Avaliação de protocolo de coêrencia de cache utilizando protótipos virtuais

Marcelo Schuck, Marcio Seiji Oyamada

Ciência da Computação – Universidade Estadual do Oeste do Paraná (UNIOESTE)
CEP 85819-110 – Cascavel – PR – Brasil

{marcelo.schuck, marcio.oyamada}@unioeste.br

Abstract. *The increasing complexity in the design of embedded systems calls for new tools and methodologies. Virtual platforms are proposed to provide an early simulation model of the system, easing the architecture evaluation in terms of performance, power consumption and others. The main purpose of a virtual prototype is to provide a global simulation model, and it is typically composed of processors, communication modules and memories. This work presents the implementation of a multiprocessor virtual prototype in SystemC with support for cache coherence using a snoop-based protocol. A case study of a JPEG decoder is used to evaluate the performance impact of the snoop cache coherence protocol. The results shown that the coherence protocol can increase the performance up to 69.18%, using an architecture with 8 processors.*

Resumo. *A crescente complexidade no projeto de sistemas embarcados requer que novos tipos de ferramentas e metodologias sejam desenvolvidos. Um exemplo disso são as plataformas virtuais, que são modelos de simulação visando facilitar a avaliação de arquiteturas quanto ao desempenho, consumo de potência entre outros requisitos de um sistema embarcado. O objetivo de um protótipo virtual é prover um modelo global de simulação, sendo tipicamente composto por processadores, módulos de comunicação e memórias. Assim, é possível avaliar de forma precisa a integração entre os componentes de hardware e software e o funcionamento do sistema como um todo. Este trabalho apresenta a implementação de um protótipo virtual utilizando SystemC de uma arquitetura nultiprocessada com suporte a coerência de cache utilizando um protocolo de rastreamento. Um estudo de caso de um decodificador JPEG paralelo é realizado visando avaliar o impacto no desempenho de um protocolo de coêrencia de cache baseado em rastreamento. Os resultados mostram que o protocolo de coêrencia podem aumentar o desempenho em até 69,18%, utililizando um arquitetura com 8 processadores.*

1. Introdução

Nos dias atuais é comum se deparar com aparelhos eletrônicos com algum processador embutido, denominados Sistemas Embarcados (SE). Estes dispositivos operam de forma semelhante a um computador de propósito geral, porém desempenham tarefas específicas. Segundo Marwedel (2006), um SE é um sistema de processamento de

informações que é incorporado em um produto maior e, normalmente, não é diretamente visível pelo usuário. Temos como exemplos de sistemas embarcados celulares, aeronaves, eletrodomésticos entre outros.

O desenvolvimento de hardware e software para SE possui algumas diferenças em relação ao desenvolvimento de sistemas computacionais de propósito geral. Ao se desenvolver um SE, deve-se pensar na arquitetura que será utilizada, pois a mesma apresenta requisitos restritos de desempenho, potência, custo de produção, design, tolerância a falhas e tempo de projeto.

A necessidade das empresas projetarem sistemas embarcados dentro de janelas de tempo cada vez mais reduzidas, satisfazendo às pressões mercadológicas, e a constante evolução tecnológica, obrigam o desenvolvimento de projetos otimizados (Wagner e Carro, 2003). Dentre os avanços tecnológicos de fabricação desses sistemas, destacam-se as soluções MPSoC (Multiprocessor System-on-Chip) (Jerraya, 2004). Estas provêem em um único chip, múltiplos processadores que podem ser incluídos juntamente com interfaces digitais, componentes de aplicação específica, memória, entre outros dispositivos.

Em uma arquitetura multiprocessada que utiliza uma memória compartilhada, uma forma de elevar o desempenho da arquitetura é a possibilidade de armazenar dados e instruções na memória cache. Porém, como cada processador tem uma cache, são necessários mecanismos para garantir a coerência. Métodos que garantem a consistência dos dados são denominados protocolos de coerência de cache (Patterson e Hennessy, 2002) e são classificados em:

- Rastreamento (Snooping): neste tipo de protocolo, a coerência é garantida através do rastreamento do barramento. Assim, cada processador fica responsável por verificar a atividade do barramento. Quando algum dado armazenado na cache local for alterado em outro processador, este dado é invalidado localmente;
- Diretório: neste tipo de protocolo, um módulo ligado ao barramento ou estrutura de comunicação, chamado de diretório, armazena os endereços da cache que estão mapeados nos diversos processadores do sistema. Em uma atualização, o diretório deve avisar os processadores que sua cópia local é inválida.

