JUSSARA MARANDOLA KOFUJI

MÉTODO OTIMIZADO DE ARQUITETURA DE COERÊNCIA DE CACHE BASEADO EM SISTEMAS EMBARCADOS MULTINÚCLEOS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

São Paulo

2012

JUSSARA MARANDOLA KOFUJI

MÉTODO OTIMIZADO DE ARQUITETURA DE COERÊNCIA DE CACHE BASEADO EM SISTEMAS EMBARCADOS MULTINÚCLEOS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia.

Área de Concentração: Sistemas Eletrônicos

Orientação: Prof. Dr. Marcelo knorich Zuffo

São Paulo

2012

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 30 de janeiro de 2012.

Assinatura do autor _____

Assinatura do orientador _____

Ficha catalográfica

Kofuji, Jussara Marandola
Método otimizado de arquitetura de coerência de cache baseado em sistemas embarcados multinúcleos / J. M. Kofuji -- ed.rev. -- São Paulo, 2012.
105 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Sistemas embutidos 2. Desenvolvimento de microprocessadores 3. Geração de código (Otimização) 4. Análise de desempenho (Simulação) 5. Benchmarks I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II. t.

AGRADECIMENTOS

Eu gostaria de agradecer minha experiencia na Universidade de Sao Paulo durante o meu programa de doutorado em Engenharia Elétrica da Escola Politécnica, especialmente meu supervisor de tese, prof Marcelo Knorich Zuffo, prof. Sergio Kofuji, prof. Edson Midorikawa, prof. Edson Horta e meus amigos do departamento, Roberto Kenji Hiramatsu e também minha participação aos programas OpenSPARC e IBM Cell BE. Meus sinceros agradecimentos aos colegas Leonardo Garcia e Durgam Vahia.

Aos meus pais por todo o suporte, esforços e também por acreditar em mim, dar a confiança para conquistar meu desafio com sucesso. A minha filha Tatiana por toda a compreensão, confiança e amor.

Tenho um grande reconhecimento de ter tido a oportunidade de trabalhar como pesquisadora a Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA) no laboratório de Sistema Embarcado e Tempo Real (LaSTRE) participando do projeto Challenge Innovation de la DRT 2008 – 2009, Tráfico de coerência otimizado por co-design aplicado as arquiteturas multicores embarcada.

Eu não imaginava realizar uma temporada científica na França durante os meus estudos de doutorado e concluo afirmando que mesmo sem ter passado por uma Universidade Francesa, eu tive a oportunidade de acompanhar o trabalho professional de perto, todos os dias de muitos pesquisadores franceses seniors, os quais eu tive a chance de ter tido contato.

Eu agradeço ao senhor Vincent DAVID, diretor do LaSTRE, por me receber e ter tido me aceitado no contexto das atividades de pesquisa do laboratório, pela disciplina e organização dos objetivos do projeto que me conduziram ao sucesso da redação do nosso memorial técnico e por ter conduzido bem o projeto após a transição como supervisor. Agradeço também o anjo guardião do projeto, senhor Thierry COLETTE, pelo suporte do projeto e sugestões particularmente preciosas durante o Forum de Inovação à Grenoble.

Ao senhor, Loïc CUDENNEC, colega do projeto, por me acompanhar durante a direção do projeto. Eu desejo agradecer de ter aprendido a respeitar certos rigores no trabalho e a ser mais metódica. Obrigado por ter me ajudado a realizar a compilação de múltiplas plataformas de sistemas embarcados.

Eu não poderia esquecer meu colega de domínio de computação de alto desempenho e primeiro supervisor de projeto: senhor Jean-Thomas ACQUAVIVA de ter tido confiança em mim uma vez que estive bem longe de imaginar elaborar uma proposta de tese na tópico de tráfico de coerência de cache aplicado as arquiteturas embarcadas.

Obrigado pela orientação excepcional oferecida, pela competência, profissionalismo e disponibilidade.

Aos meus colegas do LaSTRE, particularmente Stéphane LOUISE que esteve sempre disponível por corrigir minha redação, fornecer sugestões e suporte por chegar ao fim com sucesso. À Renaud SIDNEY, agradeço imensamente por toda a orientação, confiança e idéias para continuação da minha pesquisa. E finalmente, Thierry GOUBIER, Selma AZAIEZ, Julien HERVE, realmente um grande prazer que eu tenho tido de estar entre vocês.

REMERCIEMENTS

Je voudrais remercier pour l'expérience qu'il m'a été donné de vivre au sein de l'Université de Sao Paulo en tant qu'étudiante de doctorat du programme d'enginerie electrique de l'École Polytechnique, en particulier, mon superviseur de thése, le professeur Marcelo Knorich Zuffo, le Pr. Sergio Kofuji, le Pr. Edson Midorikawa, le Pr. Edson Horta, mes amis du département, Roberto Kenji Hiramatsu, et également pour ma participation aux programmes internationaux OpenSPARC et IBM Cell BE. Mes remerciements sincères à mes collègues Leonardo Garcia et Durgam Vahia.

A mes parents pour tout le support, les efforts et aussi pour m'a fait la confiance pour réussir mon challenge avec sucèss. A ma fille Tatiana pour ta comphrénsion, confiance et l'amour.

Je suis très reconnaissante d'avoir eu l'opportunité de travailler au sein du Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA) dans le Laboratoire de Système Embarquée et Temps Réel (LaSTRE) en participant du projet Challenge Innovation de la DRT 2008 – 2009 sur le sujet : Trafic de cohérence optimisé par co-design appliquée aux architectures multicœurs embarquées.

Je n'imaginais pas effectuer un séjour en France pendant mes études de doctorat et je vous confirme que même sans [avoir fréquenté d'<pouvoir passer à une>] Université Française, j'ai eu l'occasion d'accompagner le travail professionnel au fil des jours de plusieurs chercheurs français avec qui j'ai eu la chance d'avoir des contacts.

Je remercie Monsieur Vincent DAVID, chef LaSTRE, pour m'avoir accueillie dans le cadre des activités de recherches du laboratoire, pour la discipline et l'organisation des bilans de projet qui a conduit au succès de la rédaction de notre mémoire technique et pour avoir conduit à bien le projet après la transition, en tant que porteur du projet. Je remercie aussi l'ange gardien du projet, Monsieur Thierry COLETTE, pour tout le soutien sur le projet et des suggestions particulièrement précieuses lors du Forum de Challenge et Innovation à Grenoble. À Isabelle Touet, chargée de mission scientifique, pour ta sympathie. À tous le secrétariat de la DRT/LIST.

À Monsieur, Loïc CUDENNEC, collègue du projet, pour m'avoir accompagnée pendant la direction du projet. Je désire le remercier de m'avoir appris a respecter une certaine rigueur dans le travail et à être

plus méthodique. Merci pour m’avoir aidée à réaliser la compilation multiplateformes pour les systèmes embarquées.

Je ne pouvais pas oublier mon collègue du domaine de la Haut Performance et premier porteur du projet : Monsieur Jean-Thomas ACQUAVIVA de m’avoir fait confiance alors que j’étais bien loin d’imaginer de proposer une thèse au sujet du trafic de cohérence de cache appliquées aux architectures embarquées.

Merci pour l’encadrement exceptionnel que tu m’as offert, pour ta compétence, ton professionnalisme et ta disponibilité.

À mes collègues du LaSTRE, particulièrement Stéphane LOUISE d’avoir toujours été si disponible pour corriger ma rédaction, fournir des suggestions et du soutien pour y arriver. À Renaud SIDNEY, un très grand merci pour toute l’orientation, la confiance et des idées pour la suite. Et finalement, Thierry GOUBIER, Selma AZAIEZ, Julien HERVE, vraiment tous le plaisir que j'éprouve d’avoir pu être parmi vous.

“Imagino Design de processadores modernos,

Penso logo em Coerência de Cache,

Passo os dias a pensar: Atingir o melhor tempo de execução,

Introduzindo a latência por padrões de acesso à memória,

E explorando a localidade e espacialidade do cache

Para uma arquitetura multicore embarcada em minha vida”.

Jussara Marandola Kofuji

RESUMO

A Tese apresenta um método de arquitetura de coerência de cache especializado por sistemas embarcados. Um das contribuições principais deste método é apresentar uma proposição de arquitetura CMP de memória compartilhada orientada a padrões de acesso a memória e de um protocolo de coerência híbrido. A contribuição principal é a especificação do novo componente de hardware, chamado tabela de padrões, o qual é validado por representação formal e pela implementação da estrutura da tabela de padrões. A partir desta tabela foi desenvolvido um modelo de transação de mensagens do protocolo híbrido que diferencia as mensagens em clássicas e especulativas. A contribuição final apresenta um modelo analítico do custo efetivo de desempenho do protocolo híbrido.

Palavras-chave: Concepção de processador, descrição de hardware, padrões de acesso à memória, protocolo de coerência de cache.

ABSTRACT

This dissertation presents the optimized method of cache coherent architecture based on embedded systems. The main contribution of this method presents the proposal of shared memory architecture CMP oriented by memory access patterns and cache coherent hybrid protocol. The cache coherent architecture provided the hardware specification called pattern table witch can be validated by formal representation and the first implementation of pattern table. Through pattern table was developed the model of messages transaction to hybrid protocol witch differ the messages in classical and speculative. The final contribution presents the analytic model of effective cost of hybrid protocol performance.

Keywords: Chip Design. Hardware Description. Memory Access Patterns. Cache Coherent Protocol

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTAS DE ABREVIATURAS E SIGLAS

ACK	acknowledgement
CMP	Chip multiprocessing
CPU	chip processor unit
CMT	chip multithreading
FPGA	field programmable gate array
GPU	graphics processing unit
HMPP	heterogeneous multicore parallel programming
HMT	hardware multithreading
HPC	high performance computing
ILP	instruction level parallelism
Manycore	múltiplos processadores
MPSoC	multiprocessing system on chip
NUMA	non uniform memory access
MESI	modified, exclusive, shared, invalid
MIPS	millions instructions per second
NUCA	non uniform cache access
RTL	Register Translation level
SMS	spatial memory streaming

STMS	spatio-temporal memory streaming
TMS	temporal memory streaming
TLP	thread level parallelism

SUMÁRIO

Capítulo 1 – Introdução

Pensando na concepção de processadores multinúcleo, em termos de arquiteturas otimizadas, a tese concentrou-se no problema da limitação de memória para prover um ganho de desempenho ideal no que tange ao tempo de execução de aplicações para processadores de propósito específico.

O contexto inicial deste trabalho científico aborda uma arquitetura especializada por sistemas embarcados, trata-se de uma microarquitetura de um processador com milhares de núcleos de processamento em um único *chip*, ou seja, arquiteturas massivamente paralelas em um *chip*.

Alguns pontos considerados na concepção de um processador, como tempo de execução, largura de banda e eficiência de energia, são relativos no que diz respeito ao foco da arquitetura (podendo ser dedicados à computação de alto desempenho ou a sistemas embarcados).

A Tese de doutorado propõe um método otimizado de coerência de *cache* baseado em abstração de *hardware/software* para arquiteturas de sistemas embarcados.

A solução baseada *hardware* é uma arquitetura de coerência de *cache* otimizada para aplicações de visão computacional embarcada. As características desse *hardware* pressupõem:

- *Hardware* dedicado a gerenciamento de padrões de acesso à memória;
- Programabilidade multinúcleo;
- Concepção de hierarquia de *cache*

A solução baseada em *software* é um protocolo híbrido, orientado a padrões de acesso à memória, e pressupõe uma análise de código em termos de gerenciamento de dados entre os *caches manycore*, como:

- Modelo de coerência de *cache*;
- Protocolo de coerência de *cache*;

Nas próximas seções, apresentaremos os conceitos de processadores multinúcleo e coerência de *cache* inerentes ao princípio de evolução, concepção de processador e princípio básico de *cache*.

1.1 Arquiteturas Multinúcleo

O contexto da Tese está inserido em questões importantes para a indústria e a academia no que diz respeito à concepção de *hardware* e *software* relacionadas ao desempenho de processadores multinúcleo dedicados ao tratamento de padrões de acesso à memória.

A arquitetura de coerência de *cache* proposta na Tese baseia-se em uma arquitetura CMP (*Chip Multiprocessing*) de memória compartilhada baseada em diretórios que introduz um novo componente arquitetural e a implementação de um protocolo de coerência de *cache* otimizado ao propósito específico para o processador.

O método otimizado de arquitetura de coerência de *cache* baseado em *hardware/software* concentra-se em sistemas embarcados. Uma arquitetura CMP pode ser concebida tanto para sistemas de alto desempenho, quanto para sistemas embarcados, apresentando as seguintes características:

- arquitetura CMP de sistema embarcado: o objeto de estudo é a análise de otimização do custo do silício e a vazão (com intuito de reduzir a potência);

- arquitetura CMP HPC (*High Performance Computing*): o objeto de estudo é a otimização da latência de acesso à memória.

Os processadores "*chip multithreading*" permitem executar processos múltiplos em um *chip* paralelo. Esse paralelismo de *hardware* se utiliza dos seguintes métodos: processos múltiplos por processador *hardware multithreading - HMT* e processador com vários núcleos de processamento "*chip multithreading*" (FEDEROVA, 2006).

A arquitetura *multithreading* se enquadra dentro da categoria de *chip* basedo em diretórios de caches. Uma arquitetura dita "*baseline*" apresenta caches L1 privados e caches L2 privados. A concepção da arquitetura de coerência de cache propõe a implementação de caches híbridoss: *caches* L2 compartilhados e *caches* L1 privados.

Os mecanismos de *caches* são fatores pertinentes para que um desempenho exponencial cada vez maior seja atingido através da técnica de paralelismo de nível de processos - *thread-level parallelism – TLP* (MARTHY, 2008). O TLP é um fator muito questionado na concepção de *chip* CMP.

Atualmente, a memória principal em arquitetura multinúcleo tem uma alta demanda de vazão e capacidade (Intel) (Power4).

A partir do estudo aprofundado de arquiteturas CMP, será introduzido questões importantes para a indústria e a academia em termos de concepção de *hardware* e *software*, concernentes ao desempenho de processadores multinúcleo. Atualmente, na era *manycore*, os processadores utilizam centenas a milhares de processos (*threads*) e possuem múltiplos *chips* com baixa frequência de relógio de CPU para solucionar o problema de *cache* e da limitação de potência de forma a oferecer benefícios de desempenho. Pode-se constatar dois problemas relacionados ao fator frequência:

1. frequência da CPU aumenta muito mais que a frequência da memória, de modo que, para manter um bom desempenho, define-se a noção de hierarquia de memória (*caches*);
2. a taxa de dissipação térmica e a consumo de energia aumenta após o aumento multinúcleos.

Para solucionar o problema de frequência relacionado à questão de *cache* e potência, o método otimizado de arquitetura de coerência de *cache* estará concentrado na consistência de dados e no padrão de acesso à memória. Podemos destacar uma nova concepção de hierarquia de memória que dispõe de uma memória auxiliar (considerada um *cache*) para armazenar em forma de padrões regulares os endereços acessados recentemente.

Os processadores multinúcleo surgiram para tratar questões relacionadas à barreira de potência, forçando, dessa forma, a indústria de *software* a migrar o modelo de programação sequencial para um modelo de programação paralela, com alto grau de paralelismo explícito (ASONOVIC, 2006).

A Arquitetura de Von Neumann serviu de base a diversas arquiteturas de computadores modernas, sendo muitas de suas ideias válidas ainda hoje: o conceito de arquitetura lógica de computador considerando sua execução física (KOWALTOWSKI, 1996) (MACK, 2011). A Arquitetura de Von Neumann é associada à uma arquitetura clássica de computadores digitais com programa armazenado em memória.

Uma das motivações para o surgimento dos processadores multinúcleo foi o Gargalo de Von Neumann. Na figura 1, a seguir, será observado a comparação do desempenho da CPU por ano. A memória não acompanha esse desempenho, incrementado pela frequência.

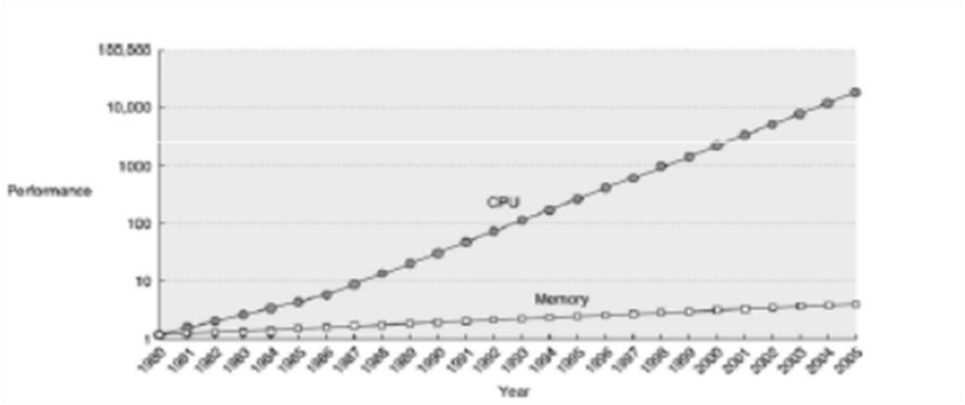


Figura 1: Gargalo de Von Neuman (MACK, 2011)

Esse gargalo de Von Neuman está relacionado ao problema 1 (relacionado ao fator de frequência de CPU), a ser solucionado pela coerência de *cache*.

Para entender a microarquitetura das arquiteturas multinúcleo, deve-se apresentar os fatores que levaram o desempenho dos processadores a evoluir, entre os quais, podemos destacar o aumento da frequência de relógio e potência de *chip*, aumento da capacidade de memória, aumento de Transistores, miniaturização do *chip* e mudança no projeto de *hardware*.

