Investigação do Uso de Caches com Suporte a Coerência de Dados em Plataformas MPSoC baseadas em NoC

Gustavo Girão
Universidade Federal do Rio Grande do Norte
girao@natalnet.br

Bruno Cruz de Oliveira
Universidade Federal do Rio Grande do Norte
bcruz@natalnet.br

Rodrigo Soares
Universidade de São Paulo
rsoares@lme.usp.br

Ivan Saraiva Silva
Universidade Federal do Rio Grande do Norte
ivan@dimap.ufrn.br

Resumo

Os Multiprocessor System-on-Chip surgem como grande solução para dar suporte a aplicações que exigem um maior poder computacional em um único chip. Nestes sistemas, a coerência e consistência de cache ainda é um assunto em aberto. Este artigo apresenta uma investigação da hierarquia de memória de uma plataforma multiprocessada baseada em Network-on-Chip no que diz respeito ao uso de cache e uma análise das conseqüências do uso de um mecanismo de coerência de cache. São apresentados resultados comparativos entre uma plataforma com cache e uma versão sem cache. Estes resultados dizem respeito à carga injetada na rede durante a execução da aplicação, o tempo de resposta das operações de leitura bem como speedup e eficiência.

1. Introdução

A crescente demanda por poder de processamento, resultado do surgimento de algoritmos e aplicações cada vez mais exigentes, aliada a capacidade de integração de bilhões de transistores em uma única pastilha [1], tem, nos últimos anos, contribuído significativamente para a consolidação dos MPSoCs (*Multiprocessor System on Chip*).

O conceito de MPSoC possibilita a integração de todo um sistema computacional, incluindo de dezenas a centenas de elementos de processamento, dispositivos de entrada e saída e software embarcado, em um único circuito integrado. O surgimento de tal conceito impôs a necessidade de solucionar vários novos problemas relacionados ao projeto dos MP-SoC, tais como: escalabilidade e reuso do sub-sistema de interconexão, hierarquia de memória, compartilhamento e distribuição de dados, modelo de programação e coerência e consistência dos dados.

Alguns artigos recentes discutem aspectos importantes do projeto de MP-SoCs, principalmente no que diz respeito ao uso de redes em chip (NoC - *Network on Chip*) como sub-sistema de interconexão. O projeto e uso de tais estruturas por si só já levanta muitas questões que já vêm sendo investigadas [2] tais como topologias, arquiteturas, implementação, avaliação de desempenho e qualidade de serviço.

Em MPSoCs baseados em NoC o compartilhamento de dados entre os processadores do sistema impõe a necessidade de desenvolver alternativas para a manutenção da consistência da memória e coerência de cache.

O problema da coerência de cache, em particular ocorre em um ambiente com memória compartilhada quando uma tarefa T1, sendo executada em um processador P1, deseja modificar um dado que está sendo compartilhado com uma tarefa T2, executando em um processador P2. Ao realizar tal modificação, o dado (do ponto de vista de T2) fica desatualizado na cache do processador P2.

Com o uso de redes em chip como estrutura de interconexão em MP-SoCs, a manutenção da coerência da memória é um problema de difícil solução, já que a utilização de protocolos do tipo *snoop* é inviável..

Este artigo apresenta uma investigação sobre o impacto do uso de cache em uma plataforma MPSoC baseada em NoC, que utiliza uma solução de diretório para o problema da manutenção da coerência. Uma implementação sem cache é comparada com uma implementação com cache. As comparações são feitas com base na execução de aplicações reais para obtenção de resultados, tais como *speedup*, carga injetada na NoC e tempo de resposta de leituras e escritas. O artigo está organizado da seguinte forma. A

seção 2 apresenta os trabalhos relacionados em coerência de cache em MPSoCs. A seção 3 apresenta a plataforma STORM enquanto que a seção 4 uma versão desta mesma plataforma sem memória cache. A quinta seção discute o suporte à programação paralela em ambas as versões da plataforma. A seção 6 apresenta os resultados da comparação entre as duas versões da plataforma. A sétima seção apresenta as conclusões seguidas das referencias.

2. Trabalhos relacionados

Nos últimos anos vários artigos que abordam o projeto de plataformas MPSoC foram publicados. A maioria deles são plataformas heterogêneas baseadas em barramento [4], [5], [6] enquanto que uma pequena, porém crescente, quantidade de trabalhos sobre plataformas baseadas em NoC vêm ganhando destaque [2], [7]. Muitos artigos sobre projeto de plataformas baseadas em NoC focam a execução da aplicação negligenciando a necessidade de garantir a consistência e coerência de cache. De fato, a execução da aplicação é transformada em um profile impreciso da aplicação. Nestes artigos, coerência e consistência de cache aparecem como problemas pretensamente resolvidos.

