel on-Chip Communication Network. In: *International Symposium on System-on-Chip*, 2004. Proceedings of the International Symposium on System-on-Chip, Tampere, pag. 15.

[Covacevice et al., 2007] COVACEVICE, A. V. T., BALDOCHI, B. Y., CASTRO, F. R. *Coerência de Cache*. Universidade Estadual de Campinas, Campinas, SP. Julho 2007.

[Dettmer, 1990] DETTMER, R.; Bright Sparc [RISC-based microprocessor], *IEE Review*, vol.36, no.9, pag.331-335, Outubro, 1990.

[Fisher et al., 2005] FISHER, J. A., FARABOSCHI, P., YOUNG, C. *Embedded Computing: A Vliw Approach to Architecture, Compilers and Tools*. 1st Edition. New York: Morgan Kaufmann, 2005.

[Garcia, 2008] GACIA, M. S. *VIPRO-MP: uma plataforma virtual multiprocessada baseada na arquitetura SimpleScalar*. Monografia – Universidade Estadual do Oeste do Paraná, Cascavel, Pr, Julho 2008. Monografia.

[Garcia e Oyamada, 2008] GARCIA, M. S. ; OYAMADA, M. S. . Desenvolvimento de uma plataforma virtual multiprocessada baseada na arquitetura SimpleScalar. In: *ix wscad - Simpósio em Sistemas Computacionais de Alto Desempenho,* 2008. Poceedings of the ix wscad, Campo Grande.

[Garcia et al., 2010] GARCIA, M. S.; SCHUCK, M.; OYAMADA, M. S. Utilizando protótipos virtuais para avaliação do impacto do DMA no desempenho de sistemas MPSoC. In: *16th Iberchip Workshop - Red Iberoamericana de Servicios de Fabricación de Microsistemas para Soporte a la Industria y Formación Continua de Expertos en Microelectrónica*, 2010. Proceedings of the 16th Iberchip Workshop. Foz do Iguaçu, pag. 1-6.

[Gaudin, 2010] GAUDIN, S. Intel´s transistors redesign in new era of technology. Disponivel em: http://www.computerworld.com/s/article/9046458/Intel_s_transistor_redesign_ushers_in_new_era_of_technology?taxonomyId=162&pageNumber=1. Acessado em 13/jul./2010.

[Gelsinger et al., 2000] GELSINGER, P.P.; GARGINI, P.A.; PARKER, G.H.; YU, A.Y.C. Microprocessors circa 2000. *Spectrum, IEEE.* Vol.26, no.10, pag.43-47, Outubro, 1989. Doi: 10.1109/6.40684

[Givargis et al., 2002] GIVARGIS, T; VAHID, F. PLATUNE: A Tuning Framework for System-on-a-Chip Platforms. *IEEE Tranactions on Computer – Aided Design of integrated circuits and systems*, Vol. 21, No. 11, pag. 1317-1327, 2002.

[Goossens, 2005] GOOSSENS, K., et al. Dynamic and Robust Streaming in and Between Connected Consumer- Electronics Devices, capítulo 2: Service-Based Design of Systems on Chip and Networks on Chip. 37 - 60, Springer, v.3, 2005.

[Hennessy et al., 1982] HENNESSY. J., JOUPPI. N., PRZYBYLSKI. S., ROWEN. C., GROSS. T., BASKETT. F., GILL. J. MIPS: A microprocessor architecture. *SIGMICRO Newsl.* Vol. 13, pag. 17-22, Dezembro, 1982.

[IBM, 1999] IBM. The CoreConnect Bus Architecture. White Paper, 1999. 8 pag.

[IBM, 2010] IBM. IBM Cell Broadband Engine Architecture, Disponível em: www.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA. Acessado em: 13/jul/2010.

[Intel, 2010a] Intel Corporation. Disponível em: http://www.intel.com. Acessado em: 13/jul/2010.

[Intel, 2010b] Intel Core 2 Duo. Disponível em: http://www.intel.com/products/processor/core2duo/index.htm. Acessado em: 14/jul/2010.

[Jerraya, 2004] JERRAYA, A. Long Term Trends for Embedded System Design, in: EUROMICRO Symposium on Digital System Design, 2004. Proceedings of IEEE Press, Washington, pag. 20-26.