Os fatores de desempenho mais importantes referentes à microarquitetura são citados por Kunle (OLUKOTUM, 2007):

- miniaturização de transistores;

- velocidade de relógio;

- diminuição de potência;

- aumento do nível de paralelismo de instruções;

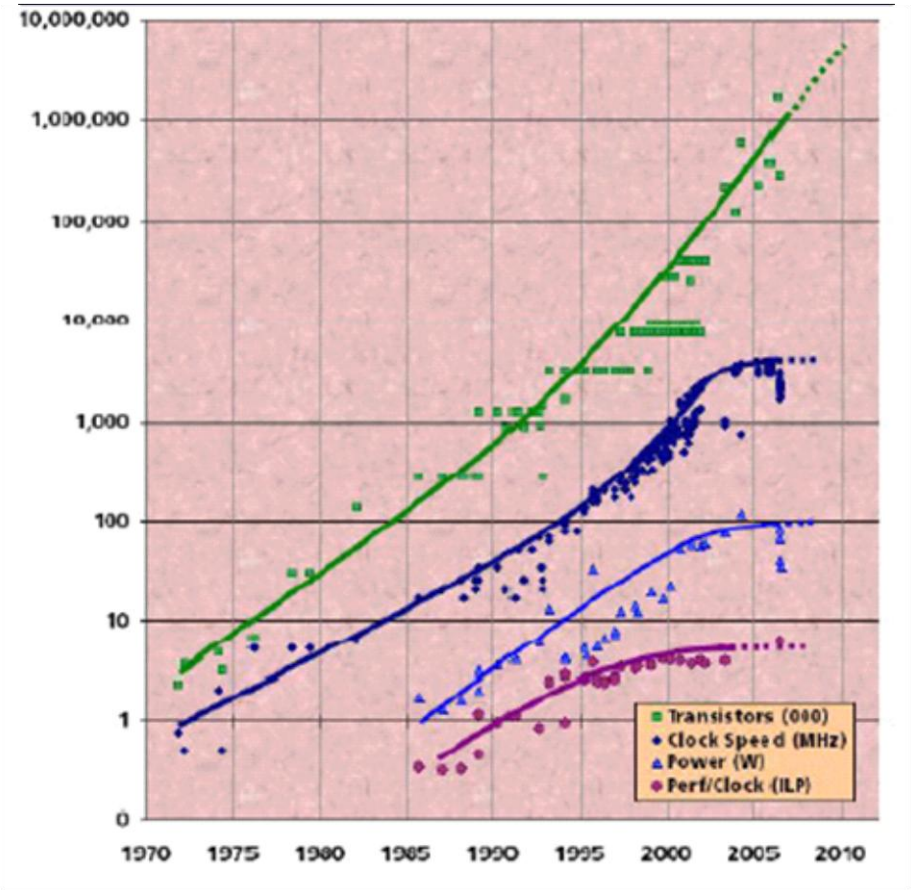


Figura 2: Fatores que afetam o desempenho (OLUKOTUM, 2007)

Escala de quantidade de transistores em relação ao ano

Na figura 2 é possível verificar que o aumento do número de transistores é crescente, chegando a quase 1.000.000,00 milhão de transistores por *chip*; a velocidade do *clock* (em MHz) é crescente, estabilizando-se em um determinado ponto; a potência inicia-se em 1, estabilizando-se em 100, e, por último, o desempenho/*clock* (ILP – *Instruction Level Parallelism*), que é bem limitado.

Atualmente, constata-se que a nova barreira de desempenho, composta das barreiras anteriores: limite de memória, limite da potência e limite de ILP. Esta última técnica, chamamos de paralelismo explícito.

O contexto de processadores multinúcleo em que relacionamos nossa arquitetura visa a noção de múltiplos *cores* e paralelismo em nível de instruções, técnica a qual chamamos de paralelismo implícito ou simplesmente arquitetura CMT (*Chip Multithreading*). Um *chip multithreading* consiste em N *cores* e M threads por processador.

A Arquitetura de Coerência de *Cache* apresentada na Tese baseia-se em uma Arquitetura CMP (*Chip Multiprocessing*) com memória compartilhada, ou seja, memória compartilhada entre todos os processadores. A arquitetura em questão poderá implementar uma memória compartilhada do tipo NUMA (*Non Uniform Memory Access*) ou do tipo NUCA (*Non Uniform Cache Access*).

Em termos de coerência de *cache*, o sistema deverá ser capaz de manter a consistência de dados em memória, ou seja, garantir a coerência de dados em *cache*.

Em se tratando de *cache*, baseamo-nos em uma arquitetura padrão CMP com *cache* compartilhado baseado em diretório. Em um primeiro momento, apresentaremos a diferença entre uma arquitetura CMP com e sem *cache* baseado em diretório.

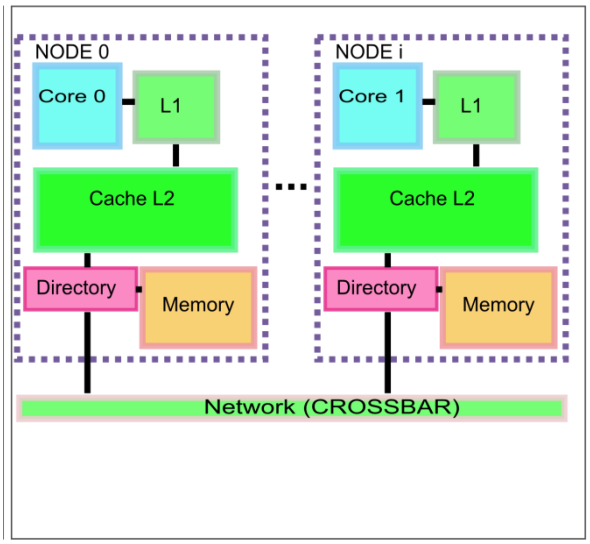


Figura 3: Arquitetura CMP com *cache* baseado em diretório

Na arquitetura CMP da figura 3, em cada nó de processamento, observa-se os seguintes componentes: o *chip* (*core*), *cache* L1 privado, *cache* L2 compartilhado, diretório de coerência de *cache*, memória principal e uma rede de interconexão *crossbar*.

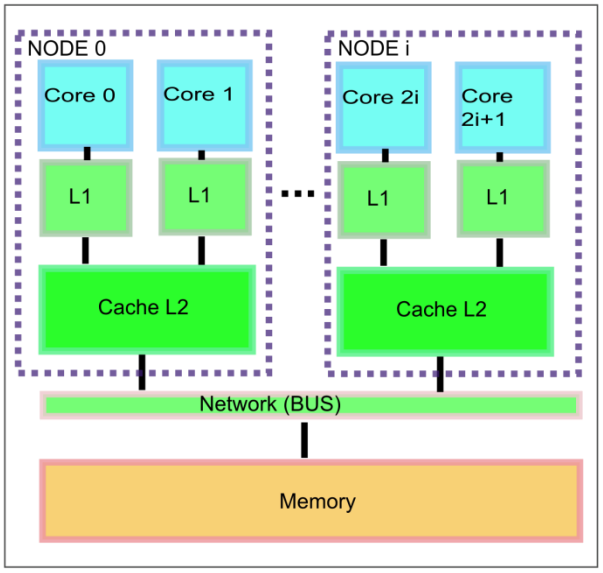


Figura 4: Arquitetura CMP com *cache* sem diretório

Contudo, nessa outra arquitetura CMP, representada pela figura 4, os nós de processamento não apresentam um diretório de coerência de *cache* e a memória principal não está localizada na estrutura do nó de processamento, tendo cada *core* de acessar a memória através da rede de interconexão. Isso significa que o tempo de acesso é muito maior, o que resulta em um desempenho ineficiente.

Inicialmente, analisamos arquiteturas multinúcleo sem coerência de *cache*. Podemos destacar, no estado da arte, as arquiteturas heterogêneas (em especial o processador CELL) (SCARPINIO, 2008), as arquiteturas híbridas (processadores multinúcleo e aceleradores – GPU ou FPGA) e, finalmente, as arquiteturas homogêneas. Todo o estudo de arquiteturas multinúcleo envolveu o contexto inicial de computação de alto desempenho. Após uma análise de três aspectos como

programabilidade, microarquitetura, desempenho em arquiteturas heterogêneas como o processador Cell, tendo em visto uma avaliação de desempenho de aplicações de detecção de faces muito inferior que para arquiteturas homogêneas, foi introduzido arquitetura CMP para contextualizar o problema de coerência de cache em memória compartilhada, especialmente para sistemas embarcados.

A seguir, definiremos o problema cientifico da Tese no que diz respeito à coerência de *cache* e ao componente de *hardware* para tratamento de padrões e implementação de protocolo.

1.2 Problema Científico

O problema principal da Tese concentra-se na proposição de uma arquitetura orientada a padrões de acessos, que mantenha os dados consistentes, e permita, dessa forma, um desempenho de aplicações de visão computacional para sistemas embarcados, que considerem padrões de acesso à memória. Um método otimizado de Arquitetura de Coerência de *Cache* baseado em *hardware*/*software* foi desenvolvido.

Para isso, podemos citar problemas específicos:

- **Conceber uma arquitetura orientada a padrões de acesso**

- Estabelecer a arquitetura *baseline* como o fundamento de nossa arquitetura de coerência de *cache* e modificar as características de hierarquia de *cache*, transformando-a em uma arquitetura CMP de memória compartilhada baseada em diretórios;

- Conceber um componente de *hardware* que armazene os padrões de acesso à memória, criando uma estrutura de *cache* para manipular padrões;

- Definir um formato para o padrão de acesso à memória.

- **Conceber um protocolo de coerência de *cache* híbrido** que gerencie mensagens *baseline* e mensagens especulativas, de acordo com a especificação da microarquitetura do processador.

1.3 Objetivos

O objetivo geral é desenvolver um método otimizado de arquitetura de coerência de *cache* baseado em *hardware/software* para sistemas embarcados, com intuito de obter o melhor desempenho de aplicações que considerem padrões de acesso à memória, fazendo uso de um componente *hardware* orientado a tratamento de padrões de acesso e otimizar o protocolo *baseline*, diferenciando as mensagens clássicas das genéricas.

Pode-se citar alguns objetivos específicos para solucionar os problemas abordados anteriormente:

O primeiro, é propor um componente de *hardware* que permita armazenar padrões e otimizar aplicações que considerem padrões de acesso à memória; permita uma concepção que reduza o custo do silício e seja inserido no modelo de hierarquia de *cache* de uma arquitetura CMP de memória compartilhada baseado em diretório.

O segundo é desenvolver um protocolo de coerência de *cache* especulativo orientado ao padrão de acesso à memória destinado a sistemas embarcados, que analise o ganho de tempo de execução e o aumento da largura de banda.

Outro objetivo específico é conservar a coerência de *cache*, de acordo com a diferenciação entre mensagens *baseline* e especulativa, com base no método *round-robin*.

1.4 Contribuições

- **Contribuições pela Arquitetura de Coerência de *Cache***

As próximas contribuições referem-se à arquitetura de coerência de *cache*. Elas visam oferecer um método otimizado, que permita o ganho de desempenho de aplicações que considerem padrões de acesso à memória.

- Componente de *hardware* (Tabela de Padrões de Acesso)

A contribuição mais importante na concepção desse componente de *hardware* foi a definição de uma estrutura de armazenamento de padrões de acesso. Primeiramente, foi essencial responder ao problema científico: qual o formato do padrão de acesso, o tamanho da tabela, identificação do endereço base, a técnica de pesquisa de endereços na tabela e, por último, como orientar esse *hardware* para transação de mensagens de um protocolo de coerência de *cache* híbrido.

- Protocolo de Coerência de *Cache* Híbrido

Esse protocolo gera um desempenho de tempo de acesso ao endereço e uma consistência de dados de forma mais otimizada. A contribuição mais evidente nesse quesito foi a diferenciação entre mensagens especulativas e mensagens clássicas (ditas *baseline*). A grande vantagem desse protocolo é o fato de ele evitar o *hotspot* do sistema (sobrecarga de mensagens no *home node*), ou seja, ele proporciona um *throughput* e um melhor tempo de acesso. A partir de dois protocolos clássicos (*proximity-aware* / *alternative home node*), implementamos uma técnica de pesquisa do *home node* chamada “*round-robin*”. Essa técnica atende à

concepção do novo protocolo direcionada a padrões de acesso à memória, intercalando mensagens de padrão de acesso e mensagens clássicas.

- **Publicações**

As publicações relacionadas diretamente com a presente tese foram, sobretudo, no sentido em que este trabalho científico foi desenvolvido com o propósito de produzir uma propriedade intelectual e, posteriormente, de publicação dos resultados experimentais à comunidade científica, chegando, por fim, à transferência de tecnologia à indústria.

Durante o período de realização do projeto de pesquisa “Challenge et Innovation”, em colaboração com o CEA (Commissariat à l’Energie Atomique et aux Energies Alternative), foram realizados:

- artigo internacional (MARANDOLA et al, 2011);

- memorial técnico interno ao CEA-LIST (KOFUJI et al, 2010);

- relatório técnico interno ao CEA-LIST (KOFUJI, 2010);

- pôster apresentado ao Forum de Challenge et Innovation 2010 (KOFUJI et al, 2010).

Anteriormente às publicações diretamente relacionadas à tese, houve outras produções importantes que culminaram na base de simulação de processadores, programação paralela e processamento de imagens:

- pôsteres internacionais (HIRAMATSU et al, 2009), (MATTES et al, 2009);

- minicurso no evento WSCAD2007 (KOFUJI et al., 2007).

Um capítulo de livro relacionado ao Ensino de Arquitetura de Computadores foi aceito para publicação eletrônica; encontra-se, no entanto, aguardando indexação da revista:

- capítulo de livro (KOFUJI et al., 2010).

1.5 Organização da Tese

A organização da Tese está baseada nos seguintes tópicos:

- Introdução – apresenta as principais contribuições, o contexto inicial, o problema científico, objetivos e metas;

- estado da arte – composto de tópicos da literatura, referentes às aplicações que utilizem padrões de acesso à memória, arquiteturas Multinúcleo de memória compartilhada e coerência de dados;

- arquitetura de coerência de *cache* otimizada a padrões de acesso à memória – analisa uma arquitetura padrão denominada *baseline* e propõe uma arquitetura de coerência de *cache* baseada em padrões de acesso relacionado a um protocolo híbrido orientado ao fluxo de mensagens desse *hardware* otimizado e um método de implementação de simulação;

- validação da arquitetura e protocolo – nesse capítulo, validaremos a arquitetura através de uma implementação inicial da estrutura da tabela de padrões, que será um módulo de uma biblioteca API que descreva o comportamento do

hardware e por um modelo analítico para avaliação do custo de desempenho do protocolo de coerência híbrido;

- conclusão – apresenta as principais contribuições da Tese, concluindo as vantagens e trabalhos futuros.

Capítulo 2 – Estado da Arte

A Tese introduz o problema de manipulação de padrões de acesso para otimizar o sistema de coerência de *cache* no contexto de arquiteturas multinúcleo com memória compartilhada. O problema de manutenção de coerência de cache é um problema antigo abordado exaustivamente na literatura, contudo, o problema aborda questão de otimização de aplicações em hardware que se utilizam de técnicas conhecidas como "*memory access patterns*".

Um sistema de coerência de *cache* permite-nos a gestão de múltiplas cópias de um único dado situadas em diferentes *caches* dos núcleos de processamento.

Para analisar a gestão da coerência de *cache*, concentramo-nos no comportamento do padrão de acesso à memória.

O estado da arte da literatura aborda o estudo de processadores embarcados massivamente paralelos especializados para aplicações que apresentem um alto grau de padrões de acesso à memória, principalmente aplicações de visão computacional e problemas científicos para processadores embarcados. Para esse tópico em específico, foram analisadas diversas patentes tecnológicas que abordam a questão de processadores aceleradores de padrões de acesso à memória. Padrões de acesso à memória constituem informações suplementares que permitem especular e antecipar os próximos acessos à memória.

Outro tópico importante abordado na literatura é a introdução do contexto da arquitetura estudada nesta pesquisa: arquiteturas multinúcleo com memória compartilhada. Inicialmente, daremos ênfase às arquiteturas CMP com base no modelo *baseline* e, a seguir, apresentaremos uma visão geral dessa mesma arquitetura baseada em um circuito integrado com múltiplas funções, também chamado *Multiprocessing System on Chip – MPSoC*.

Em termos de coerência de dados, baseamos nossa literatura nos seguintes tópicos: informação de coerência, modelo de consistência de dados e protocolo de coerência de *cache*.

Em cada tópico de coerência de dados, apresentaremos uma referência padrão na qual se baseia nosso método de coerência de *cache* baseado em *hardware*/*software*. Esse método apresenta referências sobre protocolo de coerência de *cache* (protocolo MESI), modelo de consistência de dados e protocolo *baseline*.

2.1 Aplicações de Padrões de Acesso baseadas em Visão Computacional

O novo componente de *hardware* concebido para compor o modelo de hierarquia de memória tem como objetivo oferecer melhor desempenho para aplicações chave, que têm como base os padrões de acesso à memória.

A análise desses padrões é focada em uma aplicação de visão computacional em sistemas embarcados. Em princípio, como estudo de caso, trataremos o comportamento do padrão de acesso de algoritmos, como a imagem integral e a multiplicação de matrizes em 2D.

O próximo tópico apresenta exemplos de padrões de acesso regular à memória e trabalhos correlatos que analisam o mesmo conceito.

2.1.1 Padrões de Acesso à Memória

Na literatura, analisamos trabalhos correlatos que propõem a aceleração de acesso aos dados baseados em padrões de acesso à memória. O conceito de tabela de padrões é introduzido como um novo componente de hardware composto de diversos padrões. Estes armazenam informações suplementares que possibilitam a antecipação e/ou especulação de mensagens (próximos acessos).

O estado da arte toma por base a temática da base de dados, a memória compartilhada distribuída e a gestão de *cache* de processadores.

Podemos definir um padrão de acesso regular à memória, o exemplo de acessos regulares representado a seguir. Os padrões de acesso à memória em caso de *cache* “miss”, quando não é locazado um determinado endereço em *cache*. Na figura 5, um padrão de acesso é representado de forma regular no tempo e a extração de um padrão de acesso armazenado em um arquivo.

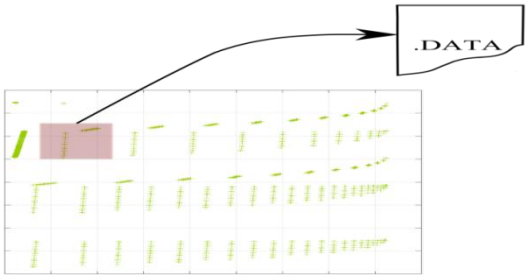


Figura 5: Exemplo de representação de padrão de acesso (KOFUJI, 2010)

A figura 5, apresenta de forma genérica, uma representação de padrão de acesso, em que, na linha horizontal, verifica-se a ordem em que os padrões aparecem no decorrer do tempo e, na linha vertical, a quantidade de acessos. O retângulo representa, por exemplo, vários endereços de um determinado padrão, armazenados em um arquivo por meio do método de *profiling.*

Em nosso método otimizado de arquitetura de coerência de *cache* baseado em *hardware/software*, concebemos inicialmente uma solução *on-chip* para tratamento de padrão de acessos regulares que será descrito no próximo capítulo.

Os avanços tecnológicos na área de fabricação de semicondutores acerca de inovações, microarquiteturas e circuitos têm levado os processadores a apresentarem um desempenho cada vez maior. Conhecendo bem o fenômeno, a memória acompanhou a rápida aceleração dos processadores resultando no gargalo do processador/memória.

Esse problema é agravado em multiprocessadores de memória compartilhada em que o acesso à coerência de *cache* precisa atravessar os diversos níveis de hierarquia de *cache*, o que implica demora no tempo de acesso, devido ao tráfego da rede, principalmente quando há acesso à memória principal.

Para isso, estudamos os trabalhos correlatos que se propunham a responder a algumas questões:

- como identificar uma sequência de endereços de um padrão de acesso;

- como identificar o código de um padrão de acesso espacial para predição de endereços não encontrados em *cache*;

- como armazenar os endereços e qual formato do padrão de acesso.

Observando essas questões, encontramos trabalhos importantes, um deles mais direcionado a *streams* e não implementado em *hardware*, *Temporal Memory Streaming - TMS* (Thomas, 2005); *Spatial Memory Streaming - SMS* (STEPHEN, 2006), baseado em padrões de acessos irregulares e implementados em *hardware on-chip*, e, finalmente, *Spatio-Temporal Memory Stream* - STMS (STEPHEN, 2007), baseado em padrões de acessos irregulares com formatos diferentes e não implementados em *hardware*.