Para identificar potenciais problemas em um estágio inicial do projeto, ou seja, antes da finalização do hardware, a comunidade de EDA (*Electronic Design Automation*) propõe o uso de protótipos virtuais. Protótipos virtuais são modelos de simulação de sistemas completos (processadores, interfaces de E/S (Entrada e Saída), hierarquia de memória, etc). Com a simulação é possível analisar o desempenho obtido variando componentes, como memória, número e frequência dos processadores a serem utilizados em um MPSoC, eliminando assim a necessidade de construir prótotipos de hardware para avaliação de desempenho.

Este trabalho tem como objetivo apresentar a implementação de módulos para suportar a coerência de cache em uma plataforma virtual denominada VIPRO-MP (Garcia, 2009). Este trabalho está dividido da seguinte forma: na Seção 2 é descrito o protocolo de coerência MESI, e na Seção 3 o simulador VIPRO-MP e as alterações realizadas para suportar a coerência de cache são descritos. Na Seção 4 é apresentado o

estudo de caso de um decodificador JPEG para avaliar o impacto da coerência de cache, e a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Protocolos de coerência de cache

Para garantir que seja mantida a coerência nos dados das caches, um protocolo de coerência de cache deve ser utilizado. Para protocolos do tipo rastreamento, são propostos vários tipos de protocolos tais como MSI, MESI, MOESI e DRAGON (Patterson e Hennessy, 2002).

O protocolo MESI (Papamarcos e Patel, 1984) utilizado neste trabalho tem esse nome pois o bloco de dados na cache pode ter quatro estados: *modified*, *exclusive*, *shared* e *invalid*, sendo usado amplamente em multiprocessadores comerciais tais como Pentium e PowerPC. Utilizando este protocolo, a memória cache inclui dois bits de controle de estado por rótulo (*tag*), assim cada bloco da cache pode estar em um dos quatro estados propostos pelo protocolo.

O estado *modified* indica que o bloco da cache foi modificado, ou seja, está diferente da memória principal, portanto somente esta cache possui o bloco atualizado. O estado *exclusive* indica que somente esta cache possui o bloco e é igual ao bloco da memória principal. O estado *shared* indica que o bloco da cache está compartilhado por outras caches e está igual à memória principal. O estado *invalid* indica que o bloco da cache não é valido. Todos os estados são controlados a partir de sinais enviados ao barramento compartilhado entre os processadores. A Figura 1 representa o protocolo MESI, sendo que a máquina de estados da esquerda representa as transições do processador e a máquina de estados à direita o monitoramento do barramento (ações de outros processadores).

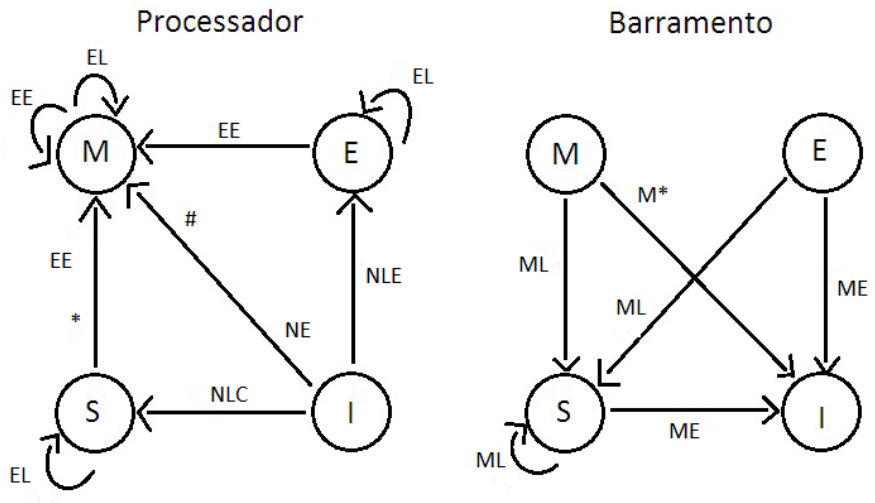

Figura 1: Diagrama protocolo MESI

As transições são as seguintes:

- EL: Indica que o processador está executando uma leitura sobre um bloco encontrado na cache local;
- *: É enviado um sinal de invalidação das demais cópias para o barramento;

- EE: Indica que o processador está executando uma escrita sobre o bloco encontrado na cache local;
- NE: Indica que o processador está executando uma escrita, porem não encontrou o bloco em sua cache;
- #: Indica que o processador está executando uma leitura no barramento com a intenção de modificação;
- NLE: Indica que o processador está executando uma operação de leitura, porém não encontrou o bloco em sua cache, e nenhuma outra cache o possui;
- NLC: Indica que o processador está executando uma operação de leitura, porém não encontra o dado em sua cache mas existe pelo menos uma outra cache no sistema que possui uma cópia deste bloco;
- ML: Indica que outro processador está requisitando um leitura sobre o bloco através do monitoramento do barramento;
- ME: Indica que outro processador está requisitando uma escrita sobre o bloco através do monitoramento do barramento;
- M*: Indica que outro processador está fazendo uma leitura com a intenção de modificar o bloco;

De acordo com as operações realizadas, os estados das linhas da cache são alterados, o que garante a consistência dos dados na ocorrência de multiplas cópias na memória cache.

3. VIPRO-MP

O VIPRO-MP (*Virtual Prototype for Multiprocessor Architectures*) é uma plataforma virtual implementada em SystemC (2011), sendo destinado à simulação de arquiteturas multiprocessadas (Garcia, 2009). O VIPRO-MP possibilita a simulação de ambientes multiprocessados, e possibilita a inclusão de novos módulos de *hardware* descritos em SystemC. Além disso, para cada arquitetura modelada é possível alterar o tamanho das caches, dados de configuração dos processadores e latência da memória. O VIPRO-MP foi desenvolvido baseado na ferramenta SimpleScalar [Burger e Austin, 1997]. Os parâmetros que configuram cada processador mantém a sintaxe do SimpleScalar, fato que facilita a utilização do VIPRO-MP.

Na organização de memória do VIPRO-MP, cada processador tem uma memória de dados e instruções privada. A comunicação entre os processadores é realizada através de um módulo de memória compartilhada, interconectado através de um barramento. A memória compartilhada é acessada através de uma faixa de endereços definida no simulador, sendo igual para todos os processadores. Na versão atual, todos os processadores executam em uma mesma frequencia. O SystemC suporta a modelagem de sistemas com multiplos relógios, no entanto haveria a necessidade do uso de pontes e adaptadores (*wrappers*) para conectar os componentes.

Na atual versão do VIPRO-MP, apenas os dados da memória privada são armazenados em cache. Os dados da memória compartilhada não são armazenados na memória cache, devido a não adoção de protocolos de coerência de cache no simulador.

Este modelo de utilização da memória, onde a memória privada utiliza a cache, mas a memória compartilhada não, é utilizado em plataformas embarcadas como o Nexperia (Goossens, 2005) ou mesmo o processador Cell (2011).

Para possibilitar o uso da memória cache e de protocolos de coerência, foram necessárias alterações na estrutura do modelo de memória cache no simulador VIPRO-MP. Essas alterações incluiram a reimplementação de alguns módulos como a memória cache, e a inclusão de novos módulos como o controlador de cache e o módulo *snoopy*, destacados na Figura 2.

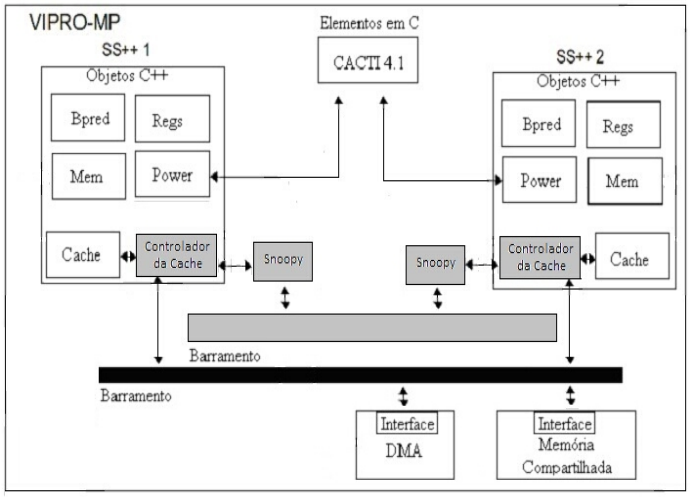

Figura 2: VIPRO-MP com módulos do protocolo de coerência rastreamento.