De acordo com nosso conhecimento, poucos artigos têm focado especificamente o problema da coerência e consistência de cache. [3] propõe uma metodologia de hardware/software para garantir coerência de cache em uma plataforma heterogênea com memória compartilhada baseada em barramento. [2] fala sobre o problema da coerência de cache em plataformas baseadas em NoC e propõe uma solução inteiramente em software. Outros trabalhos [7], [8], [9] apresentam sistemas sem cache e modelos de programação onde o problema da coerência e consistência de cache são inexistentes.

[10] apresenta uma solução baseada em diretório para o problema da coerência de cache aplicado a MPSoC baseado em NoC utilizando um ambiente de memória distribuída compartilhada. Entretanto, nesta solução o diretório é embarcado no *switch* da rede e é mostrado algum ganho de performance com essa abordagem. Neste artigo não é apresentado nenhuma informação sobre o tráfego gerado na NoC devido ao uso da solução da coerência de cache.

A próxima seção apresenta uma solução completamente em hardware para o problema de coerência de cache em uma plataforma multiprocessada baseada em NoC.

3. Plataforma STORM

A plataforma STORM (MP**S**oC Direc**t**ory-based Platf**orm**) é um modelo de SoC (System-on-Chip) multiprocessado customizável e desenvolvido com precisão de ciclo. Tendo em vista um estudo comportamental e de fácil adaptação, este projeto foi feito utilizando o conceito de desenvolvimento baseado em plataforma e foi desenvolvido em *SystemC*, No caso do projeto desta plataforma, foi utilizado um nível de abstração conhecido como Transaction Level Modeling (TLM).

Como dito, STORM é uma plataforma multiprocessada e suporta até 256 cores interconectados através de uma NoC. Existem dois modelos de NoC que podem ser utilizados pela plataforma: NoCX4 [11] e a ObTree. Ambos os modelos foram desenvolvidos no escopo deste trabalho e serão descritos na seção 3.1.

Por ser uma plataforma, STORM não conta com uma arquitetura fixa. Os cores suportados podem ser dispostos em qualquer ponto da NoC e são identificados durante o processo de boot. Em cada simulação é gerada uma instância da plataforma que descreve que recursos (processadores e memórias) estão conectados em que roteadores da NoC.

Atualmente a plataforma conta com um processador SPARC V8 e que executa código binário gerado diretamente pelo compilador GCC. Além disso, existe também um modelo de memória que pode ser instanciado na plataforma assim como caches de dados e caches de instruções para cada processador. STORM conta com um modelo de memória compartilhado, entretanto podem ser instanciados diversos módulos de memória que formam um espaço de endereçamento único.

Cada um dos processadores SPARC da plataforma executam o mesmo código e devido ao fato de terem suas próprias caches, um mecanismo de manutenção de coerência de cache teve de ser desenvolvido. Por utilizar como mecanismo de interconexão uma NoC, optou-se por implementar um mecanismo de coerência de cache baseado em diretório em detrimento do uso de *snoop*. Esta decisão tem uma razão muito simples: o protocolo *snoop* baseia-se na difusão de mensagens que refletem a ação realizada por cada cache em um bloco. Desta maneira, a cada operação de escrita ou leitura por parte de uma cache resultaria em uma mensagem trafegando no sistema para todas as outras caches. Em um ambiente com diversas caches esta quantidade elevada de mensagens degradaria rapidamente a comunicação. Esta abordagem, apesar de muito utilizada em sistemas que utilizam barramentos que dão

suporte a operações de broadcast, não oferece vantagens em sistemas baseados em NoC já que uma operação de broadcast é muito custosa nestes caso. O mecanismo de diretório implementado é discutido na seção 3.3.

A figura 1 apresenta um exemplo de instância da plataforma STORM.

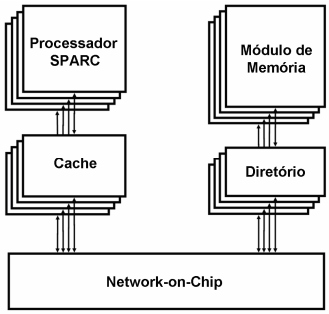


Figura 1. A Plataforma STORM

3.1 Mecanismo de Interconexão

Como já destacado, STORM utiliza como mecanismo de interconexão uma NoC. Atualmente conta com as redes NoCX4 e ObTree. A rede NoCX4 é uma mesh que utiliza chaveamento Store-and-Forward, roteamento determinístico XY, arbitragem distribuída e controle de fluxo baseado em créditos.

A ObTree implementa uma adaptação do modelo *Fat Tree* [12] criando uma conexão entre os roteadores folha e intermediário que estão no mesmo nível. Esta rede difere da NoCX4 em sua topologia (baseada em árvore) e em outros dois aspectos: no algoritmo de roteamento, já que utiliza Menor Caminho, e no chaveamento fazendo uso de *Virtual Cut Through*.