A similaridade com a tese foca em termos de conceito de tempo e espaço de acesso à memória, porém não orientado a streams e sim aplicações científicas que apresentam padrões de acesso regular à memória. E a diferença do método otimizado é que ele implementa a noção tempo-espacial de acesso à memória em hardware, atribuindo um novo componente de hardware.

A solução TMS apresenta a técnica chamada *temporal streaming*, que identifica e transfere mensagens *stream* compartilhadas entre os processadores. Ela explora os dois conceitos de padrão de acessos compartilhados: a correlação de endereço temporal para representar uma sequência em ordem de endereços de

dados não encontrados em *cache* e localidade de fluxo temporal para representar o fluxo de endereços recém-acessados (caso em que há uma antecipação ou predição de endereços).

Nosso método não é orientado ao fluxo *stream*, tampouco possui um mecanismo próprio para armazenar sequências de endereços em falta no *cache*; é introduzindo apenas um novo componente de hardware que armazena endereços segundo um formato *stride*, tendo o identificador de *pattern* que aponta para nó computacional que detém o dado em questão. Este conceito apresenta a diferença de fluxo de transação de mensagens ditas, baseline e *pattern* em que há transferência de padrão de acesso – *pattern,* para predição de endereços e não de sequência de *streams*.

A solução SMS apresenta duas estruturas em *hardware*: uma tabela de geração ativa que grava os padrões de acessos espaciais (Figura 6) e uma tabela de histórico que armazena os padrões de acessos espaciais previamente observados (Figura 7).

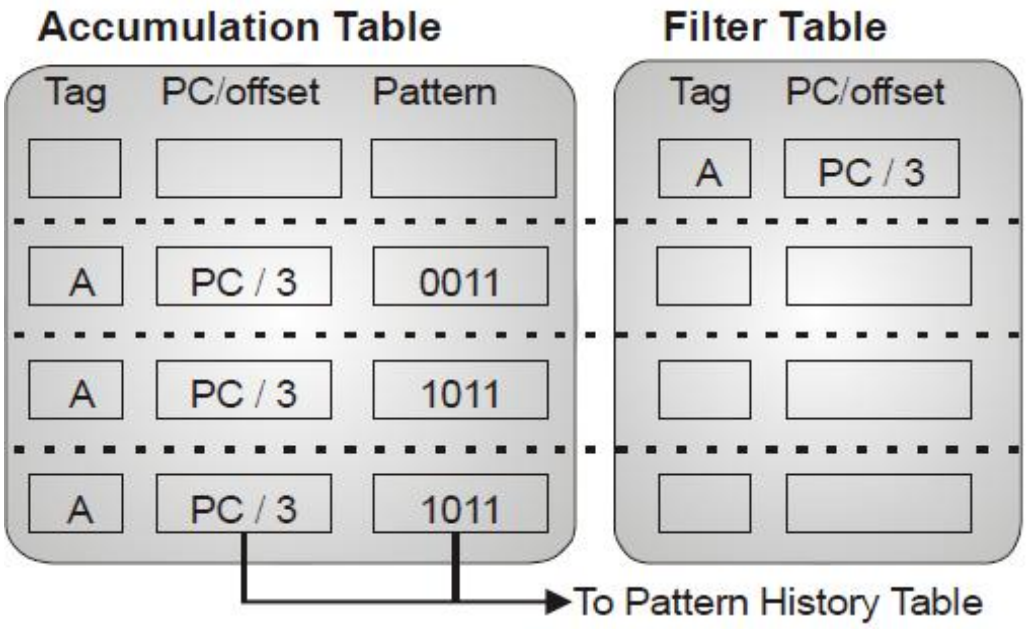


Figura 6: Tabela de Geração Ativa (Thomas, 2005)

A tabela de geração ativa (Figura 6), composta pela tabela de acumulação, apresenta um formato muito semelhante quando representamos um endereço armazenado em diretório de coerência de *cache* (*tag*, *offset*, endereço).

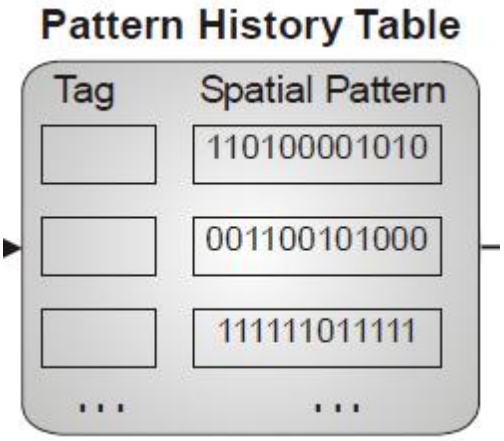


Figura 7: Tabela de Histórico de Padrões de Acessos (Thomas, 2005)

Na figura 7, representada pela tabela de histórico de padrões de acessos, notamos a extração de apenas dois campos da primeira tabela para manter o histórico de padrões de acessos: o *tag* e o padrão.

A diferença fundamental para o nosso método é que o *hardware* apresenta uma estrutura apenas, um componente auxiliar à hierarquia de *cache*, uma tabela que armazena endereços (adicionando o comando *Table lookup*, que localiza o endereço através do campo *offset*). A tabela apresenta então dois campos: *Offset* e *Pattern*. Isso significa que o componente de *hardware* é orientado ao tratamento de padrão de acessos quando identifica um *offset* de *pattern* e o *hardware* ajuda a predição de endereços. O formato do padrão de acessos, por sua vez, é do tipo *strided* para otimizar as aplicações com acesso à memória usando *stride*. Nosso método utiliza um mecanismo de localidade e espacialidade que veremos no próximo capítulo.

2.1.2 Aplicações de Visão Computacional para Sistemas Embarcados

Certas aplicações de visão computacional que requerem leitura/escrita de dados compostos de palavras binárias, armazenadas em memória sem regularidade de acessos, são um exemplo típico de aplicações para tratamento de imagens, sendo bem pertinentes para sistemas embarcados.

Nesse contexto, apresentamos um estudo de caso, uma aplicação de cálculo de imagem integral, conforme descrito em um artigo de B. Kisacanin (KISACANIN, 2008). O tratamento de imagem consiste na soma cumulativa de uma matriz de duas dimensões. Dessa forma, o algoritmo deverá acessar os blocos retangulares da imagem a ser analisada.

Para exemplificar o funcionamento de uma coerência de *cache* clássica operando individualmente sobre essas palavras binárias que compõem o dado, voltamos ao caso da imagem integral. Uma imagem é armazenada, substituindo as linhas L, umas às outras em um espaço de memória do sistema utilizado. Em um sistema multinúcleo, um número de dados M de uma mensagem é requisitado para acesso à memória. A leitura de uma coluna de imagens de uma imagem composta de C colunas e L linhas de pixels produz L x M mensagens, visando a um melhor desempenho do funcionamento de coerência de *cache* clássico e considerando as características previsíveis de acesso à memória a fim de reduzir o número de mensagens requisitadas. Podemos reduzir consideravelmente L x M mensagens requisitadas a M mensagens requisitadas para uma requisição de acesso em leitura/escrita ou requisição de invalidação por exemplo.

Pode-se calcular a imagem integral através de um quadrado, expresso na seguinte equação:

$$I_{m,n} = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j} = \sum_{i=1}^{m} U_{i,n} \qquad (1)$$

$M\ x\ N$ expressa uma matrix de uma imagem $A\ m\ x\ n$ definida pela soma acumulativa de A . No caso, $A\ i,j$ é um elemento da imagem na posição i,j. $U\ i,n$ é representado como somatório cumulativo da coluna de U, sendo o somatório cumulativo da linha A.

2.2 Arquiteturas Multinúcleo de Memória Compartilhada

As arquiteturas multinúcleo de Memória Compartilhada são arquiteturas escaláveis até milhares de *cores* interconectados por meio de uma rede de interconexão escalável baseada na topologia Mesh. As conexões são estabelecidas por nós vizinhos, como na topologia de Redes *Ad-Hoc*. Nesse contexto, o problema de coerência de *cache* ocorre quando são replicadas diversas cópias do mesmo dado em diferentes *caches*, de maneira concorrente para operações de leitura/escrita. As diversas cópias de dados podem ser distribuídas entre os *cores* e a memória principal.

Para manter a consistência de dados, existe um modelo clássico de implementação de protocolo de coerência baseado em diretório, que apresenta quatro estados, como o protocolo *MESI*. Existe uma modificação desse protocolo denominado protocolo de base de referência *Baseline*, uma derivação do protocolo de consistência *Lazy Realese* (LI, 1989), mais utilizado em sistemas de memória compartilhada distribuída.

2.2.1 Arquitetura de Referência

Nesta seção, apresenta-se uma arquitetura CMP de propósito geral que consiste em 64 processadores organizados em uma rede de interconexão do tipo *mesh* (8 x 8). Consideramos esse tipo de arquitetura homogênea, composta de uma memória principal e os 64 processadores (*cores*) em uma estrutura homogênea, sendo que cada processador apresenta um *chip* composto de uma unidade

aritmética e lógica com o *cache* L1 de dados e instruções embutidos (*cache* L1 D considerado privado), *cache* L2 distribuída, diretório de coerência de *cache* (composto de um endereço e a informação de coerência), interface de memória e a rede de interconexão.

A nossa Arquitetura de Coerência de *Cache* adota praticamente o mesmo modelo da Arquitetura de Referência baseado em *tiles*, apresentando especialmente uma concepção de *cache* com consistência baseado entre *caches* compartilhados, o que a difere da Arquitetura de referência *Baseline,* apresentando cache privado*.*

A figura 8, apresenta uma visão geral da Arquitetura de Referência, estendendo um processador (*core*) para exemplificar os componentes, dando ênfase à hierarquia de memória para o estudo da coerência de *cache*.

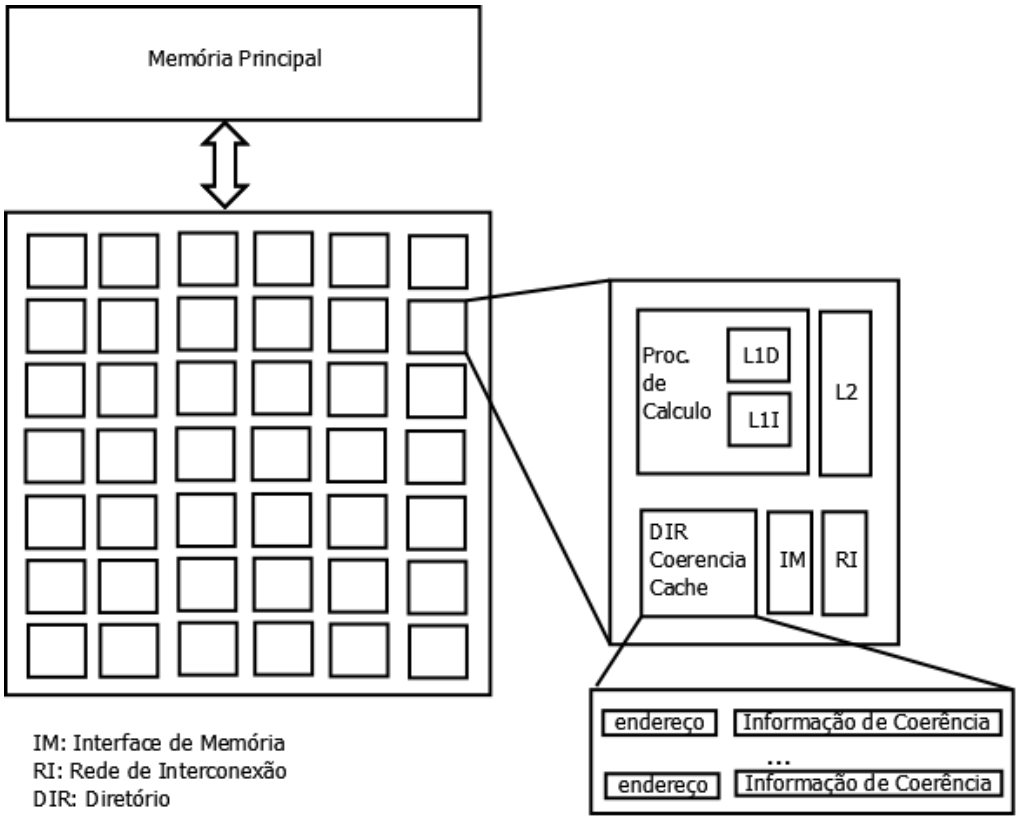


Figura 8: Arquitetura de Referência

O *cache* normalmente tende a armazenar dados com alto grau de localidade espacial, não sendo consecutivos os endereços localizados no diretório de coerência de *cache* (para evitar o congestionamento na comunicação da rede de interconexão).

O diretório de coerência de *cache* armazena o endereço e a informação de coerência para um determinado conjunto de dados armazenados em *cache*. A informação de coerência é composta de (N + 2) bits, sendo N o número de *cores* no sistema (HANDY, 1998).

A unidade padrão de coerência de um sistema de *hardware* corresponde a uma linha de *cache* (64 a 128 bytes dependendo da arquitetura).

A informação de coerência é um vetor de presença de bits, que mantém a coerência quando os dados são armazenados em *cache*. Para manter o desempenho, os dados são armazenados em múltiplas localizações, de forma que a informação espacial se apropria da representação por meio do vetor de presença de bits ou lista ligada, dependendo do número de processadores do sistema.

O tamanho do vetor de presença de bits é igual ao número de *cores*. Na arquitetura de referência representada por 64 *cores*, o vetor será 64 *bits*. Representamos os bits 0 e 1, sendo que 0 em i-ésimo significa que o dado não está localizado em *cache* e 1 em j-ésimo significa que o dado está localizado em *cache*.

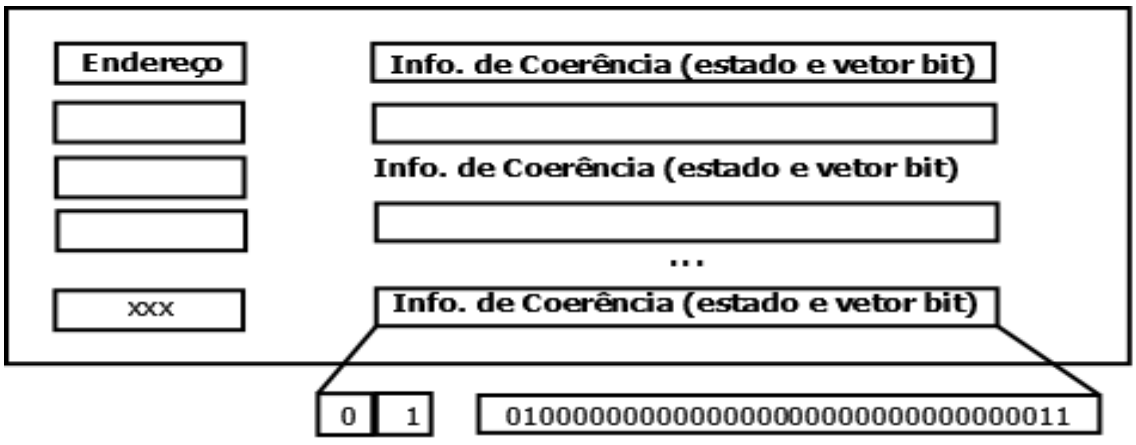


Figura 9: Informação de coerência (vetor de presença de bits)

Normalmente, o processador ao qual foi enviada a requisição de leitura/ escrita de dados, o *home node*, necessita conhecer a localização do dado e o seu estado (modificado, exclusivo, compartilhado, inválido), igualmente ao protocolo *MESI*.

Os estados representam a seguinte mudança:

- Modificado – a cópia modificada em uma operação de escrita, por exemplo, não é igual à cópia armazenada em memória.
- Exclusivo – existe uma única cópia modificada em um bloco de memória e essa é igual à cópia armazenada em memória.
- Compartilhado – existem diversas cópias distribuídas de forma compartilhada entre os processadores, sendo igual à cópia armazenada em memória.
- Inválido – não existem cópias válidas.

Para cada dado gerenciado por um protocolo de coerência de dados, há um nó (processador) chamado *home node*, que centraliza o gerenciamento da informação de coerência para esse dado em particular. Na literatura, existem várias implementações de protocolos de coerência de *cache*, mas citaremos, de modo geral, os protocolos padrão relacionados ao nosso protocolo de coerência híbrido, sobretudo os protocolos *proximity-aware* (JEFFERY, 2007), *alternative home node* (ZHUO, 2008), *MESI* (CHUNG, 2006) e *MESIF* (Intel , 2009) da Intel, derivado do protocolo *Baseline*, descrito na próxima seção.

2.2.2 Protocolo Baseline

O problema observado no tempo de requisição para acessar a memória caso o dado não seja encontrado em *cache* L2 é muito maior que o custo de implementação de um nó de gerenciamento *home node* do protocolo *baseline*, em que o desempenho medido pelo tempo de requisição é bem menor que o acesso à memória principal (JEFFERY, 2007).

Para um protocolo de coerência de *cache* baseado em diretório, um dado não encontrado em *cache* L2 diante de uma requisição de leitura/escrita para uma linha em modo compartilhado ou inválido sempre resulta na busca do dado a partir do *home node*. Entretanto, se o dado não for encontrado no *cache* L2 do home node, os acessos chamados *off-chip* acarretam um custo bem maior.

O protocolo *baseline* considera a busca padrão do dado em *cache* L1/L2 e envia a requisição do dado ao *home node*. Caso o dado não seja encontrado no *cache* L2 do *home node*, a requisição é encaminhada a um outro processador (nó mais próximo). Assim, os dados são compartilhados com o processador que requisitou o dado depois de a mensagem de confirmação *ACK* ser enviada ao *home node*. Nesse caso, todas as requisições enviadas ao *home node* geram um congestionamento de mensagens que chamamos de *hotspot*.

A seguir, apresentamos um modelo geral do protocolo *baseline* para requisição de leitura:

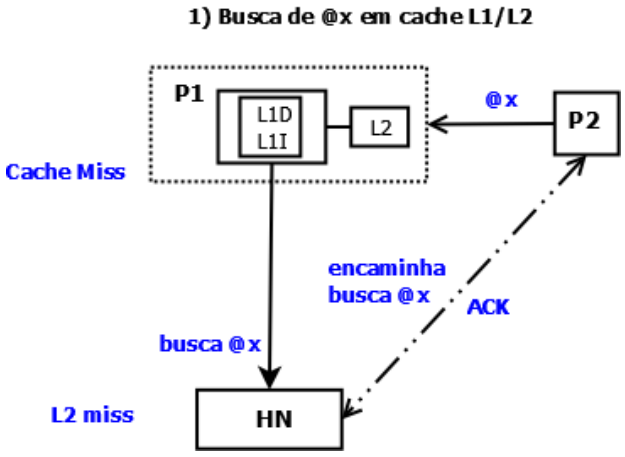


Figura 10: Implementação do protocolo *baseline*

2.2.3 Plataformas MPSoC

O contexto da tese é baseado em plataformas de processadores múltiplos em um único *chip*, o que chamamos de MPSoC – *Multipliprocessor System-on-chip*.

Conforme mencionado no tópico anterior, a respeito da arquitetura de referência, baseamo-nos em uma arquitetura multinúcleo de memória compartilhada e com paradigma de programação de memória compartilhada.