3.1 Alterações Estruturais na Memória Cache

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

Para que o módulo de cache pudesse ser utilizado neste trabalho, foi desenvolvido e inserido métodos para a real manipulação e armazenamento dos dados. Os dados passarram a ser inseridos na cache, alterando o fluxo de movimentação de dados do simulador.

Foram adotados os seguintes critérios na configuração da memória cache:

a) mapeamento direto de dados da memória compartilhada;

b) política de substituição de linhas de cache FIFO e;

c) política de atualização write back.

Para implementação do protocolo de coerência, um campo que controla o estado de coerência de um bloco presente na cache foi inserido nesta memória. O protocolo utilizado para garantir a coerência nos dados é o protocolo MESI, descrito na Seção 2. Assim quando uma operação ocorre sobre um dado em um determinado bloco, uma busca é realizada na memória cache do processador, caso o dado presente seja válido, este é utilizado pelo processador. Caso contrário, uma busca na cache dos demais

processadores é realizada na tentativa de encontrar o endereço em estado válido para utilização. Em último caso, uma busca é realizada na memória compartilhada, para se obter o dado necessário. Um acesso a um endereço sempre é seguido da atualização de seu estado de coerência, garantindo que o sistema permaneça coerente. Para a implementação do modelo de rastreamento os módulos implementados foram:

a) Controlador da Memória cache;

b) Snoop;

c) Barramento de coerência.

A Figura 2 apresenta a organização do simulador após a inserção dos módulos citados. Nas subseções seguintes é apresentado cada um desses módulos.

3.2 Controlador da memória cache

Após as alterações na cache, foi desenvolvido e inserido no simulador um módulo de *hardware* responsável por controlar as atividades relacionadas à utilização da memória cache. Este dispositivo recebe solicitações do processador para acesso aos dados, solicita a um controlador de cache de outro processador a busca de dados em outras memórias cache, busca dados na memória compartilhada e controla diretamente a movimentação e manipulação de dados na cache. Este dispositivo foi inserido diretamente no simulador SimpleScalar, sendo um objeto deste. Isto ocorreu devido à facilidade de acesso aos demais componentes do sistema tais como registradores, barramento e memória cache. O fluxo de execução do controlador da cache, no momento de uma requisição realizada pelo processador é demonstrado na Figura 3.

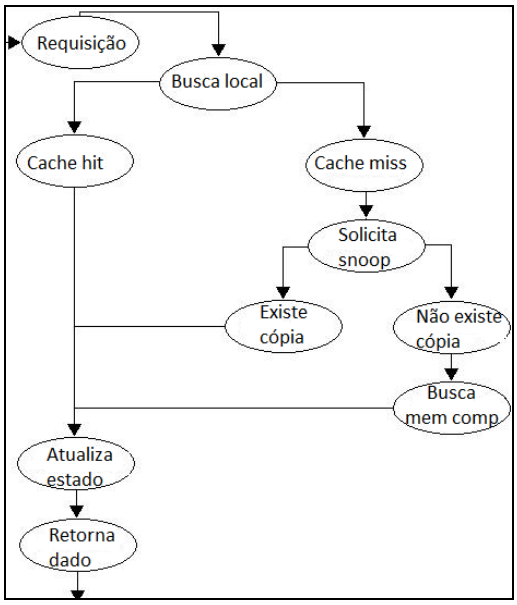

Figura 3: Fluxo de execução do controlador da cache.