3.2 Coerência de cache

Por utilizar uma NoC e pelos motivos já mencionados, optou-se por utilizar uma solução de manutenção de coerência de cache baseada em diretório na plataforma STORM.

Como dito antes, cada processador da plataforma tem uma cache associada. Este módulo de cache é subdividido em Cache de Dados e Cache de Instruções. Além disso, o módulo de cache conta com um módulo responsável por fazer a interface entre a cache e o roteador da NoC. Este módulo e denominado CaCoMa (Cache Communication Manager).

Tanto a Cache de Dados quanto o módulo CaCoMa foram desenvolvidos com suporte a operações de coerência de cache de forma que possam se comunicar com o módulo Diretório. Cada módulo de memória tem um módulo Diretório que gerencia a comunicação vinda da NoC bem como as solicitações de leitura/escrita vindas das caches de modo a manter a coerência dos dados. Este módulo tem a informação do estado de cada bloco na memória (se o mesmo encontra-se limpo ou sujo) e os processadores que tem uma cópia deste bloco.

A plataforma faz uso da implementação de um Diretório do tipo Dir_n NB, onde NB significa sem difusão (do inglês, *Non-Broadcast*). Dir_n por sua vez significa que são necessários n+1 bits por bloco na memória para armazenar seu estado, onde n diz respeito ao número de processadores na plataforma.

Para armazenar o estado do bloco, o diretório faz uso de duas tabelas: Status TAble (STA) e Processor TAble (PTA). A STA contém informações de estado que dizem respeito a todos os blocos na memória a qual o diretório está conectado. A PTA informa o endereço dentro da NoC de todos os processadores do sistema. O diretório necessita desta informação para que possa enviar solicitações que dizem respeito à manutenção da coerência de cache para as caches corretas. Na STA cada linha representa o estado de um bloco na memória (onde cada coluna representa um processador). Um bit indica se o bloco está sujo ou não. Se o bloco estiver sujo somente um processador deverá ter a cópia deste bloco. Uma vez que a STA não contém nenhuma informação sobre a localização dos processadores, é necessária a tabela PTA. Tanto a PTA quanto a STA são implementadas como memórias dedicadas do diretório, logo nenhum acesso à memória do sistema é necessário.

Como já mencionado, o CaCoMa e a cache de dados foram implementados com suporte a operações de manutenção da coerência de cache. Logo, além de operações de leitura de bloco e escrita de bloco existem também operações de requisição de escrita de bloco, substituição de bloco, invalidação de bloco, *test-and-set* e *test-and-reset*. Estas duas últimas são resultado do uso de mutexes como mecanismo de consistência de dados. A implementação dos mutexes será discutida com maiores detalhes na seção 5. As figuras 2 e 3 mostram um fluxograma das operações de leitura e escrita e resumem o processo de manutenção de coerência de cache.

Devido ao fato do CaCoMa ser responsável por enviar e receber solicitações do diretório, é necessário que ele saiba em que endereço de NoC o diretório se encontra. Para isso o módulo conta com uma tabela chamada ATA (Address TAble) que contém o endereço de NoC de todos os módulos de memória (e consequentemente seus respectivos diretórios) bem como os endereços contidos nesta memória (uma vez que o espaço de endereçamento é único porém

podendo existir mais de uma memória fisicamente distintas umas das outras). Assim como a PTA e a STA, a ATA é uma memória dedicada do módulo CaCoMa.

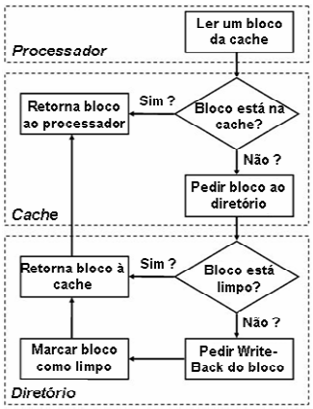


Figura 2. Operação de leitura utilizando diretório.

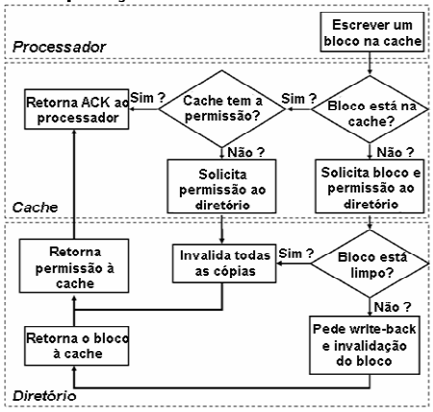


Figura 3. Operação de escrita utilizando diretório.