Tanto o MPSoC quanto para Sistemas Híbridos compostos de processadores multinúcleo e aceleradores de *hardware* podem adotar o modelo de arquitetura multinúcleo de memória compartilhada.

Em computação de Alto Desempenho, podemos citar três tipos de sistemas híbridos: arquitetura multinúcleo com GPU integrado (*Graphics Processing Unit*), NVIDIA Tesla (cluster de processadores com GPU) e AMD FireStream (Processador AMD com GPU), todos eles apresentando um novo paradigma de programação paralela chamado HMPP - *Heterogeneous Multicore Parallel Programming* (HMPP). Esse paradigma de programação também é dedicado a processadores embarcados e processadores superescalares. No que se refere a HPC - *High Performance*

Computing, os fabricantes mais eminentes são: AMD64, Intel64, Processadores MIPS.

Para processadores embarcados ou computação reconfigurável, podemos apresentar alguns sistemas clássicos de prototipagem rápida na literatura, que apresentam um *chip* de tamanho pequeno como BEE2 (CHANG, 2005), adotando como elemento primário somente FPGA's Xilinx Virtex-2 Pro com o processador PowerPC 405 embutido, e o ProtoFlex (CHUNG, 2006), um simulador funcional híbrido para acelerador de FPGA que suporta o processador SPARC V9 e plataformas X86.

Um trabalho correlato e bastante pertinente a nossa concepção de arquitetura de coerência de *cache* é o projeto TSAR (TSAR, 2011), que descreve uma arquitetura multinúcleo de memória compartilhada, escalável, suportando coerência de *cache*. É um MPSoC que contém milhares de processadores RISC-32 bits.

Pensando no ambiente de simulação de MPSoC, apresentaremos os conceitos referentes ao tipo de simulação, podendo ser consideradas *bit-accurate* e *cycle-accurate*.

A simulação *bit-accurate* é normalmente utilizada na concepção de *hardwares* modernos, essencial para a verificação funcional de algoritmos complexos. Um modelo RTL padrão para descrição de *hardware* é muito lento para simulação de *software* diretamente em um SoC [*System on Chip*], por isso que se torna imprescindível um simulador funcional que apresente detalhes abstratos do *hardware*. Concluindo, esse tipo de simulação descreve a funcionalidade do *hardware*, não se preocupando com algumas características (*cache, pipeline*) que computam o tempo de acesso (desempenho). O uso dessa simulação é interessante se o objetivo for a descrição do novo componente de *hardware*, a Tabela de Padrões de Acesso.

Normalmente, plataformas virtuais permitem algumas atividades chaves como: exploração do sistema de arquitetura, antecipação de atividades de verificação e desenvolvimento de *software* pré-silício. Podemos citar como exemplo,

a plataforma virtual SoClib (SoClib, 2011) que oferece um simulador *bit-accurate* chamado SystemCASS.

Outra plataforma importante na literatura é o SimSoc (SimSoC, 2011).

Já uma simulação "*cycle-accurate*" simula uma microarquitetura, podendo conceber novos microprocessadores, testando e executando *benchmarks* (com sistemas operacionais completos ou compiladores). Todas as operações de predição de desvios, *caches misses* etc. são executadas no simulador. Podemos citar como exemplos de simuladores o SIMICS (FEDOROVA, 2007) e o GEMS (RUPNOW, 2010).

2.3 Consistência de Dados

Nas seções anteriores, discutimos a respeito da diferença entre *cache* e diretório de coerência de *cache* e apresentamos o conceito de informação de coerência.

Para entendermos melhor o tópico da consistência de dados, apresentamos o seguinte contexto: milhares de cópias dos mesmos dados podem estar distribuídos entre os processadores no sistema de memória compartilhada.

Os endereços são armazenados no diretório de coerência de *cache* de maneira que, quando há requisições de leitura/escrita para esses dados armazenados em *cache*, o protocolo de *cache* troca mensagens entre os processadores. A seguir, demonstramos uma transação de escrita do dado e invalidação das múltiplas cópias deste.

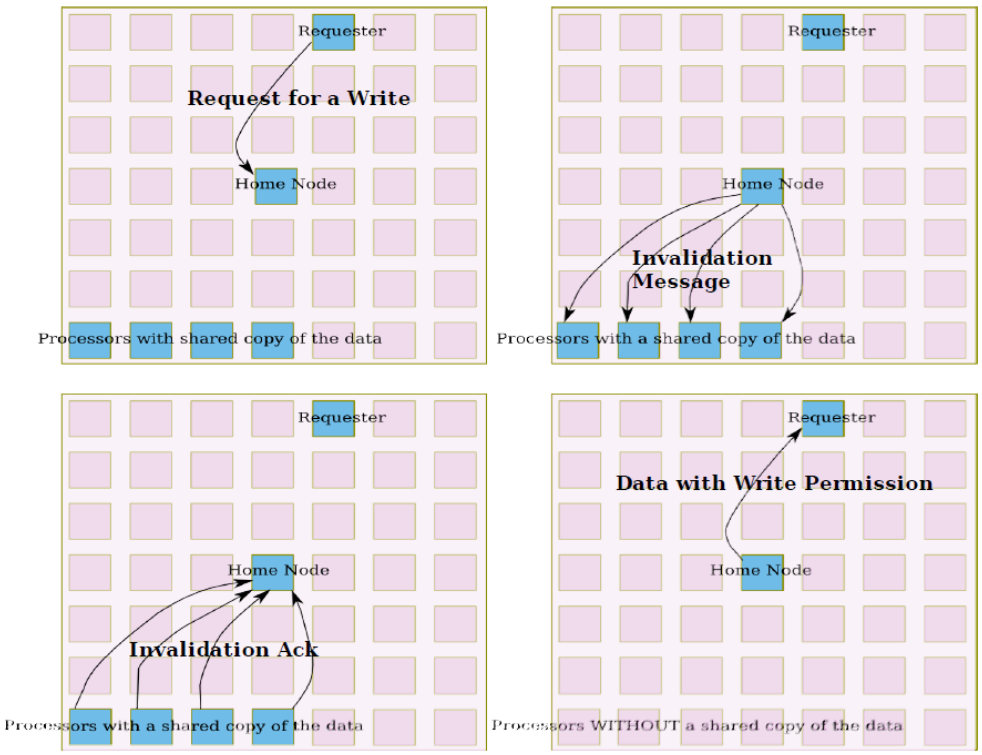


Figura 11: Transação de Escrita em dado armazenado em 4 *cores* (KOFUJI, 2010)

2.3.1 Método de Busca do *Home Node* de Dados

Quando um processador necessita de um dado, primeiro ele checa com o "*home node*" do dado requisitado o estado de coerência em que ele se encontra. Esta tarefa é compartilhada com o processo de coerência que analisa todas as mensagens enviadas/recebidas para o acesso à memória compartilhada.

O primeiro passo para a requisição de dado de escrita é localizar o *home node* do dado requisitado. Cada protocolo de coerência de *cache* implementa um método de acordo com as características arquiteturais. Podemos citar o método *round-robin* para a busca do *home node* em uma rede em anel descrito na implementação do protocolo de coerência *Ring-Order's round-robin* (MARTHY, 2008).

Na tese, utiliza-se o método *round-robin* para alocação de *home node* a um dado requisitado. O método *Round-Robin* é desempenhado pela função módulo, em que os últimos 4 bits do endereço base determinam o *home node* do dado.

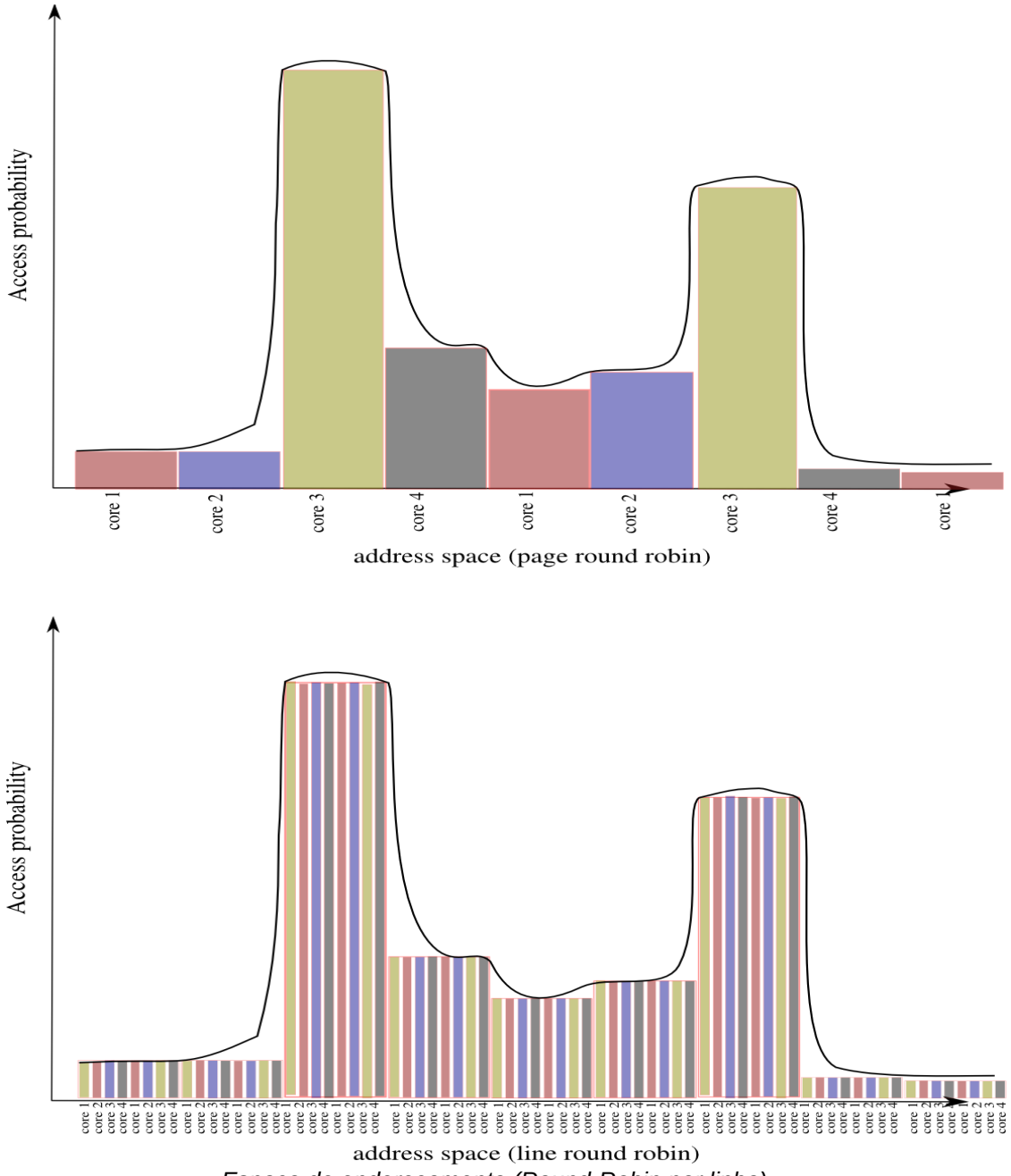
Esse algoritmo *round-robin* tem inúmeras vantagens: simples e eficiente, considerando acessos à memória compartilhada, ele distribui as requisições entre os processadores e oferece uma boa utilização da largura de banda.

A seguir, através da fórmula representa-se a identificação do home node segundo determinados parâmetros.

Home Node Id = (Endereço na linha de cache) %(Nb cores); (2)

A figura 12, apresenta a probabilidade de acesso aos processadores do sistema, enquanto *home node,* pela granularidade de página de memória. No gráfico (a), observamos dois picos de página de memória em requisições ao mesmo *home node*. Podemos chamar esse fenômeno de “*hot home*”, o que significa uma saturação de mensagens de coerência por um mesmo *core* do sistema.

Na figura 12 (b), observa-se a probabilidade de acesso pela granularidade mais fina (linha de *cache*). Neste exemplo, observamos que o fenômeno de *hot home*, contudo possui pontos de saturação de mensagens de coerência distribuídos em diversos *cores* do sistema.



Espaço de endereçamento (Round-Robin por linha)

Figura 12: Método Round-robin pela granularidade (KOFUJI, 2010)

2.4 Síntese do capítulo

O capítulo de estado da arte propôs uma análise detalhada da literatura, embasada nos tópicos de otimizações de arquiteturas, orientada em padrões de acesso, base de microarquitetura das arquiteturas CMP com memória compartilhada, expondo detalhes da arquitetura *baseline* e exemplificando um circuito integrado com múltiplas funções e, finalmente, os conceitos de coerência de dados.

A partir desse estado da arte, podemos considerar algumas conclusões para o nosso método de coerência de *cache* baseado em *hardware*/*software*.

Na literatura, apresentamos todos os modelos de consistência de dados:

Consistência fraca é dividida em *realese* e *entry*. O modelo de consistência a ser adotado para nossa arquitetura de coerência de *cache* será o *entry*.

Consistência forte: escrita, sequencial, linear, casual, FIFO.

Depois de definido o modelo de consistência de dados, apresentamos uma arquitetura CMP padrão inspirada no modelo *baseline*, que expõe um protocolo de coerência de *cache* baseado em diretório. A partir dessa arquitetura, será proposta a arquitetura de coerência de *cache* na próxima capítulo.

No estudo da literatura, a análise do fluxo de transações de um protocolo de coerência de *cache* baseado no protocolo *baseline* foi importante para conceitar o processo de localização do dado em cache.

O estado da arte foi essencial para abordar os conceitos fundamentais da tese e também investigar, na literatura, métodos de otimização de aplicações que abordem padrões de acesso à memória e possíveis soluções em *hardware* e *software* que façam uso do tratamento de padrões de acesso à memória.

Capítulo 3 – Arquitetura de Coerência de *Cache* Otimizada a Padrões de Acesso à Memória

Neste capítulo, apresentos uma proposta de arquitetura de coerência de *cache* otimizada a padrões de acesso à memória e uma proposta de protocolo híbrido de coerência.

O método de coerência de *cache* baseado em *hardware* é uma proposição de arquitetura CMP de memória compartilhada otimizada a padrões de acesso à memória, ou seja, um *hardware* especializado que otimiza a execução de um arquivo de dados que contém padrões de acesso e informações de localidade espacial. Esses arquivos armazenam estruturas regulares de padrões de acesso à memória que são extraídos através de técnicas de *profiling*, por análise estática ou dinâmica de código.

Por exemplo, a partir do *benchmark* NPB (*NAS Parallel Benchmarks)* ou algoritmo LU, pode ser analisado esse comportamento de acesso à memória. Por meio de uma análise estática de código, gera-se o código binário interpretado pelo *hardware*. Os métodos de extração de padrões de acesso à memória são um tópico bem pertinente à concepção da nossa arquitetura de coerência de *cache*, porém não considerados como proposta da Tese.

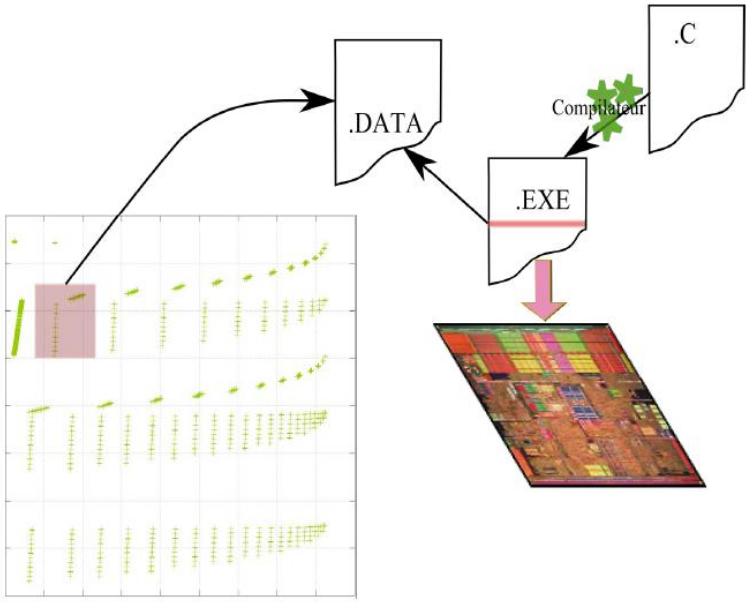


Figura 13: Otimização de padrões de acesso pelo *Hardware (KOFUJI, 2010)*

No tópico a seguir, sobre a proposição de Arquitetura de coerência de *cache*, será apresentado um modelo de hierarquia de *cache* que expõe a concepção de um novo componente de *hardware* chamado Tabela de padrões, destinado ao armazenamento de padrões. Além desse modelo de hierarquia, está prevista a apresentação de um algoritmo que descreve o fluxo de mensagens de um protocolo de coerência de *cache*, o módulo de memória (Tabela de padrões) e o método de implementação do comportamento desse componente.

O método de coerência de *cache* baseado em *software* é uma proposição de um protocolo híbrido baseado no protocolo *MESI* modificado. A arquitetura de coerência de *cache* permite-nos otimizar o tempo de execução de transações de mensagens no sistema de coerência de *cache*, como também o tempo de acesso ao *cache* por especulação de mensagens.

Um protocolo de coerência híbrido é definido pela alternância de mensagens segundo o método round-robin de escolha de home node. Um protocolo MESI modificado, por exemplo, em caso de "*miss* em cache", ele envia uma mensagem

simples de requisição de um determinado endereço. No caso do protocolo híbrido, ele atribui a escolha do home node ao terceiro nível de busca de endereço em hardware. Em caso de “hit de tabela de padrões”, ele envia uma mensagem especulativa a home node híbrido. Uma messagem especulativa é composta de um padrão de endereços.

Alguns tópicos são discutidos na proposição do protocolo hibrido: característica e definição do protocolo hibrido, conceito fundamental do protocolo *MESI* modificado *baseline*, método *round-robin* aplicado à escolha de um núcleo de processamento (*home node*), algoritmo *round-robin*, modelo de transações de leitura/escrita de mensagens, política de *cache write-back* e método de implementação do protocolo hibrido.

3.1 Arquitetura de Coerência de *Cache* baseado em Padrões

3.1.1 Proposição de Arquitetura de Coerência de *Cache*

A Arquitetura de Coerência de *Cache* proposta apresenta uma hierarquia de *cache* particular, com um novo componente de *hardware*, especialmente concebido para o tratamento de padrões de acesso à memória, em forma de tabela para armazenar padrões regulares de acesso.

Os componentes do sistema de hierarquia de *cache* são baseados em uma rede de interconexão do tipo *mesh* para permitir maior desempenho do *cache,* mesmo que haja aumento do número de processadores.

Cada processador do sistema apresenta: *cache* L1I (instruções) / *cache* L1D (dados) privado, *cache* L2 compartilhado, diretório de coerência de *cache*, conforme visto anteriormente na figura 08, sendo que nossa Arquitetura de Coerência de *Cache* apresenta esse novo componente de *hardware*: tabela de padrões.