No momento em que um dado é necessário, uma requisição é feita ao controlador da cache. Neste momento, o processador fica bloqueado até que esta requisição seja atendida e finalizada. O controlador realiza uma busca pelo endereço solicitado na memória cache local. Caso o endereço esteja na cache e seja válido, a operação solicitada é realizada, com a atualização dos dados conforme o protocolo

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

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

3.3 Rastreador de movimentação de dados - Snoop

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

Toda movimentação de dados entre processadores, obrigatoriamente, deve passar por Snoops. O Snoop utiliza portas no nível de transações (TLM-*transaction level modeling*) com o barramento local e com o seu respectivo controlador de cache, facilitando a implementação. O módulo Snoop foi estruturado e implementado para que não exista a necessidade de alteração, independente da quantidade de processadores presentes na arquitetura.

3.4 Barramento de coerência

O barramento de coerência de cache foi implementado adaptando-se o barramento utilizado pelo VIPRO-MP para comunicação entre processadores e memória compartilhada. É utilizado especificamente para a movimentação de dados entre as caches dos processadores presentes na arquitetura. Utiliza portas TLM, e não apresenta um árbitro centralizador. Para garantir a integridade dos dados, quando uma solicitação de movimentação é solicitada, o barramento de coerência fica bloqueado e apenas o Snoop do processador que está realizando a movimentação de dados pode utilizá-lo. Quando a movimentação de dados é finalizada, o barramento é liberado, e qualquer Snoop pode utilizá-lo novamente.

4 Estudo de Caso

Para validar os módulos de *hardware* que foram desenvolvidos foi realizado uma simulação com um decodificador JPEG paralelo (Garcia, 2009). Como pode ser observado na Figura 4, esse algoritmo é dividido em duas partes, na primeira parte cada

processador é responsável por realizar a transformada DCT (*Discrete Cosine Transform*) de N/P blocos JPEG de uma imagem e reescrever os resultados na memória compartilhada (onde N é o número total de blocos desta imagem e P é o número total de processadores presentes na simulação). Quando todos os processadores terminarem esta atividade, o processador *master* passa a realizar a parte final do algoritmo. Esta parte final é a compressão JPEG (*entropy encode*), sendo realizada de forma sequencial e gerando o arquivo de saída.

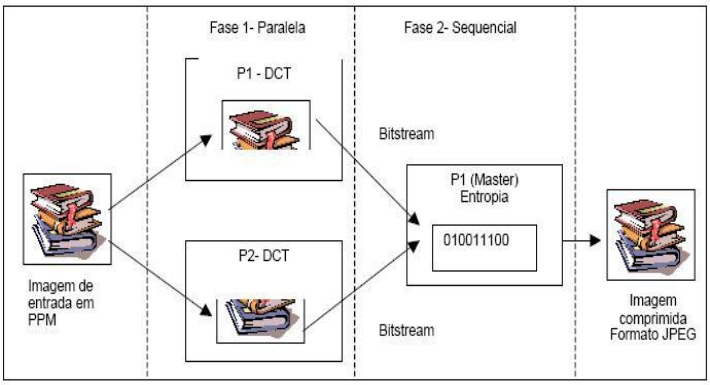

Figura 4: Algoritmo de compressão JPEG paralelo

No modelo original, não é utilizado a cache dos dados presentes na memória compartilhada. Desta forma, toda vez que um dado for requerido pelo processador uma busca deste dado é realizada na memória compartilhada, sendo em seguida fornecido para manipulação na memória local do processador. A memória compartilhada foi configurada com latência de 18 ciclos. Esse valor é o padrão utilizado no Simplescalar, e foi mantido para possibilitar comparações com resultados já reportados. Adicionalmente, conforme descrito em Panda et al. (2000), o acesso a memórias *off-chip* do tipo DRAM em processadores embarcados varia tipicamente entre 10 a 20 ciclos. Em contrapartida, a latência da memória cache é de apenas 1 ciclo para um cache *hit*.

O segundo modelo, utiliza os modulos de coerência de cache, possibilitando o armazenamento dos dados presentes na memória compartilhada na cache. Com isso o acesso a memória compartilhada é reduzida, diminuindo o tempo necessário na obtenção de um determinado dado, quando ocorre um cache *hit*. Para ambas as simulações é utilizado uma imagem de 256x256 *pixels*, e arquiteturas com um, dois, quatro e oito processadores. Os processadores foram configurados para utilizar memória cache de instruções de 128Kbytes e cache de dados de 256Kbytes. Essa configuração de cache foi considerada como ideal para esta aplicação conforme apresentado em (Garcia, 2009).