4. STORM Cacheless

A partir da plataforma STORM detalhada na seção anterior, foi criada uma nova versão da mesma sem a utilização de caches denominada STORM Cacheless. Dessa forma, todas as solicitações de leitura e escrita por parte do processador têm o mesmo efeito de um *miss* na cache da versão anterior da plataforma. A diferença está no fato de que se não existem caches não faz sentido ter um módulo diretório associado a cada módulo de memória.

Esta versão da plataforma não acarreta no envio de mensagens para a manutenção da coerência de cache, entretanto toda solicitação vinda do processador resulta em um acesso à memória através da NoC. No caso de uma leitura o tempo de retorno do dado ainda deve ser acrescido.

Em suma, os módulos de Cache (incluindo Cache de Dados, Cache de Instruções e CaCoMa) e Diretório foram substituídos por módulos que simplesmente geram mensagens a serem enviadas de acordo com a solicitação (leitura, escrita, *test-and-set* ou *test-and-reset*).

5. Suporte a programação paralela

Dado que a plataforma STORM (e a plataforma STORM Cacheless) utiliza um modelo de memória compartilhado e que cada processador executa o mesmo código é necessário o suporte a algum mecanismo de exclusão mútua para a manipulação de variáveis globais. Para dar esse suporte foi desenvolvida uma biblioteca de mutexes. A biblioteca disponibiliza operações de *up* e *down* que representam o travamento e a liberação do mutex, respectivamente.

Estas operações devem ser executadas de maneira atômica e por isso foram implementadas utilizando instruções atômicas do SPARC. A instrução *LDSTUB* executa um *load* atômico foi utilizada para simular a operação de *down* enquanto que a instrução *SWAP* que realiza uma troca atômica entre um valor num registrador e um valor na memória foi utilizada para simular a operação de *up*. Quando o processador SPARC encontra uma destas instruções ele sinaliza para a Cache (ou para o gerenciador de pacotes, no caso da STORM Cacheless) que necessita fazer uma operação de *test-and-set* (no caso da instrução *SWAP*) ou uma operação de *test-and-reset* (no caso da instrução *LDSTUB*).

Uma vez disponibilizada a biblioteca, é possível executar algoritmos paralelos nas plataformas STORM e STORM Cacheless.

As simulações foram realizadas utilizando os seguintes algoritmos: Mergesort, Estimação de Movimento e Multiplicação de Matrizes.

Para paralelizar o algoritmo Mergesort foi utilizada uma modelagem de forma tal que o vetor de entrada é dividido em N partes iguais onde N é o número de processadores escravos. Ao fim da ordenação de cada uma das partes do vetor, um processador mestre é responsável por fazer a junção dos sub-vetores ordenados. Os resultados apresentados na seção 6 são relativos à execução do Mergesort com uma entrada de 6000 posições.

O algoritmo de estimação de movimento realiza uma subtração entre os elementos de uma matriz de tamanho p x p e as sub-matrizes de mesmo tamanho contidas em uma matriz (chamada região de busca) de tamanho n x n (onde $n > p$). O menor resultado é apontado como o índice do vetor de movimento. Para

paralelizar este algoritmo foi utilizada a estratégia de dividir igualmente a região de busca de forma que no fim da busca um processador é responsável por comparar todos os melhores resultados dos processadores e eleger o melhor dentre eles. Nas simulações a região de busca utilizada tem tamanho 64 x 64.

Por fim o algoritmo de multiplicação de matrizes foi implementado de forma que cada processador é responsável por realizar a multiplicação de um número igualmente dividido de linhas da matriz A por todas as colunas da matriz B. Ao fim do cálculo de todas as linhas, cada processador é responsável por gerar sua parte correspondente da matriz C resultante. Os resultados obtidos da execução deste algoritmo dizem respeito à multiplicação de matrizes de tamanho 64 x 64.

6. Simulações e Resultados

6.1 Simulações

Por utilizar uma linguagem de descrição de hardware tal qual systemC, a metodologia de uso da plataforma bem como o ambiente de simulação são bastante específicos. A figura 4 ilustra a metodologia de uso da plataforma STORM.

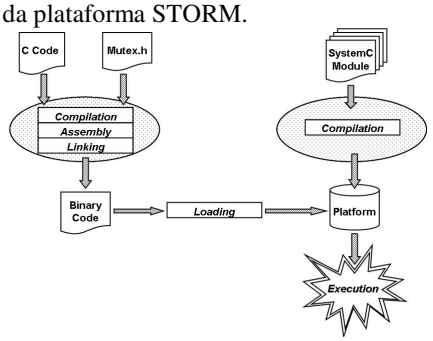


Figura 4. Metodologia de uso da plataforma.