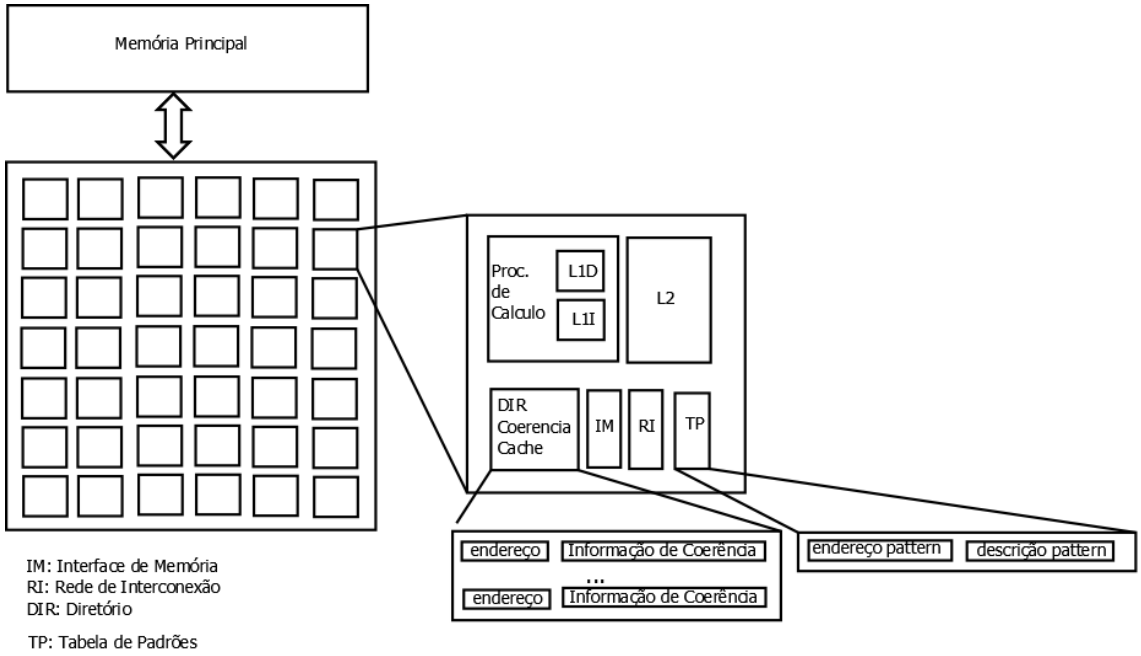


Figura 14: Proposição da Arquitetura de Coerência de *Cache*

A hierarquia de memória da nossa Arquitetura de Coerência de *Cache* é composta de *caches* de tamanho muito pequeno, em que o dado é localizado prioritariamente em *cache* L2.

Diferentemente, as novas gerações de arquitetura representadas pelos processadores AMDK10 (AMD, 2011), Intel Nehalem (BARKER, 2008) utilizam hierarquia de memória a três níveis de *cache* (L1/L2/L3). Na literatura, podemos destacar o processador AMD Athlon64 (AMD, 2011), por exemplo, que apresenta dois níveis de *caches* especializados (*cache* L1 de instrução, *cache* L1 associado ao TLB, *cache* L1 de dados e *cache* L2).

Em um sistema de coerência de *cache* tradicional baseado no protocolo *baseline*, descrito anteriormente, apresentamos alguns pivôs do sistema, como nó que requisita o dado (*requester*), nó que detém informação de coerência (*home node*) e o nó que compartilha o dado próximo ao nó que o requisita (*sharer*). Nesse sistema de coerência de *cache* tradicional, centralizamos as requisições. A nossa proposição de arquitetura de coerência de *cache*, a partir do nosso componente de

hardware (tabela de padrões), apresenta dois *home nodes*, os quais diferenciamos como mensagens *baseline* e mensagens especulativas. O sistema de transação de mensagens é composto por quatro pivôs:

- Nó requisitante: busca o dado em *cache* L1/L2 e, no caso do *cache miss* e tabela de padrões *miss*, encaminha o pedido ao *home node*;
- *home node*: nó em questão que detém a informação de coerência do dado, podendo receber uma requisição de dado ou um padrão de acesso;
- compartilhado: detém uma cópia do dado em *cache*. Essa cópia está em modo compartilhado, podendo existir milhares de cópias distribuídas nos diversos *caches* dos *cores* do sistema;
- proprietário: tem a cópia do dado em *cache*. Essa cópia está em modo exclusivo ou modificado, de modo que existe somente uma cópia.

3.1.2 Pseudocódigo de busca de dados em *cache*

Considerando a hierarquia de memória, a figura 15 apresenta um pseudocódigo que descreve o comportamento para busca de dados em primeiro nível. Cada *core* apresentará o mesmo comportamento de busca de dados em *cache* L1/L2 e tabela de padrões.

1. CPU Core carrega a instrução para leitura busca do endereço @x

- L1/L2 cache busca o endereço @x

Se cache hit:

- dados são lidos do cache e a operação de load instruction

END

else / cache miss */*

Busca de Tabela de padrões (TP look-up)

Se tabela de padrões hit

tp_home=get_home_node_RR_page(@x)

send RD_RQ_SPEC(padrao_@x) to tp_home

END @x process

else / tabela de padrao miss */*

Baseline_home= get_home_node_RR_line(@x)

send RD_RQ(@x) to baseline_home

END @x process

Figura 15: Pseudocódigo de busca de dados em memória com a tabela de padrões

3.1.3 Tabela de padrões

A tabela de padrões é um componente *hardware* especializado ao tratamento de padrões de acesso à memória, normalmente orientado a aplicações que apresentem otimização de código utilizando unidade *stride*.

Considerando que o tamanho do *cache* é limitado, quando consideramos, por exemplo, um algoritmo de multiplicação de matrizes com unidade *stride*, otimizamos o desempenho de padrões de acesso.

Para conceber um componente de *hardware* orientado a padrões de acesso, a funcionalidade da Tabela de Padrões é semelhante a uma tabela *hash*, que armazena padrões de acesso do tipo *stride*. Isso significa que a base de dados já foi processada e extraídos os padrões de acesso da aplicação devidamente armazenada na tabela de padrões.

São duas as estratégias para otimizar o desempenho: o formato do padrão de acesso baseado em *stride* e o método de acesso à tabela de padrões para o uso do protocolo especulativo.

Portanto, podemos considerar o componente de *hardware* (PT) como um acelerador de processador de identificação e uso de padrões de acesso à memória. O novo componente torna-se acelerador quando otimiza a performance diretamente em hardware o padrão de acesso à memória.

Em um primeiro momento, apresentamos um formato de padrão de acesso simples composto de um endereço base, tamanho e *stride*. A seguir, um pequeno exemplo do formato do padrão de acesso:

Função do Formato de padrão de acesso:

(*offset*, tamanho, *stride*) @ -> @LIST (3)

Offset	Tamanho	*Stride*

Podemos instanciar os elementos do padrão de acesso:

{1, 4, 2} dado @1 + 1 = @2; (4)

O endereço base (*offset* ou base@) indica o primeiro endereço de memória do padrão de acesso, o tamanho (*length*) indica o número de elementos (endereços) do padrão de acesso e o *stride* indica a distância entre dois endereços consecutivos.

Conforme observado no formato de padrão de acesso, a tabela de padrões é similar a uma tabela *hash*, compondo uma chave que identifica cada padrão de acesso. Na figura 16, apresentamos a tabela de padrões:

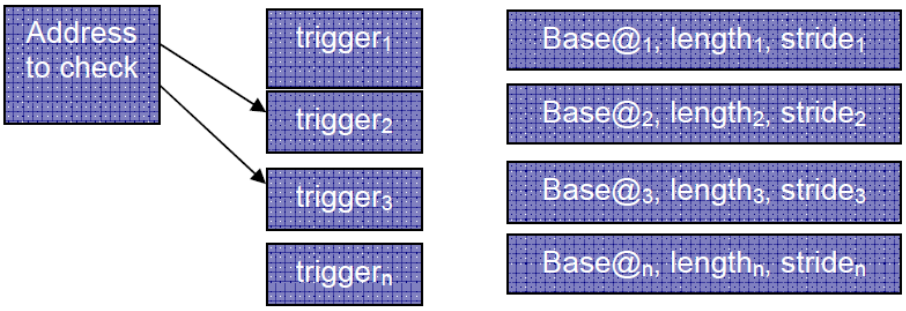


Figura 16: Tabela de padrões de acesso (KOFUJI, et al., 2010)

Onde:

- Address to check: é uma operação de busca de endereço;
- Trigger: é uma tabela unidemencional que armazena as chaves que identificam os padrões;
- Os elementos do padrão de acesso são composto por: Base@ (endereço offset), lengh (tamanho do *pattern*), *stride* (avanço / passo).

O método otimizado de arquitetura de coerência de *cache* proposto nesta tese apresenta como objeto um sistema de múltiplos *cores* embarcados, interconectados por meio de um sistema de hierarquia de cache que permite aos *cores* do sistema comunicarem-se entre si, sendo cada *core* composto de um processador, níveis L1/L2 de *cache*, a tabela de padrões e uma memória principal.

A partir dessa tabela de padrões, concluímos que esse componente de *hardware* armazena uma sequência de endereços de memória em cada padrão. A tabela em si apresenta diversos de padrões identificados pelas chaves.

A partir da tabela de padrão de acesso, podemos definir a seguinte função de descrição de padrão:

$$Desc = f(EdB, T, S) \qquad (5)$$

Onde:

f( ) representa uma função do padrão de acesso que contém uma sequência de endereços, iniciando pelo endereço de base *EdB*, procedendo *T* endereços espaçados de S endereços;

Desc representa o descritor do padrão de acesso que resulta em um conjunto de endereços a partir da aplicação determinada na função ***f()*** com os parâmetros apresentados, ***EdB*****,** ***T*****,** ***S*****;**

EdB representa o endereço de base, endereço correspondente ao primeiro endereço de um conjunto de endereços pertencentes ao padrão de acesso de dados em uma busca de dado (endereço);

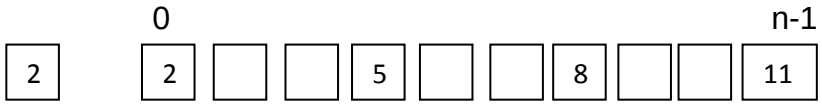
T é o número inteiro correspondente ao número de palavras binárias apresentadas na busca de dados;

S corresponde ao espaçamento que permite avançar um endereço de uma palavra binária ao endereço da palavra binária seguinte pertencente ao padrão de acesso.

Quando definimos a instanciação do padrão de acesso:

{1,4,2} dado @1 + 1 = 2 (6);

Representamos a busca de endereço quando temos um *cache miss* do endereço @2, exemplificado da seguinte forma:



Aplicando o endereço @1, retornamos o seguinte descritor:

Desc = {2, 5, 8, 11} *(7)*

3.1.4 Contribuições da Tabela de Padrões

A contribuição principal da tabela de padrões está fundamentada no método de acesso a padrões baseado em formato *stride*, criando uma modificação no protocolo *baseline* (MESI modificado), por exemplo, a fim de que seja possível beneficiar-se do mecanismo de acesso a padrões e predição de dados por meio da transação de mensagens padrão (*patterns*).

Sem o componente de *hardware*, tabela de padrões, o protocolo *baseline* envia várias mensagens tradicionais ou ditas *baseline* para a requisição de um único endereço, a cada mensagem enviada.

Adicionando esta tabela de padrões a cada *core* do sistema, possibilitamos a otimização do tempo de acesso a padrões, além da implementação de um protocolo especulativo baseado em transação de mensagens especulativas (*patterns*). Portanto, a tabela de padrões (padrões de acesso à memória) contribui para a redução do número de transações de mensagens, otimizando, dessa forma, o protocolo de coerência de dados. A figura 17, apresenta-se duas abordagens: padrões (*pattern*) e *baseline*.

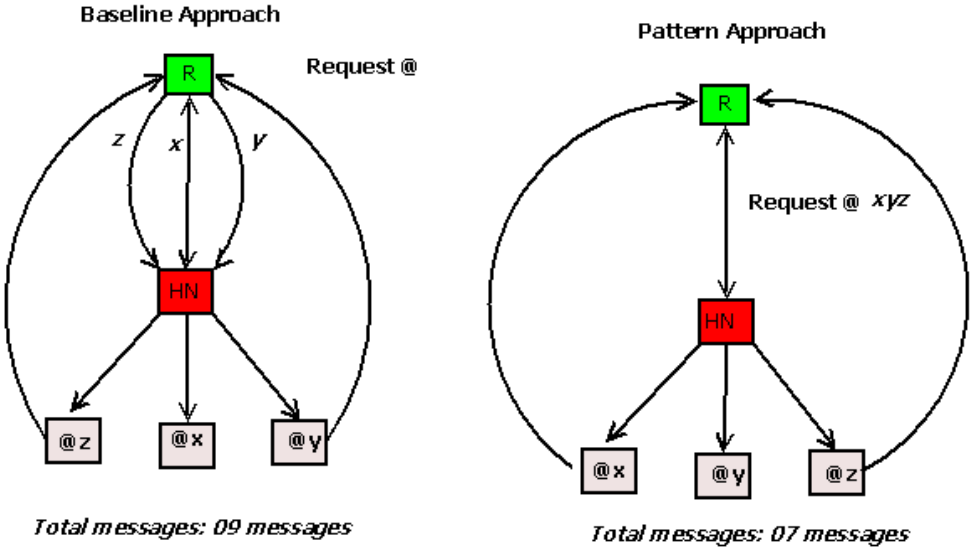


Figura 17: Comparação entre duas abordagens: *baseline* & padrão (KOFUJI, 2011)

Na figura 17, apresentamos dois cenários: abordagem *baseline*, em que o nó requisitante envia de maneira sequencial os endereços de x, y, z, de modo que o total de mensagens seja 9; e a abordagem *pattern*, em que o nó requisitante envia um padrão de acesso com um conjunto de endereços (xyz), totalizando 7 mensagens.

3.1.6 Método

O ideal seria desenvolver um modelo de descrição de *hardware* desse novo componente, mas inicialmente será desenvolvida uma implementação simples, que descreva o funcionamento da Tabela de padrões, uma biblioteca que poderá ser adicionada à chamada do simulador. A linguagem utilizada foi a linguagem C.

3.2 Protocolo de Coerência Híbrido

A partir do componente de *hardware* (Tabela de padrões), propomos o modelo do protocolo de coerência de *cache* híbrido orientado a padrão de acesso.

O protocolo de coerência híbrido foi desenvolvido a partir desse componente de *hardware* da Arquitetura que foi proposta, otimiza o tráfego de coerência de *cache* do sistema, sempre que exista um padrão.

O novo modelo de protocolo de coerência híbrido foi baseado na especificação de tratamento de diferenciação de tráfego de mensagens do sistema de consistência de dados. A partir desta especificação, introduz-se o conceito de granularidade de mensagens para evitar o congestionamento de mensagens no sistema.

A concepção deste protocolo de coerência híbrido apresenta as seguintes características:

- Diferenciação entre mensagens *baseline* e especulativas;
- mensagens especulativas que permitem a leitura de todos os endereços de um padrão (*patterns*) a partir do endereço de base;
- envio de mensagens especulativas por granularidade de página;
- técnica de *round-robin* para escolha do *home node*.

A implementação do protocolo híbrido apresenta uma transação de mensagens de acordo com o tipo de mensagem. A seguir, analisaremos alguns detalhes da técnica *round-robin*, que define a diferenciação de mensagens segundo a granularidade.

3.2.1 Técnica de *Round-Robin*

O componente de *hardware* (tabela de padrões) em cada *core* do sistema nos permite distribuir dois fluxos de mensagens para o acesso aos dados. O primeiro fluxo é referenciado por acesso aos dados que não são referenciados na tabela de padrões. Nesse caso, introduzimos a função de determinação do *home node* quando temos um *miss* de tabela de padrões:

Home node = (endereço / tamanho da linha de cache) % número de cores; *(8)*

O algoritmo *Round-Robin* baseia-se na granularidade do tamanho de linha de *cache*. O segundo fluxo que é gerado pelos acessos quando o endereço requisitado é encontrado na tabela de padrões, o que gera o resultado *hit* de tabela de padrões:

Home node = (endereço / tamanho da página) % número de cores; *(9)*

Essas funções representam dois níveis na hierarquia do *home node*. Representaremos a função *Round-Robin* para escolha do *home node* usando algoritmo de multiplicação de matrizes.

Na figura 18, inicialmente, apresentaremos um trecho do código do nosso programa para calcular o *home node* a partir dos endereços de memória de uma matriz M x N.

Trecho do código da figura 18 exemplifica a função *home node*:

```
#include <stdio.h>
#include <stdlib.h>
#define NB_CORE_IN_THE_SYSTEM 64
#define PAGE_SIZE 4096
#define LINE_SIZE 128
#define ROUNDROBIN_PAGE_GRANULARITY 1
#define ROUNDROBIN_LINE_GRANULARITY 2
/* This function returns the home node corresponding to an address  */
int get_home_id (void* an_address, int nb_core_in_the_system, int HN_POLICY)
{
int home_node_id;
 switch (HN_POLICY)
   {
   case ROUNDROBIN_LINE_GRANULARITY:
       home_node_id = ((long) an_address / LINE_SIZE % nb_core_in_the_system);
           break;
   case ROUNDROBIN_PAGE_GRANULARITY :
       home_node_id = ((long) an_address / PAGE_SIZE % nb_core_in_the_system);
       break;
   default: /* centralized home node */
           home_node_id = 0;
       break;
}return home_node_id;
}
```

Figura 18: Função para escolha do *Home Node*

Analisaremos o programa address_generator que calcula o *home node* para os diferentes elementos da matriz. O usuário pode inserir os seguintes dados: altura

(M) e largura (N) da matriz e o tipo de granularidade para escolha do *home node* (linha ou página).

Por meio dos comandos de compilação e execução do programa:

jussara@debian:~/programme/cocca$ gcc address_generator.c -std=c99 -o address_gen.exe

jussara@debian:~/programme/cocca$ ./address_gen.exe

This program computes the home node id for every elements of a NxM matrix
Please enter the (N) height of the matrix:
-->4
Please enter the (M) width of the matrix:
-->4
Please select the home node allocation policy:
Default behavior is centralized coherency (home node always set to 0)
Otherwise, please select:
1) Page granularity.
2) Line granularity.
-->1
Policy set to page round robin
Matrix to allocate is 4X4
my_matrice[0][0] address 0xc3c010 value 0.50 home node 60
my_matrice[0][1] address 0xc3c018 value 1.50 home node 60
my_matrice[0][2] address 0xc3c020 value 2.50 home node 60
my_matrice[0][3] address 0xc3c028 value 3.50 home node 60
my_matrice[1][0] address 0xc3c030 value 4.50 home node 60
my_matrice[1][1] address 0xc3c038 value 5.50 home node 60
my_matrice[1][2] address 0xc3c040 value 6.50 home node 60
my_matrice[1][3] address 0xc3c048 value 7.50 home node 60
my_matrice[2][0] address 0xc3c050 value 8.50 home node 60
my_matrice[2][1] address 0xc3c058 value 9.50 home node 60
my_matrice[2][2] address 0xc3c060 value 10.50 home node 60
my_matrice[2][3] address 0xc3c068 value 11.50 home node 60
my_matrice[3][0] address 0xc3c070 value 12.50 home node 60
my_matrice[3][1] address 0xc3c078 value 13.50 home node 60
my_matrice[3][2] address 0xc3c080 value 14.50 home node 60
my_matrice[3][3] address 0xc3c088 value 15.50 home node 60

Figura 19: Execução do Programa Matriz pelos Endereços de Memória

3.2.2 Transação de Mensagens

No protocolo híbrido, três atores podem ser envolvidos: o *core* que requisita os dados (*requester*), o *baseline home node* e o *home node híbrido*. O *baseline home node* é o *core* indicado pela granularidade fina da fórmula *Round-Robin* ((endereço / tamanho de linha de *cache*) % número de *cores*), enquanto o *home node híbrido* apresenta uma granularidade grossa usando a fórmula *Round-Robin* ((endereço / tamanho de página) % número de *cores*).