Como o desempenho da aplicação é relacionado com o número de ciclos gastos, na Tabela 1 são apresentados o total de ciclos gastos para realizar uma transferência de um dado entre a memória cache e a memória compartilhada. Neste trabalho foi considerado que em arquiteturas com maior número de processadores, maior será o número de ciclos para o acesso, devido a necessidade de um barramento fisicamente maior, resultando em uma frequência de operação menor.

Tabela 1: Ciclos de execução do protocolo de coerência por rastreamento.

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

Os resultados das simulações do algoritmo JPEG paralelo sem cache dos dados da memória compartilhada e com cache dos dados utilizando o protocolo de coerência modelo rastreamento estão apresentados na Tabela 2. O número de ciclos utilizado para realizar esta comparação se refere ao montante total de ciclos gastos pelo processador *master*. Isto se deve ao fato deste processador ser responsável pela carga da imagem a ser decodificada na memória compartilhada, e também pela ultima parte do algoritmo (compressão). Analisando o total de ciclos gastos é possível perceber uma redução no tempo de execução variando de 53,57% para arquiteturas com um processador até 69,18% para arquiteturas com oito processadores. Desta forma, mesmo considerando latências necessárias para o protocolo de coerência de cache, um redução significativa foi alcançada na utilização da cache para a memória compartilhada. Estes dados comprovam os benefícios que a utilização da memória cache pode prover em um sistema embarcado.

Tabela 2: Ciclos gastos para execução do algoritmo JPEG paralelo, sem cache da memória compartilhada e com protocolo de coerência.

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

5 Conclusão

Geralmente destinados ao mercado consumidor, um sistema embarcado deve ser elaborado em curto espaço de tempo e com elevado desempenho e custos reduzidos. Erros de projeto devem ser evitados para não gerar um acréscimo de custo e tempo de finalização. Buscando evitar potenciais erros de projeto, protótipos virtuais passam a ser ferramentas indispensáveis em projetos de sistemas embarcados.

A utilização de protótipos virtuais e a facilidade de modelagem de componentes de hardware através de software, possibilitam avaliar rapidamente diferentes configurações da arquitetura, possibilitando avaliar o impacto da inclusão de um determinado componente em uma arquitetura. Através dos módulos de coerência de

cache implementados e integrados ao VIPRO-MP foi possível analisar o impacto, em termos de desempenho, que o uso da memória cache pode prover uma arquitetura. Os resultados mostram que o uso do protocolo de coêrencia de cache pode resultadar em um ganho de desempenho de até 69,18% para o caso de um processador de 8 núcleos. Mesmo com a utilização de apenas 1 núcleo, um ganho de 53,57% é obtido, pois no modelo original do VIPRO-MP os dados da memória compartilhada não são armazenados em cache. Esse ganho é resultado do reuso dos dados na cache, evitando assim o acesso à memória compartilhada.

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

Referências

GARCIA, M. S.; SCHUCK, M ; OYAMADA, M. S. VIPRO-MP: a virtual prototype for multiprocessor architectures based on the SimpleScalar. In: 17th Annual IFIP International Conference on Very Large Scale Integration VLSI-SoC, 2009, Florianopolis. Proceedings of VLSI-SoC 2009, 2009. v. 1.

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

IBM Cell Broadband Engine Architecture, Disponível em: www.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA. Acessado em: jul/2011.

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

MARWEDEL, P. Embedded Systems Desing. 1st Edition, Netherlands: Springer, 2006.

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

PATTERSON, D. A., HENNESSY, J. L. Computer architecture: a quantitative approach, 3rd Edition. San Francisco: Morgan Kaufmann Publishers, 2002.

SystemC. Disponível em www.systemc.org. Acesso em Maio/2011.

PANDA, R.P., DUTT, N., NICOLAU, A. On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. *ACM Transactions on Design Automation of Electronics Systems.* v.5,no. 3, Julho 2000, pp. 682-704. http://doi.acm.org/10.1145/348019.348570.

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