Para realizar uma simulação da plataforma, em primeiro lugar é gerado um código binário do programa a ser executado. Este binário é gerado a partir de um *cross-compiler* para a arquitetura SPARC (GCC-SPARC). Uma vez de posse deste binário, se faz necessário a obtenção de uma versão executável da plataforma. Este executável é gerado a partir da compilação dos módulos descritos em systemC. Ao simular a plataforma, o código binário do programa é carregado nos módulos de memória da mesma.

Nas simulações o tamanho da cache de dados variou de 32 bytes até 64Kbytes enquanto que a memória teve um tamanho de 40Mbytes. As caches utilizadas tem tamanho de linha de 8 palavras (cada palavra de 4Kbytes), completamente associativas e política de substituição FIFO.

6.2 Resultados da Plataforma STORM

Primeiramente foram feitas simulações dos algoritmos com instâncias da plataforma STORM com caches. As figuras 5, 6 e 7 mostram a carga injetada na NoC para os algoritmos Mergesort, Estimação de movimento e Multiplicação de Matrizes, respectivamente, para estas simulações.

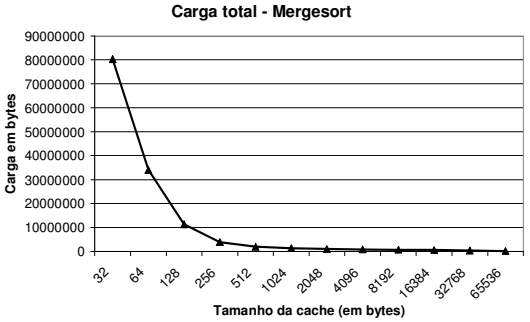


Figura 5. Carga – Mergesort.

Observando os gráficos conclui-se que devido a menor necessidade de solicitações de leitura/escrita, a carga injetada na NoC diminui com o aumento do tamanho da cache.

As figuras 8, 9 e 10 apresentam o tempo médio de resposta para operações de leitura e escrita para os três algoritmos.

Com base nestes gráficos pode-se ver que alguns tamanhos de cache resultam em execução mais eficiente dos algoritmos, quando considerados apenas os parâmetros medidos, ou seja, carga injetada na NoC e tempo de resposta de operações de leitura e escrita. No que diz respeito à quantidade de carga injetada na NoC, percebe-se que com caches entre 256 bytes e 1Kbyte o trafego injetado na rede assume valores razoáveis. Por outro lado, considerando o tempo de resposta das operações de leitura e escrita, caches a partir de 1Kbyte apresentam resultados satisfatórios.

Analisando estes resultados optou-se por fixar o tamanho da cache em 1Kbyte e variar a quantidade de processadores para a execução de cada algoritmo e realizar simulações com as mesmas instâncias (tamanho da cache, número de processadores, algoritmo) com a STORM Cacheless.

6.3 STORM versus STORM Cacheless

Para cada algoritmo foram realizadas simulações com 1, 2, 4 e 8 processadores. A partir das simulações foram analisados os seguintes aspectos: Carga injetada na NoC, Tempo médio de espera para operações de leitura/escrita, *Speedup* e Eficiência.

As curvas da figura 11 e 12 mostram o *speedup* obtido para as aplicações Mergesort e Estimação de movimento no uso de 2, 4 e 8 processadores. Os resultados mostram um claro ganho de performance da plataforma com cache em comparação com a plataforma sem cache. Isso se deve ao fato do tempo de resposta de pedidos de leitura/escrita na plataforma sem cache ser muito alto deixando os processadores ociosos por muito tempo.