Apresentaremos a transação de mensagens do protocolo híbrido para requisição de leitura durante os três estados: requisitante (*requester*), *baseline home node* e *home node* híbrido.

O primeiro modelo de árvore de decisão para transação de mensagens em uma requisição de leitura de dado é sempre condicionado à busca de dados na tabela de padrões - *TP lookup,* igualmente ao processo de *cache lookup* como demonstrado na figura 20 a seguir:

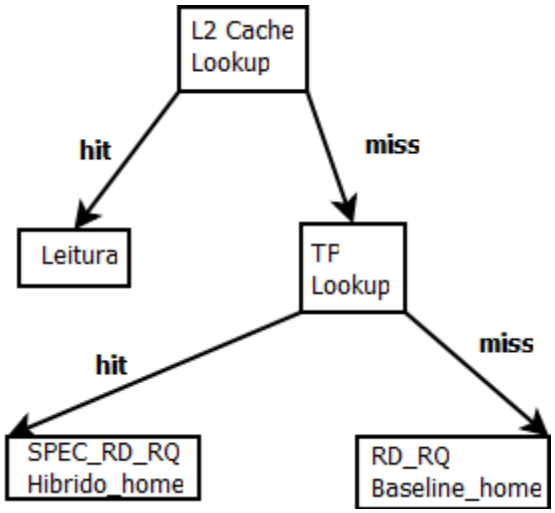


Figura 20: Requisição de Leitura: Árvore de decisão

A descrição do pseudocódigo para busca de dado em *cache* apresenta exatamente essa árvore de decisão. Considerando o caso de *hit* de tabela de padrões, enviamos uma mensagem especulativa (por granularidade de página). Ao identificarmos o indexador de um endereço na tabela de padrões, enviamos a requisição do padrão contido na tabela de padrões, composta de um conjunto de endereços do tipo *stride*.

A tabela de padrões é essencial para o protocolo de coerência híbrido, no sentido de que ela armazena os endereços extraídos do padrão *stride* de aplicações que apresentem padrões regulares. Quando temos um acesso *hit* na tabela de padrões, otimizamos a transação de mensagens, evitando o congestionamento de mensagens – *hotspot* e gerando um envio de mensagens por meio de um formato particular, o padrão – *pattern*.

Na figura 21, representamos a recepção desse padrão no *home node* híbrido, para o caso de uma operação de leitura.

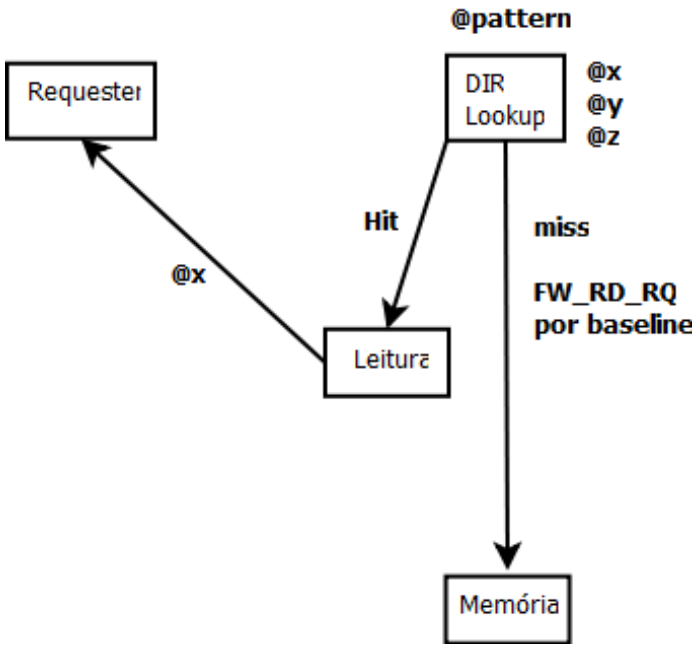


Figura 21: Leitura: *Home Node* Híbrido

O desempenho em termos de tempo de acesso à memória poderá ser medido por meio de três acessos importantes:

- Tempo de acesso ao diretório de coerência de *cache* (DIR);

- Tempo de acesso à tabela de padrões do requisitador (*requester*);

- Tempo de acesso ao DIR do *baseline home node*;

- Tempo de acesso à memória (pior caso);

O *baseline home node* recebe uma mensagem *baseline* de requisição por apenas um endereço, como apresentado na figura 22:

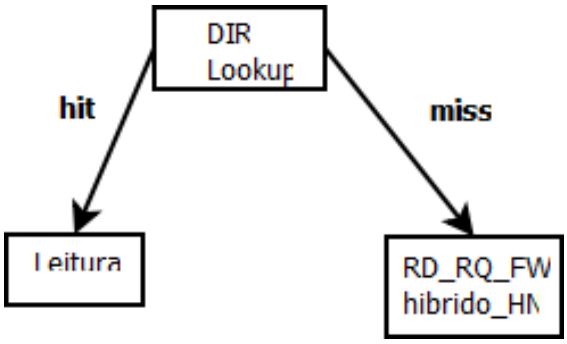


Figura 22: Árvore de decisão: *Baseline Home Node*

A árvore de decisão para o recebimento de mensagem *baseline* a partir do *miss* da tabela de padrões é representado no pseudocódigo da figura 23:

Recepção da requisição RD_RQ (@x);

DIR Lookup para o @x;

Se DIR hit:

- processo de execução de leitura do dado;

FIM processo @x;

Senão {DIR miss}

- Envio da requisição RD_RQ_FW(@x) - > Hibrido_HN;

FIM processo @x;

Figura 23: Árvore de decisão do *Baseline Home Node*

Na próxima seção, apresentaremos um modelo analítico de transação de mensagens do protocolo de coerência híbrido.

3.2.3 Modelo de Transação de Mensagens

O diagrama de transações de mensagens de leitura do protocolo de coerência híbrido apresenta três atores importantes, conforme discutido na seção anterior: Requisitador, *home node* híbrido, *baseline home node*. Para cada requisição de acesso de leitura, a primeira ação consiste na pesquisa do endereço na tabela de padrões. Se a pesquisa retorna uma entrada (endereço base de um padrão), se gera então, uma mensagem ao *home node* híbrido. Caso contrário, uma mensagem *baseline* (dita clássica) é enviada ao *Baseline home node*.

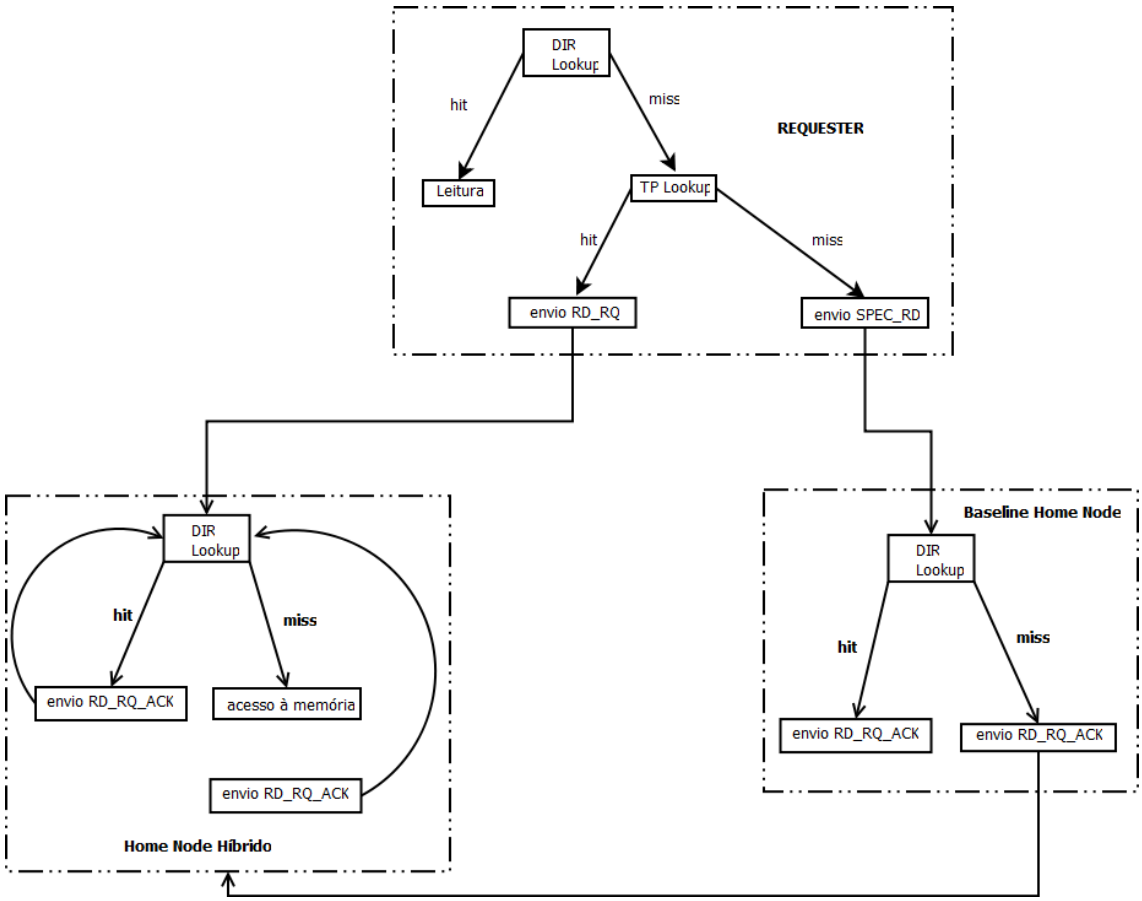


Figura 24: Modelo de Transação de Mensagens do Protocolo Híbrido

3.2.4 Método

O método adotado para validação do protocolo de coerência híbrido será o desenvolvimento de um modelo analítico que avalia o custo efetivo de desempenho do protocolo e compará-lo com o protocolo *baseline*. Através de parâmetros da hierarquia de cache temos uma fórmula de representação de avaliação do custo.

3.3 Síntese do capítulo

Neste capítulo, apresentamos uma arquitetura de coerência orientada a padrões de acesso.

A proposição dessa arquitetura é eminentemente baseada em arquiteturas CMP com memória compartilhada, mais especificamente na arquitetura *Baseline*, que apresenta um modelo de hierarquia de *cache* baseado em diretórios e dois níveis de cache (L1/L2).

A arquitetura *Baseline* é baseada em núcleos de processamento chamados *tiles* interconectados através de uma rede mesh, em que cada núcleo de processamento apresenta um chip, cache de instruções e de dados L1 privado e cache L2 privado, diretório de coerência e canal de memória.

A diferença fundamental em termos de conceitos para nossa arquitetura de coerência de *cache* é que ela se baseia em *caches* L1 privados e *caches* L2 compartilhados.

Genericamente, estamos tratando de uma arquitetura CMP de *cores* múltiplos com memória compartilhada em um único circuito integrado, o que chamamos de “MPSoC” e *caches* baseados em diretório.

Comparando nossa arquitetura com a nova geração de processadores, no que diz respeito ao modelo de hierarquia de memória, verificamos que modelos sofisticados de gestão de memória são baseados em múltiplos níveis de *cache*. A nossa arquitetura de coerência de *cache* apresenta um novo componente de memória auxiliar que compõe a hierarquia de *cache*, especializado para o armazenamento e tratamento de padrões de acesso à memória.

A partir do modelo de hierarquia de *cache,* apresentamos um algoritmo que descreve o processo de coerência de *cache,* fazendo uso da nossa arquitetura de coerência de *cache*. Nossa contribuição é baseada na especificação do fluxo de mensagens para checar se um dado requerido se encontra em *cache* e no método de acesso à tabela de padrões.

Uma das maiores contribuições em termos de concepção do novo componente de *hardware* “tabela de padrões” foi o método de tratamento de padrões de acesso à memória e a especificação deste componente em termos de formato do padrão. Essa tabela de padrões otimiza aplicações baseadas em padrões, acelerando o reconhecimento de padrão e realizando especulação de

mensagens, além de possibilitar a implementação de um protocolo híbrido que apresente mensagens clássicas *baseline* e mensagens especulativas.

Na proposição do protocolo híbrido, foi desenvolvido um método *round-robin* para gestão de escolha do *home node* baseado no protocolo *baseline*. A inovação baseada em *software* é o desenvolvimento do modelo do protocolo híbrido em que o *home node* pode ser referente a mensagens *baseline* ou referente a mensagens especulativas (baseado em padrões).

O modelo do protocolo apresenta um método de diferenciação de mensagens, descrevendo as transações de leitura/escrita baseadas na política de *cache*: *write/back*.

Capítulo 4 – Validação da Arquitetura e Protocolo

Neste capítulo, apresentaremos o método de arquitetura de coerência de *cache* otimizado a padrões de acesso regulares através do modelo da estrutura de *hardware*. Nossa contribuição principal será descrever a estrutura de *hardware* que representará a tabela de padrões. Em uma segunda etapa, será proposto como trabalho futuro, uma simulação de um modelo de linguagem de abstração de *hardware*. Para muitas arquiteturas e/ou processadores embarcados, utilizamos a simulação chamada *cycle accurate*. Em termos de validação da arquitetura e protocolo, optamos por utilizar uma simulação baseada em *performance/accurate*.

A simulação sequencial baseada em *cycle accurate* é muito lenta, além de se basear em modificações microarquiteturais referentes ao controlador de memória e não oferecer suporte à implementação de protocolo de coerência de *cache*, apenas a políticas de *cache* considerando questões de leitura/escrita.

O uso da simulação, atualmente, é essencial para experimentos relacionados à concepção de processadores ou tecnologias e a rápida exploração da arquitetura. O contexto de simulação aborda arquiteturas massivamente paralelas.

A arquitetura de alto nível vem se tornando mais importante que a microarquitetura e o foco da tese de doutorado, nesse requisito, é oferecer um estudo de coerência de *cache* e hierarquia de memória e especificação da estrutura do componente de hardware representado por uma linguagem de alto nível. Em um simulador normalmente, são analisados NoCs, acessos a DRAM, hierarquia de memória e coerência de *cache* e modelos de processadores.

Após analisar a literatura a respeito de simuladores de arquitetura, adotamos a direção de simuladores de *software* aberto, simuladores que possam simular componentes de hardware como um descritor de linguagem de hardware (HDL – Hardware Description Lannguage) ou software de emulação.

Em uma primeira abordagem a arquitetura de coerência de *cache* foi desenvolvida por meio de uma biblioteca API que descreve o comportamento do *hardware*. A implementação desta biblioteca utilizou linguagem C por ser uma linguagem de alto nível mais próximo da linguagem de máquina. Inicialmente, apresentaremos a implementação do módulo da API: representação de *patterns*.

A biblioteca é composta de dois módulos da API: representação de *patterns* (formato e inserção de elementos) e função de ativação (a qual compara a chave com o endereço base de cada *pattern*). Prevendo como trabalhos futuros, a biblioteca que descreve o comportamento do *hardware* poderá ser integrada a um módulo chamado *HAL – Hardware Abstraction Level*. A princípio, a biblioteca é uma função de chamada do comportamento de *hardware,* cuja integração está prevista no modelo de coerência de *cache* de um simulador como meta futura de trabalho.

A seguir apresentamos uma exposição do ambiente de simulação, descrevendo a previsão da configuração básica da arquitetura e dos processadores. Normalmente, a meta principal seria uma análise de desempenho, através do uso de *benchmarks* SPLASH-2 modificados e algoritmo de multiplicação de matrizes otimizado por *strides* visando verificar a implementação do protocolo *baseline* e, posteriormente, do protocolo híbrido.

Porém, o objetivo realizado durante o trabalho científico foi especificar o novo componente de hardware bem como modelar suas funções para otimizar um protocolo de coerência de cache.

4.1 Validação por simulação

Tanto os simuladores do tipo *cycle accurate* para HPC e *bit accurate* para SoC apresentam diversas plataformas de processadores comerciais para os dois tipos de propósitos, os quais não apresentam a tabela de padrões como componente de *hardware*.

Para isso, inicialmente representamos um primeiro modelo em linguagem de alto nível para descrever o comportamento do *hardware*. O ideal, em termos de desenvolvimento industrial, seria desenvolver um modelo de *Hardware Abstraction Layer - HAL* para compor o subsistema multicamada de *software* embarcado caracterizado por API de comunicação, API sistema operacional e HAL API. Quando limitamos a arquitetura do *hardware*, a plataforma API pode ser modelada pelo *Transaction level model – TLM*. Na seção de validação de componente de *hardware* apresentaremos o primeiro modelo e descreveremos algumas funções.

Em uma validação por simulação, adotaremos o simulador SimSoc, que utiliza dois tipos de processador padrão: processador ARM e processador PowerPC. Trataremos como referência a arquitetura *baseline*, definindo a configuração e a política de *cache*.

Tabela 1: Parâmetros de Arquitetura a serem estudados

Característica	**Valor**
Processador	ARM / PowerPC
***Clock* frequência**	3.16 GHz
L1 *cache*	privado
L2 *cache*	compartilhado
Rede	*Mesh*

4.2 Validação do componente de *hardware*

No que diz respeito ao primeiro modelo, descreveremos a estrutura do *hardware*: formato da tabela de padrões, formato dimensional composto de padrões encadeados, associados a identificadores que chamamos de chave de padrão (*trigger*).

Para iniciar a validação do componente de *hardware*, analisaremos primeiramente a estrutura em linguagem C.

- código que apresenta o tipo de estrutura do novo componente de hardware e suas funções para adicionar elementos.

A estrutura é representada na figura 25 pelo comando *typedef struct* e as seguintes estruturas: *pattern*, *keytable*.

```
typedef struct Pattern_ {
  /* sizeof(address) */
  long unsigned int capacity;
  /* address nr */
  long unsigned int size;
  /* offset pattern */
  long unsigned int * offset;
   /* lenght pattern */
  long unsigned int * lenght;
  /* stride pattern */
  long unsigned int * stride;

} Pattern_t;
```

Figura 25: Estrutura *Pattern*

Conforme representado no capítulo 3, a tabela de padrões é composta de padrões que chamamos de *pattern*. Dessa forma, apresentamos neste pequeno trecho de código a estrutura de um *pattern*. Observamos que cada um possui um formato único, composto por elementos e estritamente baseado em *stride*. A priori, os elementos chave que compõem um *pattern* são *offset*, *lenght* e *stride*. Nesta estrutura de *pattern,* adicionamos ainda sua capacidade e tamanho. A capacidade é representada como espaço de memoria para armazenar cada *pattern*. E, finalmente, o tamanho é o espaço que ele ocupa efetivamente na memória.