[Junior, 2007] JUNIOR, A. A. A., GARIBOTTI, R. F. *Memória Cache em uma Plataforma Multiprocessada (MPSoC)*. Monografia - Pontificia Universidade Católica, Porto Alegre, Rs, março 2007. Monografia.

[Junior, 2009] JUNIOR, J. T., *Projeto e Implementação de Multiprocessador Embarcado em Dispositivos Lógicos Programáveis*. Dissertação (Dissertação de Mestrado) - Universidade Federal do Paraná, Curitiba, Pr, Agosto 2009. Dissertação.

[Junior, 2010] JUNIOR, N. A. G. *Análise e Simulação de Topologias de Redes em Chip.* Dissertação (Dissertação de Mestrado) - Universidade Estadual de Maringa, Maringa, Pr.Março 2010. Dissertação.

[Karim et al., 2002] KARIM, F.; NGUYEN, A.; DEY, S. An Interconnect Architecture for Networking Systems on Chips. *IEEE Micro*, vol. 22, no. 5, pag. 36-45, 2002.

[Kilts, 2007] KILTS, S. Advanced FPGA Design: Architecture, Implementation, and Optimization. Hoboken, EUA: Wiley & Sons, 2007. 352 pag.

[Krasner, 2003] KRASNER, J. Embedded Software Development Issues and Challenges – Failure Is NOT An Option – It Comes Bundled With the Software. *Embedded Market Forecasters*. Julho, 2003.

[Kumar et al., 2003] KUMAR, Sm, JANTSCH, A., TENHUNEN, A. Networks on Chip, capítulo 5: On Packet Switched Network for Chip Communication, 85 - 160, Kluwer Academic Publishers, 2003.

[Lamboia, 2008] LAMBOIA, F. *Análise comparativa de uso dos conjuntos de instruções dos microprocessadores de 32bits MIPS, POWERPC e SPARC*. Dissertação (Dissertação de Mestrado) – Universidade Federal do Paraná, Curitiba , Pr, Agosto 2008. Dissertação.

[Levy et al., 1996] LEVY, H. M.; JACK L. L; EMER, J. S.; STAMM, R.L.; EGGERS, S.J.; TULLSEN, D.M. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In: *23rd Annual International Symposium on Computer Architecture,* 1996. Proceeedings of the 23rd Annual

International Symposium on Computer Architecture, [s.n.], 1996, pag. 191- 191. Doi: 10.1109/ISCA.1996.10020

[Maia, 1998] MAIA, L. P. *Multithread*. Monografia – Universidade Federal do Rio de Janeiro, Rio de Janeiro, Rj, Maio 1998. Monografia.

[Mammana et al., 2009] Back to School with Tablets Embedded in Digital Desks. Disponivel em: http://www.informationdisplay.org/issues/2009/09/art8/art8.htm. Acessado em 13/jul/10.

[Marsala e Kanawati, 1994] MARSALA, A.; KANAWATI, B. PowerPC processors, System Theory, In: *26th Southeastern Symposium*, 1994. Proceedings of the 26th Southeastern Symposium, [s.n.], 1994, pag. 550-556. Dói: 10.1109/SSST.1994.287816.

[Marwedel, 2006] MARWEDEL, P. *Embedded Systems Desing*. 1st Edition, Dordrecht, Netherlands: Springer, 2006.

[Millberg et al., 2004] MILLBERG, M. et al. The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip. In: *17th International Conference on VLSI Design*, 2004. Proceedings of the 17th International Conference on VLSI Design, Mumbai, pag. 693-696.

[MIPS, 2010] MIPS. Disponível em: http://www.mips.com. Acessado em: 13/jul/2010.

[Ost, 2004] OST, L. C. *Redes Intra-Chip Parametrizaveis com Interface Padrão para Sintese em Hardware*. Dissertação (Dissertação de Mestrado) – Pontifícia Universidade Católica, Porto Alegre, Rs, Março 2004. Dissertação.

[Papamarcos e Patel, 1984] PAPAMARCOS, M. S., PATEL, J. H. A low-overhead coherence solution for multiprocessors with private cache memories. In: *11th International Symposium on Computer Architecture*, 1984. ACM Press, pag 348–354.

[Patterson e Hennessy, 1996] PATTERSON, D. A., HENNESSY, J. L. *Computer architecture: a quantitative approach*, 2nd Edition. San Francisco: Morgan Kaufmann Publishers, 1996.