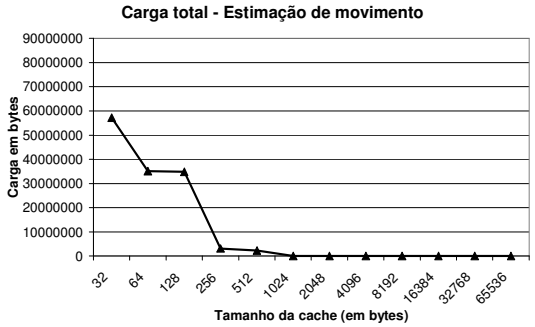


Figura 6. Carga– Estimação de movimento

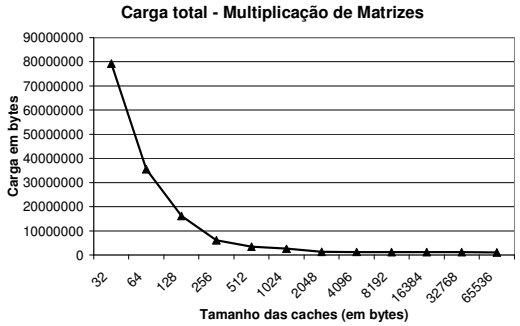


Figura 7. Carga – Multiplicação de matrizes

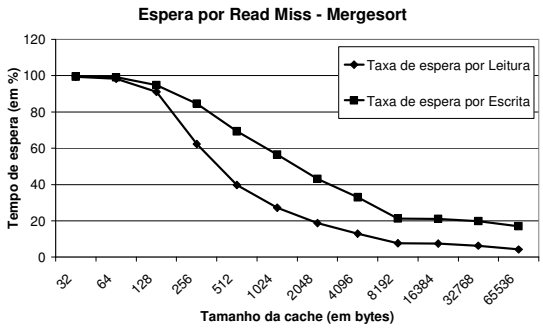
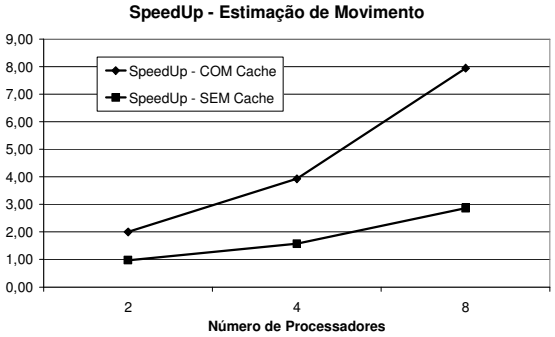


Figura 8. Tempo de resposta para operações de leitura e escrita – Mergesort.

As figuras 13 e 14 ilustram a eficiência das aplicações Mergesort e Multiplicação de Matrizes. Aqui, a superioridade da plataforma com cache já não é tão grande, entretanto este é um fator que é bastante influenciado pela abordagem de paralelização utilizada para os algoritmos. Mesmo assim percebe-se vantagem da plataforma STORM sobre a plataforma STORM Cacheless neste aspecto.

No que diz respeito ao tempo de resposta para operações de leitura (figuras 15 e 16) e escrita (figuras 17 e 18) nas aplicações de Estimação de Movimento e Multiplicação de Matrizes ocorre um enorme prejuízo em desempenho na plataforma sem cache devido à necessidade constante de acesso a memória.



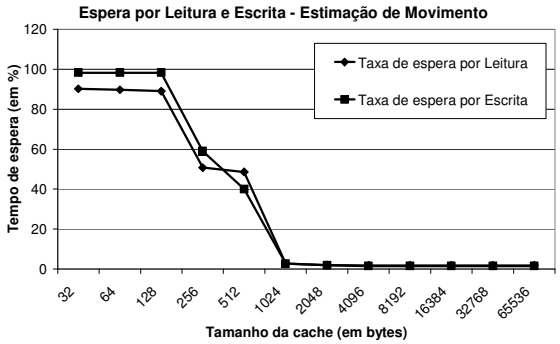


Figura 9. Tempo de resposta para operações de leitura e escrita – Estimação de movimento.

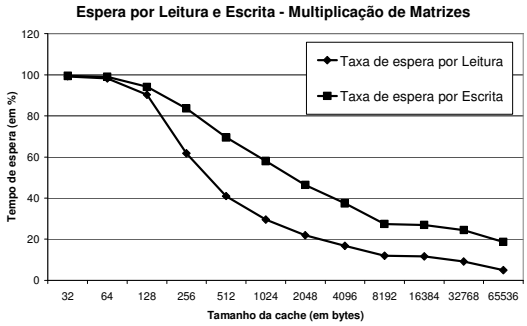


Figura 10. Tempo de resposta para operações de leitura e escrita – Multiplicação de matrizes.

A carga total de dados que trafegaram na rede é mostrada nas figuras 19 e 20 e refletem os resultados obtidos nas simulações dos algoritmos de Estimação de Movimento e Mergesort. Nestes gráficos percebe-se que a quantidade de dados que trafegam na rede na

versão da plataforma sem cache é muitas vezes maior do que a versão com cache. Isso se deve ao fato de que, em decorrência da falta de uma memória cache, o processador precisa enviar várias solicitações de leitura e escrita durante toda a simulação. Estes resultados comprovam a pouca quantidade de tráfego na rede gerado pela implementação de coerência de cache utilizado na plataforma STORM.

Por falta de espaço não foi possível inserir os gráficos de todos os algoritmos em cada aspecto analisado. Entretanto os gráficos omitidos demonstram comportamentos semelhantes aos apresentados aqui.
Em nenhum dos aspectos analisados, a plataforma STORM Cacheless se mostrou melhor do que a versão da plataforma com cache.