A figura 26, discute a tabela unidimensional que representa as chaves e os identificadores de cada *pattern*.

```
typedef struct KeyTableElement_ {

/* address */
  long unsigned int address;
/* pointer to pattern */
  Pattern_t * pattern;

}KeyTableElement_t;


typedef struct KeyTable_ {


  /* sizeof(address) */
  long unsigned int capacity;

  /* key nr */
  long unsigned int size;

  /* key table */
  long unsigned int * keytable;

} KeyTable_t;
```

Figura 26: Estrutura elemento chave tabela e chave tabela

A primeira estrutura, que chamamos de elemento da tabela de chaves (*KeyTableElement*), descreve os elementos da tabela chave: endereço (*address*) e ponteiro para o padrão (*pattern*);a segunda estrutura, que chamamos tabela chave (*KeyTable*), descreve os elementos: capacidade, tamanho, chave.

A figura 27, representamos as funções que descrevem a tabela de padrões:

```
Pattern_t *
patternNew() {

  Pattern_t * pattern = NULL;

  pattern = malloc(sizeof(Pattern_t));
  assert(pattern);

  pattern->capacity = _MEMORY_REALLOC_CAPACITY * sizeof(long unsigned int);
  pattern->size = 0;

  pattern->offset = malloc(pattern->capacity);
  assert(pattern->offset);
  pattern->lenght = malloc(pattern->capacity);
  assert(pattern->lenght);
  pattern->stride = malloc(pattern->capacity);
  assert(pattern->stride);

  return pattern;
}
```

Figura 27: Função Criação de *Pattern*

```
void
patternAddOffset(Pattern_t * pattern, long unsigned int offset) {

  assert(pattern);

  if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->offset = realloc(pattern->offset, pattern->capacity);
    assert(pattern->offset);
  }

  pattern->offset[pattern->size] = offset;
  pattern->size++;
}
```

Figura 28: Função Adicionar *Offset*

```
void
patternAddLenght(Pattern_t * pattern, long unsigned int lenght) {

  assert(pattern);

   if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->lenght = realloc(pattern->lenght, pattern->capacity);
    assert(pattern->lenght);
  }

  pattern->lenght[pattern->size] = lenght;
  pattern->size++;
}
```

Figura 29: Função Adicionar Tamanho

```
Void
patternAddStride(Pattern_t * pattern, long unsigned int stride) {

  assert(pattern);

   if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->stride = realloc(pattern->stride, pattern->capacity);
    assert(pattern->stride);
  }

  pattern->stride[pattern->size] = stride;
  pattern->size++;
}
```

Figura 30: Função Adicionar *Stride*

```
void
patternFree(Pattern_t ** pattern) {

  assert(*pattern);

  if ((*pattern)->capacity > 0)
    free((*pattern)->offset);

  free((*pattern));
  (*pattern) = NULL;
}
```

Figura 31: Função Inicializar *pattern*

```
void
patternPrint(Pattern_t * pattern) {

  long unsigned int i = 0;

  assert(pattern);

  for (i = 0; i < pattern->size; i++) {
    fprintf(stdout, "%lu        %lu       %lu\n", pattern->offset[i], pattern->lenght[i],
pattern->stride[i]);
}
 };
```

Figura 32: Função Imprimir *pattern*

```
KeyTable_t *
tableNew () {


  KeyTable_t * table = NULL;


  table= malloc(sizeof(KeyTable_t));
  assert(table);

  table->capacity = _MEMORY_REALLOC_CAPACITY * sizeof(long unsigned int);
  table->size = 0;
  table->keytable = malloc(table->capacity);

  return table;
}
```

Figura 33: Função Adicionar Tabela chave

```
void
tableAddKeytable(KeyTable_t * table, long unsigned int keytable) {

  assert(table);

  if ((sizeof(long unsigned int) * (table->size + 1)) > table->capacity) {
    table->capacity = sizeof(long unsigned int) * (table->size +
_MEMORY_REALLOC_CAPACITY);
    table->keytable = realloc(table->keytable, table->capacity);
    assert(table->keytable);
  }

  table->keytable[table->size] = keytable;
  table->size++;
}
```

Figura 34: Função Adicionar Chave Tabela

```
void
tablePrint(KeyTable_t * table) {

  long unsigned int i = 0;

  assert(table);

  fprintf(stdout, "%s", "{");

  for (i = 0; i < table->size; i++) {
    fprintf(stdout, "%lu ", table->keytable[i]);
  }

  fprintf(stdout, "%s", "}\n");

};
```

Figura 35: Função Imprimir Tabela

```
void
tableFree(KeyTable_t ** table) {

  assert(*table);

  if ((*table)->capacity > 0)
    free((*table)->keytable);

  free((*table));
  (*table) = NULL;

}
```

Figura 36: Função Limpar Tabela

A partir da execução desse algoritmo, obtemos a impressão da tabela de padrões, as estruturas criadas (tabela de padrões e tabela de chaves), conforme apresentado a seguir:

PATTERN TABLE

Offset	Lenght	Stride
3	0	0
5	0	0
7	0	0
9	0	0
0	4	0
0	4	0
0	4	0
0	4	0
0	0	2
0	0	2
0	0	2
0	0	2

Figura 37: Tabela de Padrões

KEY TABLE

Key

2
4
6
7

Figura 38: Table de Chaves

Nesse algoritmo, através das funções para *pattern table* e *key table*, adicionamos os elementos e, depois, imprimimos os resultados das tabelas. Na primeira tabela, adicionamos elementos do campo do *pattern*: *offset*, *lenght* e *stride*. Por representar os endereços em um *pattern*, apresentamos outro algoritmo simples, que imprime uma matriz M x N, em que cada célula representa um *pattern* composto de quatro endereços hexadecimais.

```
jussara@debian:~/src/jussara/dev/cocca$ ./pattern0.c
This programm print the address of pattern cocca table like the elements of a NxM matrix
Please enter the (N) height of the table:
-->4
Please enter the (M) width of the table:
-->4
tablecocca[0][0]={0x9d09008 0x9d09008 0x9d09008 0x9d09008}
tablecocca[0][1]={0x9d09010 0x9d09010 0x9d09010 0x9d09010}
tablecocca[0][2]={0x9d09018 0x9d09018 0x9d09018 0x9d09018}
tablecocca[0][3]={0x9d09020 0x9d09020 0x9d09020 0x9d09020}
tablecocca[1][0]={0x9d09008 0x9d09008 0x9d09008 0x9d09008}
tablecocca[1][1]={0x9d09010 0x9d09010 0x9d09010 0x9d09010}
tablecocca[1][2]={0x9d09018 0x9d09018 0x9d09018 0x9d09018}
tablecocca[1][3]={0x9d09020 0x9d09020 0x9d09020 0x9d09020}
tablecocca[2][0]={0x9d09008 0x9d09008 0x9d09008 0x9d09008}
tablecocca[2][1]={0x9d09010 0x9d09010 0x9d09010 0x9d09010}
tablecocca[2][2]={0x9d09018 0x9d09018 0x9d09018 0x9d09018}
tablecocca[2][3]={0x9d09020 0x9d09020 0x9d09020 0x9d09020}
tablecocca[3][0]={0x9d09008 0x9d09008 0x9d09008 0x9d09008}
tablecocca[3][1]={0x9d09010 0x9d09010 0x9d09010 0x9d09010}
tablecocca[3][2]={0x9d09018 0x9d09018 0x9d09018 0x9d09018}
tablecocca[3][3]={0x9d09020 0x9d09020 0x9d09020 0x9d09020}
```

Figura 39: Impressão de endereços da Tabela de Padrões

4.3 Validação do Protocolo Híbrido por Modelo Analítico

Para validar o protocolo híbrido, apresentaremos um modelo analítico simples, com intuito de avaliar o custo eficaz do desempenho do protocolo de coerência de *cache* híbrido e compará-lo com o protocolo *baseline*.

O custo de desempenho é avaliado por meio de pseudocódigos de diferentes tipos de transações em um protocolo de coerência. Utilizando um modelo padrão, simplificamos um modelo analítico para a Arquitetura de Coerência de *Cache* otimizada a padrões regulares. O modelo avalia cinco parâmetros importantes que pressupõem um custo de desempenho durante o processo de busca de dado em memória, como apresentado a seguir:

- *L2 lookup*: custo de inspeção do *cache* L2;

- *DirLookup*: custo de inspeção do cache diretório de coerência de *cache*;

-- *PTLookup*: custo de inspeção de tabela de padrões;

- *Mem*: custo de acesso à memória;

- *Msg*: custo de transferência de mensagens entre dois processadores do sistema (nesse parâmetro, podemos considerar a distância de Manhattan).

A cada inspeção de memória, obtemos um *hit* ou *miss* de *cache*. Dessa forma, verificamos o custo de desempenho por meio de:

%hitL2: fração do número total de acesso ao *cache* L2 em caso de *hit* de *cache* L2.

%missL2: 1 -%hitL2.

%hitDir: fração do número total de acesso ao diretório de coerência.

%missDir: 1 - %hitDir.

%hitPT: fração do número total de acesso a tabela de padrões em caso de *hit* de PT.

~Pattern_lengh:

Conforme descrito na seção 2.1.2, analisamos o algoritmo de imagem integral para mensurar a taxa %*miss*L2 pelo custo de transação de leitura (em ciclos) expresso pelo protocolo híbrido (de nome *cocca*) em comparação ao protocolo *baseline*.

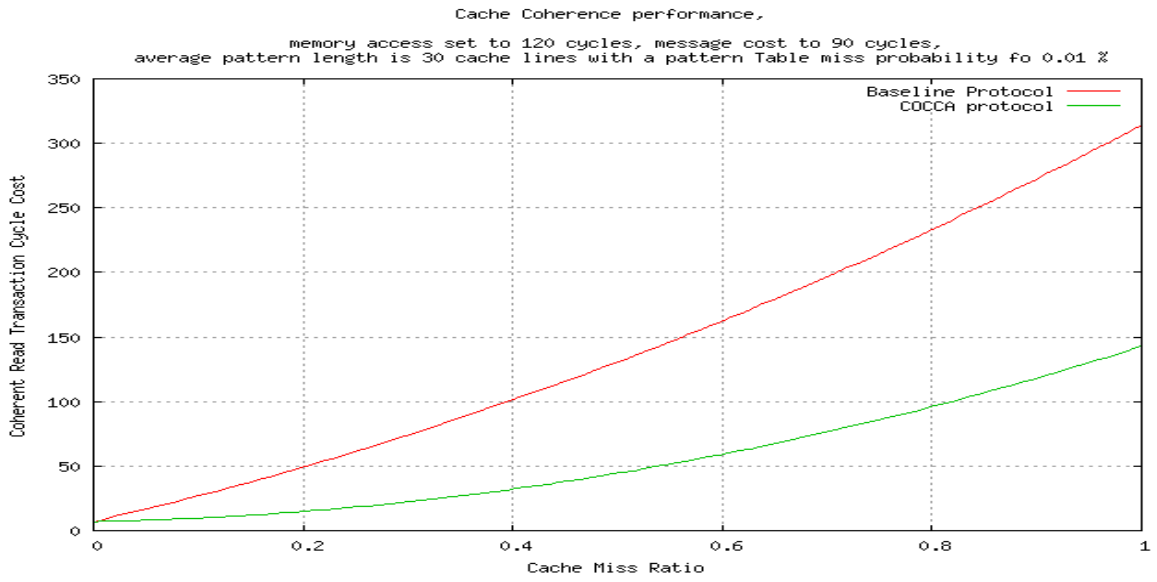


Figura 40: Taxa de *Miss* em comparação ao custo de transação de leitura (KOFUJI, 2011)

No custo de transação de leitura em função da taxa de *cache miss*, observamos o acesso à memória configurado a 120 ciclos, custo de mensagens a 90 ciclos e média de tamanho de *pattern* aferido a 30 linhas de *cache*. A aplicação quando apresenta padrões regulares, o protocolo híbrido torna-se bem mais performante que o protocolo *baseline*, em termos de número de mensagens necessárias para manter a coerência de cache e portanto, estima-se um melhor desempenho no tempo de execução de aplicação.

Tabela 2: Custo de desempenho dos protocolos de coerência

Cache Miss	Custo de transação de leitura (Híbrido vs. *Baseline*.	
0.2	20	50
0.4	40	100
0.6	60	165
0.8	98	240
1	148	310

Notamos que o desempenho do protocolo *baseline* não se apresenta da mesma maneira. Mais precisamente, a frequência de *cache miss* é maior que com o protocolo híbrido devido à predição de dados via tabela de padrões.

Recordando o modelo de transações do protocolo híbrido, concluímos que o custo do acesso ao serviço em caso de *cache hit* não possui valor, todas as mensagens da rede possuem o mesmo custo de msg. Consideramos que o diretório de coerência de *cache* e o *cache* possuem o mesmo tamanho. Portanto, consideramos que eles possuem custos de desempenho (L2Lookup e DirLookup) idênticos.

Como exemplo, em um acesso simples de leitura à memória compartilhada, podemos representar a seguinte equação, para determinar seu custo.

(2) L2Lookup + %missL2 *(msg + DirLookup + (%DirHit * msg) +

(%missDir * (msg + mem)));

Simplificando os parâmetros dessa equação, obtemos:

(3) (1 + %miss)Lookup + %miss*(2msg + %miss*mem);

Podemos ainda escrever essa equação pela forma polinomial:

(4) $Lookup + (Lookup + 2msg) * \%miss + miss * \%miss^2$

Posteriormente à avaliação do modelo analítico, validamos o protocolo híbrido quanto à eficiência do custo de desempenho do protocolo de coerência. Como trabalho futuro, propomos a implementação do protocolo híbrido e sua simulação em um simulador *cycle-accurate*.

Capítulo 5 – Conclusão

Analisando a proposição da Tese, o objetivo geral de desenvolver um método de arquitetura de coerência de *cache* orientada a padrões de acesso regulares para sistemas embarcados foi atingido com sucesso. A arquitetura de coerência de *cache* desenvolvida no trabalho, introduz um novo componente de *hardware* que nos orienta em direção a uma separação de tráfego de coerência em dois fluxos distintos (funcionais e especulativos) de coerência distribuída. Esses fluxos são orientados de acordo com a tabela de padrões (componente de *hardware*). Em um modelo analítico de fluxo de transação de mensagens de um protocolo de coerência, enfatizamos dois atores importantes (*Home Node* Híbrido e *Baseline Home Node*) segundo a diferenciação de mensagens:

- um tráfego funcional (*baseline*) necessário à coerência da memória, direcionado ao primeiro nível de *home node*, *baseline home node*;

- um tráfego especulativo que administra o segundo nível de *home node*, *home node* híbrido;

A tabela de padrões possibilita o armazenamento de padrões regulares, a otimização das aplicações que consideram padrões de acesso à memória e a redução do custo do silício, evitando acessos à memória principal e possibilitando especulação de mensagens no mesmo *chip*. A redução do custo do protocolo em sicilicio foi demonstrado pela equação de desempenho do protocolo híbrido.

O método de Arquitetura de Coerência de *Cache* apresentado nesta tese de doutorado uma primeira contribuição importante: a proposição de uma arquitetura CMP de memória compartilhada (*cache* baseado em diretório).

Vimos que essa arquitetura CMP é caracterizada por *tiles* e apresenta os seguintes níveis de cache:

- *Cache* L1 privado;

- *Cache* L2 compartilhado;

Concluímos que a nossa arquitetura de coerência de *cache* é baseada na referência da arquitetura *baseline,* igualmente composta por *tiles* e *cache* baseado em diretórios, com a diferença no tipo de *cache* L2 compartilhado entre os processadores – *tiles*. Já na arquitetura *baseline*, observamos uma característica de *cache* distribuído, porém privado a cada processador.

O modelo de hierarquia de novas gerações de processadores como Intel Nehalem e AMD K10 apresenta três níveis de *cache* (L1/L2/L3) e, às vezes, *caches* especializados. Nosso método de arquitetura de coerência de *cache*, por outro lado, mantém os dois níveis de *cache* tradicionais e acrescenta um componente de *hardware* auxiliar, a tabela de padrões, que reconhece os endereços requisitados através do endereço *offset* de cada padrão (*pattern*).

Concluímos que esse novo modelo de hierarquia de *cache* otimiza, de maneira considerável, além de aumentar o desempenho de sistemas embarcados quando se reduz o tempo de execução da transação de mensagens do protocolo de coerência de *cache*.

Uma das contribuições da Tese, relacionada diretamente ao modelo de hierarquia de *cache*, é a especificação de busca de dados no novo modelo de hierarquia.

Em relação à tabela de padrões, podemos citar as principais contribuições:

- Estrutura (semelhante a uma tabela *hash*);

- Formato padrão (baseado em *stride*);

- Modelo de pesquisa

As contribuições parciais relacionada com a tabela de padrões são:

- Componente de *hardware* (tabela de padrões) como acelerador de padrões de acesso à memória;

- Formato padrão (*pattern*) e instanciação de elementos;

- Especificação formal da função de descrição de padrão;

- Redução do número de mensagens na abordagem *pattern* (evitando *hotspot*).

Outra contribuição importante se refere à validação do componente de *hardware* por meio do módulo de representação de *pattern* da biblioteca API, implementada em C. Nossa arquitetura de coerência de *cache* é validada inicialmente, descrevendo em detalhes a estrutura do componente de *hardware* e o formato da tabela de padrões que será posteriormente incorporado a um modelo de *hardware* de um simulador.

O protocolo híbrido complementa a especificação técnica da arquitetura de coerência de *cache*, no sentido em que otimiza o tráfego de transação de mensagens do protocolo de coerência de *cache* (*baseline*).

Concluímos que o nosso protocolo híbrido é baseado nas mensagens clássicas do protocolo *baseline* e apresenta um método de coerência de *cache* que nos permite o gerenciamento e tratamento de mensagens especulativas (envio de *pattern* – conjunto de endereços indexados).

As principais contribuições desse protocolo são:

- Diferenciação de mensagens clássicas e especulativas a partir da verificação da tabela de padrões;

- Mensagem especulativa que permite leitura de todos os endereços de *pattern* a partir do endereço base;

- Envio de mensagem especulativa por granularidade de página;

- Técnica de *Round-Robin* para escolha do *Home Node*.

As contribuições pertinentes à validação da técnica de *home node* são apresentadas por meio de representação formal de fórmulas para diferenciação de mensagens e escolha do *home node*.

Concluímos que a técnica de escolha do *home node* pode ser validada e implementada por meio da implementação da técnica de *round-robin*; apresentamos um algoritmo de matriz para imprimir os elementos (endereços) de *pattern* em forma hexadecimal apontado a um *home node* especifico.

Outra contribuição importante refere-se à especificação de pseudocódigos para a representação da transação de mensagem em caso de *cache miss* e modelos de árvores de decisão do home node que requisita, *baseline home node* e *home node* híbrido.

Ao final, apresentamos a especificação completa, um modelo geral de transação de mensagens do protocolo.

Inicialmente, a validação do protocolo híbrido é realizada por um modelo analítico que avalia o custo de desempenho do protocolo de coerência de *cache*. O custo efetivo é aferido por meio de parâmetros de valor de certos elementos da hierarquia de *cache,* como a busca em *cache* L1/L2/Dir e na tabela de padrões. Podemos dizer que nosso modelo analítico é especializado e caracterizado pelo componente de *hardware*.