[Patterson e Hennessy, 2002] PATTERSON, D. A., HENNESSY, J. L. *Computer architecture: a quantitative approach*, 3rd Edition. San Francisco: Morgan Kaufmann Publishers, 2002.

[Patterson et al., 2007] PATTERSON, D. A., HENNESSY, J. L., ARPACI-DUSSEAU, A. C. *Computer architecture: a quantitative approach*, 4th Edition. San Francisco: Morgan Kaufmann Publishers, 2007.

[Pizzol, 2002] PIZZOL, G. D. *SimMan: Simulation Manager. Definição e Implementação de um Ambiente de Simulação de Arquiteturas Superescalares para a Ferramenta SimpleScalar.* Monografia - Universidade Federal do Rio Grande do Sulm, Porto Alegre, Maio 2002. Monografia.

[Rijpkema et al., 2003] RIJPKEMA. E.; GOOSSENS. K.; RADULESCU. A.; DIELISSEN. J.; MEERBERGEN. J. V.; WIELAGE. P.; WATERLADER. E.Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. In: *Computers and Digital Techniques,* 2003. Proceedings of the IEEE, [s.n.], pag. 294-302. Doi: 10.1049/ip-cdt:20030830.

[Silvestre e Bachiega, 2007] SILVESTRE, E., BACHIEGA, P. Estudo Sobre Processador ARM7, Universidade Federal de Santa Catarina, Florianópolis, Fevereiro 2007.

[Simplescalar, 2010] Simplescalar. Disponível em: www.simplescalar.com. Acessado em:: 28/jan/2010.

[Smith e Sohi, 1995] SMITH, J.E.; SOHI, G.S.The microarchitecture of superscalar processors. In: *Proceedings of the IEEE*, 1995. Proceedings of the IEEE, [s.n.], pag.1609-1624. Doi: 10.1109/5.476078

[Simunic et al., 2001] SIMUNIC, T., et al. Dynamic Voltage Scaling and Power Management for Portable Systems. In: *Annual ACM IEEE Design Automation Conference*. Procedings., June, 2001, Las Vegas.

[Stacpoole e Jamil, 2000] STACPOOLE, R.; JAMIL, T. Cache memories. *IEEE Potentials,* vol.19, no.2, pag. 24-29, Apr/May 2000.

[Sun, 2010] Sun Microsystems. Disponível em: http://www.br.sun.com. Acessado em: 13/jul/2010.

[Sweazey e Smith, 1986] SWEAZEY, P., SMITH A. J. A class of compatible cache consistency protocols and their support by the IEEE futurebus. In: *13th nternational Symposium on Computer Architecture*, 1986.

[SystemC, 2010] SystemC. Disponível em: http://www.systemc.org/home. Acessado em: 21/jul/2010.

[Thacker et al., 1988] THACKER, C. P., STEWART, L.C., SATTERTHWAITE, Jr. E.H. Firefly: A multiprocessor workstation. *IEEE Transactions on Computers*, Vol. 37, no 8, pag :909–920, 1988.

[Wagner e Carro, 2003] WAGNER, F.; CARRO, L. livro das jornadas de atualização em informática , capítulo 2: Sistemas Computacionais Embarcados. 153 – 158, Campinas, v. 1, 2003.

[Wolf, 2001] WOLF, W. *Computers as Components - Principles of Embedded Computing. System Design*. 1st Edition. San Francisco: Morgan Kaufmann Publishers, 2001.

[Xinmin et al., 2003] XINMIN. T.; YEN-KUANG. C.; GIRKAR. M.; GE. S.; LIENHART. R.; SHAH. S. Exploring the use of Hyper-Threading technology for multimedia applications with Intel® OpenMP compiler. In: *Parallel and Distributed Processing Symposium*, 2003. Proceedings of the International Parallel and Distributed Processing Symposium, [s.n], pag. 22-26. Doi: 10.1109/IPDPS.2003.1213118.

[Zeferino e Susin, 2003] ZEFERINO, C.A.; SUSIN, A.A. SoCIN: A Parametric and Scalable Network-on-Chip. In: *16th Symposium on Integrated Circuits and Systems Design*, 2003. Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, São Paulo, pag. 169-174.

[Zivojnovic, 2007] ZIVOJNOVIC, V. Symchronous Debugging of Multicore Systems. VP ESL Tools ARM Ltda. MPSoC'05. July, 2007.