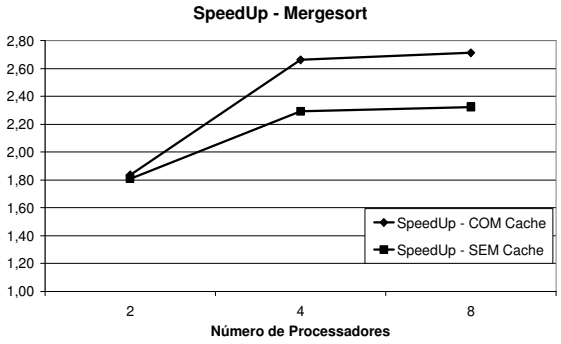


Figura 11. *Speedup* – Mergesort.

Figura 12. *Speedup* – Estimação de Movimento.

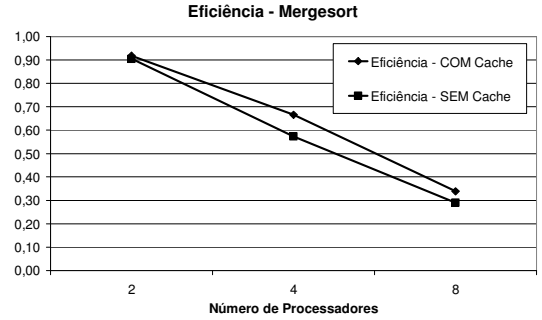


Figura 13. Eficiência – Mergesort.

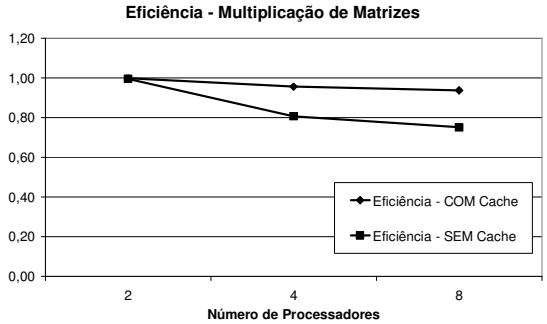


Figura 14. Eficiência – Multiplicação de Matrizes.

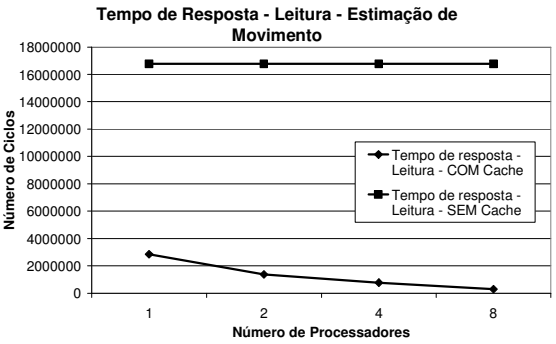


Figura 15. Tempo de resposta - Leitura – Estimação de Movimento.

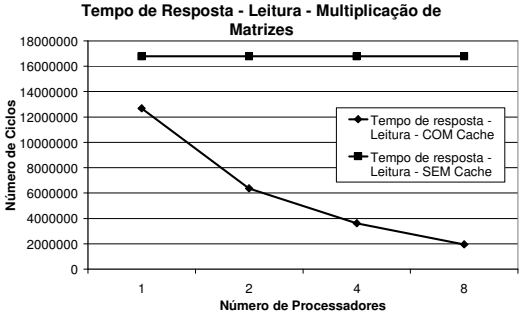


Figura 16. Tempo de resposta - Leitura – Multiplicação de Matrizes.

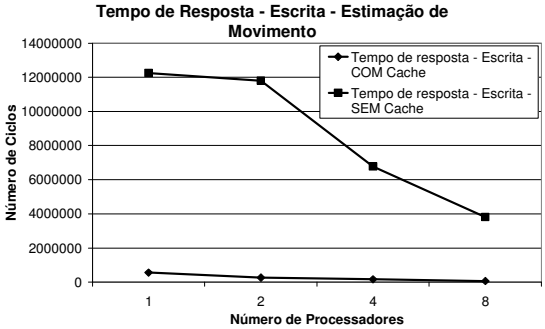


Figura 17. Tempo de resposta - Escrita – Estimação de Movimento.

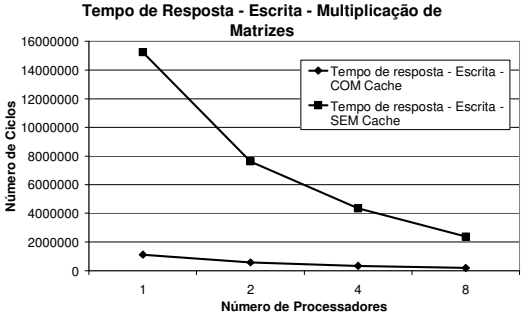


Figura 18. Tempo de resposta - Escrita – Multiplicação de Matrizes.