5.1 Trabalhos Futuros

Inicialmente, apresentamos um código que descreve a estrutura do novo componente de *hardware* (tabela de padrões) e suas funções. Foram definidas as seguintes estruturas: *pattern* e *keytable*; e funções: criar *pattern* e adicionar (*offset, lenght, stride*), inicializar e imprimir; criar *keytable*, inicializar e imprimir.

Esse código faz parte do módulo da biblioteca apresentada a seguir:

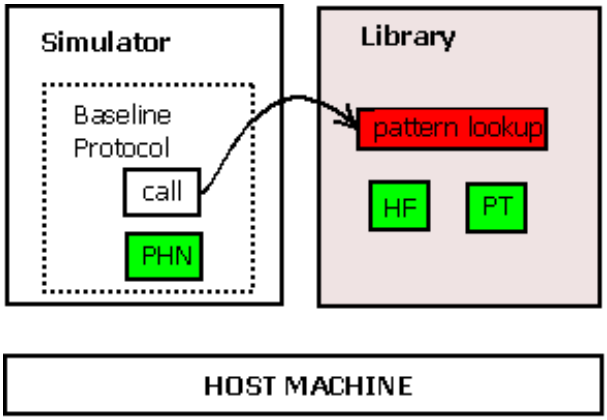


Figura 38: Modelo de Simulação a partir da Biblioteca (KOFUJI, 2011)

Observe que o primeiro módulo, chamado Tabela de padrão (PT) e Função de Ativação, foi implementado para representar a estrutura e as funções do novo componente de *hardware*. Como trabalhos futuros para essa abordagem de simulação *cycle-accurate*, implementaremos a função (*pattern lookup*) que realiza a técnica de busca na tabela de padrões.

A partir dessa API, a biblioteca pode ser chamada ou invocada a partir do modelo de protocolo de coerência de *cache* de simulador. Outro módulo a ser tratado refere-se ao modelo de *hardware* do simulador.

Para esse tipo de simulação, seria ideal um simulador paralelo e distribuído para exploração da concepção de processadores como o simulador de *software* aberto do MIT, o simulador Graphite baseado em uma ferramenta de *profiling* (MILLER, 2010) e a otimização de código da Intel (*pintools*) que também gere o *trace* dos acessos à memória.

No entanto, pensando-se em termos de sistemas embarcados, o módulo da API pode ser utilizado para compor uma linguagem de abstração de *hardware* (*Hardware Abstraction Language – HAL*). A vantagem de se utilizar um modelo HAL é que podemos acessar os recursos da microarquitetura mais facilmente.

Como proposta futura de trabalho técnico, o desenvolvimento do modelo HAL é interessante, porque podemos aplicá-lo em sistemas de *hardware*, como, por

exemplo, FPGA’s, ou mesmo protótipos de processadores embarcados comerciais. Para isso, é mais aconselhável o uso inicial de simuladores de *bit-accurate*. Pensamos em usar o simulador SimSoc.

Em termos de *benchmark* e análise de desempenho, esperamos apresentar a primeira análise (*profiling*) do código de multiplicação de matrizes e escolha do *home node* (técnica de *round-robin*), avaliando a questão do impacto de desempenho do *cache*.

O pacote a ser utilizado para teste em sistemas embarcados será o SPLASH-2 modificado (PRAMOD, 2011).

Referências

AMD**. Revision guide for AMD family 10th processor**. [s.l.]: Advanced Micro Devices, publ. 41322, ref. 3.82, feb 2011.

ASANOVIC, Krste et al**. The landscape of parallel computing research: a view from Berkely**, EECS Department, University of California, Berkeley, 2006. Technical Report, UCB/EECS-2006-183.

_____ . Disponível em:

http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

Acesso em: Julho 2011.

BARKER, Kevin et al. **A performance evaluation of the nehalem Quad-core Processor for Scientfic Computing.** Parallel Processing Letters, v.18, Dec 2008.

CHANG, C et al. **BEE2: the high-end reconfigurable computing system**. In: Design & Test of computers, IEEE, v.22, n.2, p.114-125, 2005.

CHUNG, E. S. et al. **ProtoFlex: co-simulation for component-wise FPGA Emulator Development.** In: 2nd Workshop on Architecture Research using FPGA Platforms (WARFP'06), 2006.

FEDEROVA, Alexandra. **Operating System Scheduling for Chip Multithreaded Processors**. Cambridge, Massachusetts, PhD Thesis, Harvard University, 2006.

FEDEROVA, Alexandra. **Simics x86 Target Guide**. Technical Report, 2007.

KISACANIN, B. **Integral image optimizations for embedded vision applications.** In: Symposium on Image Analysis and Interpretation - SSIAI'08, Santa Fe, New Mexico. p. 181-184, 2008.

HANDY, Jim. **Cache Memory Book**. 2nd Edition Morgan Kaufman Series in Computer Architecture and Design, 1998.

HMPP. Disponível em: http://www.openhmpp.org. Acesso em: 2011.

HIRAMATSU, Kenji et al. **FaceID: Benchmarks for Face Detecting Algorithms Using Cell Processor**. Poster presentation, 2nd Annual Carleton Cell BE Programming Workshop. University of Ottawa, 2009.

K. Rupnow et al. **Accurately Evaluating Application Performance in Simulated Hybrid Multi-Tasking Systems.** Accepted for publication at ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2010.

Intel. **Intel Core Duo Processors**. Disponível em: http://www.intel.com/products/processor/coreduo/. Acesso 2011.

Intel. **Intel Quick Patch Interconnect**. White Paper, 2009. Disponível em:

http://www.intel.com/technology/quickpath. Acesso em 2011.

JEFFERY A. **Proximity-aware Directory-based Coherence for Multicore Processor Architectures.** In: ACM Symposium on Parallel Algorithms and Architectures - SPAA'07, San Diego, California: Proceedings of SPAA'07, p. 126-134, 2007.

KOFUJI, Jussara. **Co-Designed Cache Coherency Architecture for Embedded Multi-core Systems**. Paris, França: Apresentação, 2011.

KOFUJI Jussara e CUDENNEC Loic. **Trafic de coherence optimisé par co-design aplique aux architectures multinúcleos embarquée.** CEA-LIST, Saclay, 2010. Memorial técnico, LIST/DACLE/2010-0583/S-JK, 2010.

KOFUJI, Jussara. **Traffic de coherence optimisé par co-design, aplique aux architectures multicoeur embarquée**. Challenge Innovation (DRT), Grenoble, 2010. Relatório técnico, 2010.

KOFUJI, Jussara et al. **Tproposta de um elenco de disciplinas de pos-graduação para formação de arquiteturas avançadas de computadores.**. Capítulo de livro

aceito para revista eletrônica sobre Educação em Arquitetura de Computadores, 2010.

KOFUJI, Jussara et al. **Programando Multicore com IBM full-System Simulator --- CELL Broadband Engine**. Minicurso apresentado no Worshop de Sistemas Computacionais de Alto Desempenho – WSCAD2007, Gramado, 2007.

KOWALTOWSKI, Tomasz. **Von Neumann: suas contribuições à computação** . Estudos Avançados, v.10, São Paulo,1996.

LI, K. **Memory Coherence in Shared Virtual Memory Systems**. In: ACM Transactions Computer Systems, ACM Transactions Computer System, 1989. Revista 4.v.7, p 321-359.

MACK, C.A. **Fifty Years of Moore's Law.** Transactions on Semiconductors, Vol.24, p.202-207, May 2011.

MARANDOLA, Jussara, CUDENNEC, Loic. **Co-Designed Cache Coherency Architecture for Embedded Multicore Systems**. IPSoC 2011, Grenoble.

MARTHY,Michael. **Cache Coherence Techniques for Multicore Processors** .Wisconsin, Madson, PhD thesis, University of Wisnconsin, 2008.

MATTES, Leonardo et al. **Finite-Difference Time-Domain on cluster of Cell BE Processor.** Poster presentation, 2nd Annual Carleton Cell BE Programming Workshop. University of Ottawa, 2009.

MILLER, E. Johson et al. **FGraphite: A Distributed Parallel Simulator for Multicores.** PThe 16th EEE International Symposium on High-Performance Computer Architecture (HPCA), Jan 2010.

OLUKOTUM, Kunle. **Chip Multiprocessor Architecture: Technics to improve the Throughput and Latency**. Morgan & Claypool Publishers, 2007.p. 145.

PRAMOD, Joisha et al. **A Technique for the Effective and Automatic Reuse of Classical Compiler Optimizations on Multithreaded Code**. 38th ACM Symposium on Principles on Programming Languages, Jan 26 Austin, 2011.

Power4. **Power4 System microarchitecture**. Disponível em: http://03.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html. Acesso em 2010.

SCARPINIO, Mathew. **Programming Cell Processor for Games, Graphics and Computation**. Foreward by Dr. Duc Vianny (Technic Solution Architect, IBM), 2008.

SimSoC. Disponível em: http://formes.asia/cms/software/simsoc. Acesso em: 2011.

SoClib Disponível em: http://www.soclib.fr/trac/dev. Acessado em: 2010.

STEPHEN, Somogyi et al. **Spatial memory streaming**. IEEE International Symposium on Computer Architecture - ISCA'06, 2006, pp. 252-263.

STEPHEN, Somogyi et al. **Spatio-temporal memory streaming**. IEEE International Symposium on Computer Architecture - ISCA'09, 2009, pp. 69-80.

THOMAS, F. et al. **Temporal streaming of shared memory**. IEEE International Symposium on Computer Architecture - ISCA'05. – 2005, pp. 222-233.

TSAR **TSAR Project**. Disponível em: https://www-soc.lip6.fr/trac/tsar. Acesso em: 2010.

ZHUO, H. et al. **Alternative Home: Balancing Distributed CMP Coherence Directory**. Workshop on Chip Multiprocessing Memory Systems and Interconnections, 2008, Beijing, China.

APENDICES A - Algoritmos

A1 – Multiplicação de Matrizes para Escolha do Home Node

```
// Programa Escolha do Home Node

#include <stdio.h>
#include <stdlib.h>

#define NB_CORE_IN_THE_SYSTEM 64

#define PAGE_SIZE 4096
#define LINE_SIZE 128

#define ROUNDROBIN_PAGE_GRANULARITY 1
#define ROUNDROBIN_LINE_GRANULARITY 2

/*
 * This function returns the home node corresponding to an address
 */

int get_home_id (void* an_address, int nb_core_in_the_system, int HN_POLICY)
{
int home_node_id;


 switch (HN_POLICY)
    {
    case ROUNDROBIN_LINE_GRANULARITY:
        home_node_id = ((long) an_address / LINE_SIZE % nb_core_in_the_system);
          break;
    case ROUNDROBIN_PAGE_GRANULARITY :
        home_node_id = ((long) an_address / PAGE_SIZE % nb_core_in_the_system);
        break;
    default: /* centralized home node */
          home_node_id = 0;
        break;
    }
return home_node_id;

}



/*
 * Function to initialize the matrice with simple floating point value
 */
void initialization(double * my_matrice, int NB_LINE, int NB_COLUMN)
{
```

```
int i, j;
double elt_id=0.5;
for(i=0; i < NB_LINE ; i++)
   {
   for(j=0 ; j < NB_COLUMN ; j++)
      {
      my_matrice[i*NB_COLUMN + j]=elt_id;
      elt_id++;
      }
    }
}

/*
 * Main part of the program. This program generates a log file with the address and
the values of a matrice
 *
 */
void main()
 {
 int home_node_id;
 char HN_POLICY;
 double* my_matrice;
 FILE * LOG_FILE;
 int NB_LINE;
 int NB_COLUMN;
 int policy;

 printf("This programm computes the home node id for every elements of a NxM
matrix \n");

 printf("Please enter the (N) height of the matrix:\n-->");
 scanf("%d", &NB_LINE);
 printf("Please enter the (M) width of the matrix:\n-->");
 scanf("%d", &NB_COLUMN);

 printf("Please select the home node allocation policy: \n");
 printf("Default behavior is centralized coherency (home node always set to 0)\n
Otherwise, please select:\n");
 printf("%d) Page granularity. \n", ROUNDROBIN_PAGE_GRANULARITY );
 printf("%d) Line granularity. \n-->", ROUNDROBIN_LINE_GRANULARITY);
 scanf("%d", &policy);

 switch(policy)
    {
    case ROUNDROBIN_PAGE_GRANULARITY: printf ("Policy set to page round
robin \n"); printf("Matrix to allocate is %dX%d\n", NB_LINE, NB_COLUMN);
                    break;
    case ROUNDROBIN_LINE_GRANULARITY: printf ("Policy set to line round robin
\n"); printf("Matrix to allocate is %dX%d\n", NB_LINE, NB_COLUMN);
                    break;
```

```
    default: printf ("Policy set to centralized home node \n"); printf("Matrix to allocate is
%dX%d\n", NB_LINE, NB_COLUMN);
                        break;
   }
 my_matrice=(double *) malloc (NB_LINE * NB_COLUMN*sizeof(double));
/*
 * Openning a log file with the name my_address_log_file
 */
LOG_FILE= fopen("my_address_log_file", "w");

if (LOG_FILE== NULL)
   {
   printf ("CRITICAL ERROR: unable to create log file \n");
   exit(0);
   }

/*
 * Initialization of the matrice
 */
initialization(my_matrice, NB_LINE, NB_COLUMN);

/*
 *
 *
 */
 for(int i=0; i < NB_LINE ; i++)
    {
    for(int j=0 ; j < NB_COLUMN ; j++)
        {
        home_node_id= get_home_id(&my_matrice[i*NB_COLUMN + j] ,
NB_CORE_IN_THE_SYSTEM, policy);
        printf ("my_matrice[%d][%d] address %p value %.2f home node %d
\n",i,j,&my_matrice[i*NB_COLUMN + j], my_matrice[i*NB_COLUMN + j],
home_node_id);
        fprintf (LOG_FILE, "%d %d %p %.2f \n", i,j, &my_matrice[i*NB_COLUMN + j],
my_matrice[i*NB_COLUMN + j], home_node_id);
       }
     }
fclose(LOG_FILE);

printf("\n-- End of address_gen -- \n");
/*
 * end of program
 */
 }
```

A2 – Impressão de Padrões Regulares através da Estrutura de Tabela de Padrões

```
// Programa de Impressão de padrões da estrutura tabela de padrões

/* assert */
#include <assert.h>

/* fprintf */
#include <stdio.h>

/* malloc */
#include <stdlib.h>
/* string */
#include <string.h>


#define _MEMORY_REALLOC_CAPACITY 128

typedef struct Pattern_ {

  /* sizeof(address) */
  long unsigned int capacity;

  /* address nr */
  long unsigned int size;

  /* offset pattern */
  long unsigned int * offset;

  /* lenght pattern */
  long unsigned int * lenght;

  /* stride pattern */
  long unsigned int * stride;

} Pattern_t;


typedef struct KeyTableElement_ {

/* address */
  long unsigned int address;
/* pointer to pattern */
  Pattern_t * pattern;

}KeyTableElement_t;
```

```
typedef struct KeyTable_ {

  /* sizeof(address) */
  long unsigned int capacity;

  /* key nr */
  long unsigned int size;

  /* key table */
  long unsigned int * keytable;

} KeyTable_t;


Pattern_t *
patternNew() {

  Pattern_t * pattern = NULL;

  pattern = malloc(sizeof(Pattern_t));
  assert(pattern);

  pattern->capacity = _MEMORY_REALLOC_CAPACITY * sizeof(long unsigned int);
  pattern->size = 0;

  pattern->offset = malloc(pattern->capacity);
  assert(pattern->offset);
  pattern->lenght = malloc(pattern->capacity);
  assert(pattern->lenght);
  pattern->stride = malloc(pattern->capacity);
  assert(pattern->stride);

  return pattern;

}


void
patternAddOffset(Pattern_t * pattern, long unsigned int offset) {

  assert(pattern);

  if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->offset = realloc(pattern->offset, pattern->capacity);
    assert(pattern->offset);
  }
```

```
  pattern->offset[pattern->size] = offset;
  pattern->size++;

}

void
patternAddLenght(Pattern_t * pattern, long unsigned int lenght) {

  assert(pattern);

   if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->lenght = realloc(pattern->lenght, pattern->capacity);
    assert(pattern->lenght);
  }

  pattern->lenght[pattern->size] = lenght;
  pattern->size++;

}

void
patternAddStride(Pattern_t * pattern, long unsigned int stride) {

  assert(pattern);

   if ((sizeof(long unsigned int) * (pattern->size + 1)) > pattern->capacity) {
    pattern->capacity = sizeof(long unsigned int) * (pattern->size +
_MEMORY_REALLOC_CAPACITY);
    pattern->stride = realloc(pattern->stride, pattern->capacity);
    assert(pattern->stride);
  }

  pattern->stride[pattern->size] = stride;
  pattern->size++;

}

void
patternFree(Pattern_t ** pattern) {

  assert(*pattern);

  if ((*pattern)->capacity > 0)
    free((*pattern)->offset);

  free((*pattern));
  (*pattern) = NULL;
}
```

```
void
patternPrint(Pattern_t * pattern) {

  long unsigned int i = 0;

  assert(pattern);

  for (i = 0; i < pattern->size; i++) {
    fprintf(stdout, "%lu        %lu       %lu\n", pattern->offset[i], pattern->lenght[i],
pattern->stride[i]);
  }

 };

KeyTable_t *
tableNew () {


  KeyTable_t * table = NULL;


  table= malloc(sizeof(KeyTable_t));
  assert(table);

  table->capacity = _MEMORY_REALLOC_CAPACITY * sizeof(long unsigned int);
  table->size = 0;
  table->keytable = malloc(table->capacity);

  return table;

}

void
tableAddKeytable(KeyTable_t * table, long unsigned int keytable) {

  assert(table);

  if ((sizeof(long unsigned int) * (table->size + 1)) > table->capacity) {
    table->capacity = sizeof(long unsigned int) * (table->size +
_MEMORY_REALLOC_CAPACITY);
    table->keytable = realloc(table->keytable, table->capacity);
    assert(table->keytable);
  }

  table->keytable[table->size] = keytable;
  table->size++;

}
```

```
void
tablePrint(KeyTable_t * table) {

  long unsigned int i = 0;

  assert(table);

  fprintf(stdout, "%s", "{");

  for (i = 0; i < table->size; i++) {
    fprintf(stdout, "%lu ", table->keytable[i]);
  }

  fprintf(stdout, "%s", "}\n");

};

void
tableFree(KeyTable_t ** table) {

  assert(*table);

  if ((*table)->capacity > 0)
    free((*table)->keytable);

  free((*table));
  (*table) = NULL;

}

int
main (int argc, char *argv[]) {

  Pattern_t * pattern;

  assert(pattern);

  Pattern_t * myp = NULL;

  myp= patternNew();


/*. . . . .TABULATION COMPLETED AND PRINTING BEGINS. . . .*/

       printf("\n\n");
       printf("              PATTERN TABLE \n\n");
       printf("-----------------------------------------\n");
       printf("Offset  Lenght  Stride \n");
       printf("-----------------------------------------\n");
```

```
                patternAddOffset(myp, 3);
                patternAddLenght(myp, 4);
                patternAddStride(myp, 2);
                patternPrint(myp);
                patternFree(&myp);

        printf("-----------------------------------------\n");
    /*. . . . . . . . . PRINTING ENDS. . . . . . . . . . .*/
return 0;

}
```