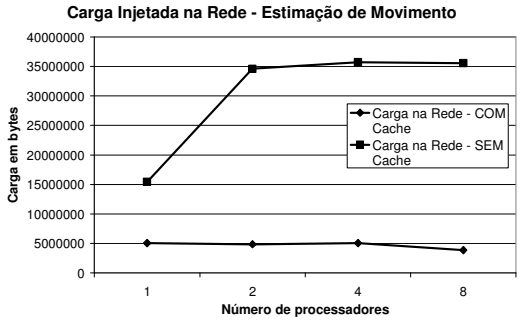


Figura 19. Carga – Estimação de Movimento.

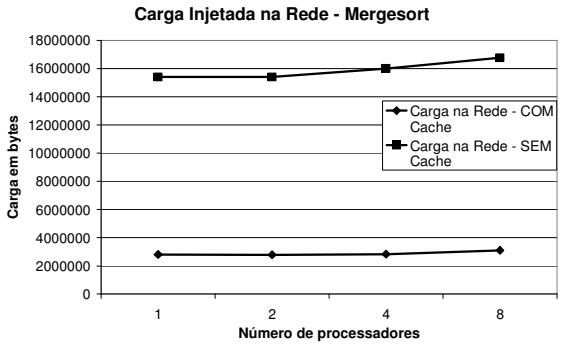


Figura 20. Carga – Mergesort.

7. Conclusões

Este artigo apresentou uma implementação de uma plataforma MPSoC baseada em NoC com coerência de cache baseada em diretório bem como uma versão da mesma plataforma sem cache e por conseqüência sem módulos dedicados a manutenção da coerência de cache.

Várias simulações foram realizadas. Primeiramente com a plataforma com vários tamanhos de cache diferentes. Num segundo passo foram feitas diversas simulações de ambas as plataformas com algoritmos bastante conhecidos na literatura com o propósito de comparar aspectos relevantes como carga injetada na NoC, tempo de resposta de operações de leitura/escrita, *speedup* e eficiência.

Os resultados apresentados comprovam que, baseado nos aspectos analisados, o uso de cache, apesar de criar um custo maior de área e potência (fatores não considerados nesta análise) acarreta em um grande ganho de performance. Além disso, os resultados mostram que os efeitos colaterais do uso do mecanismo de coerência de cache, tais como a necessidade do envio de mensagens para controle, são compensados pelo fato de não ser necessário fazer acesso à memória em todas as tentativas de leitura/escrita feitas pelo processador.

8. Referências

[1] ITRS, International technology roadmap for semiconductors update 2003, http://public.itrs.net

[2] Petrot, F.; Greiner, A.; Gomez, P.; "On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures". In: Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on 30-01 Aug. 2006 Page(s):53 – 60.

[3] Hwang, K. "Advanced Computer Architecture: Parallelism, Scalability, Programmability", McGraw-Hill, Inc. 1993.

[4] Benini, L. et al; "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC". The Journal of VLSI Signal Processing, 41(2): 169 – 182, September 2005

[5] Fummi, F. et al; "Native ISS-SystemC Integration for Co-Simulation of Multi-Processor SoC". Design, Automation and Test in Europe (DATE), Proceedings, 2004

[6] Suh, T.; Blough, D. M.; Lee, H. H. S. "Supporting cache coherence in heterogeneous multiprocessor systems". Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings Volume 2, 16-20 Feb. 2004 Page(s):1150 - 1155 Vol.2

[7] Forsell, M. "A scalable high-performance computing solution for network on chip". IEEE Micro, vol. 22, no. 5, pp. 46–55, Sep.—Oct. 2002.

[8] Paulin, P.; Pilkington, C.; Bensoudane, E. Stepnp: "A system-level exploration platform for network processors". IEEE Design & Test of Computers, pages 17–26, Nov. 2002.

[9] Salapura, V.; Georgiou, C. J.; Nair, I. "An efficient systemon- a-chip design methodology for networking applications". In Proc. of the Int'l Conf. on Compilers, Arch. and Synth. Of Embedded Sys., Washington, DC, Sept. 2004.

[10] Kim, D.; Kim, M.; Sobelman, G.E. "DCOS: Cache Embedded Switch Architecture for Distributed Shared Memory Multiprocessor SoCs," *Proceedings, IEEE International Symposium on Circuits and Systems,* pp. 979-982, 2006.

[11] Soares, R.; Silva, I.S.; Azevedo, A. "When Reconfigurable Architecture Meets Network-on-Chip", 17th Symposium on Integrated Circuits and Systems Design, pp. 216 – 221, 2003.

[12] Cho, S.L.; Yang, M.K.; Lee, J.; "Analytical modeling of a fat-tree network with buffered switches**",** Communications, Computers and signal Processing, 2001, pp:184 – 187