Comparação de Modelos de Memória para Plataformas MPSoC Usando SystemC

Bruno Cruz de Oliveira[1], Ivan Saraiva Silva[2]

[1]Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brasil

[2]Departamento de Informática e Estatística
Universidade Federal do Piauí (UFPI) – Teresina, PI – Brasil

bcruz@lasic.ufrn.br, ivan@ufpi.edu.br

Abstract. *This paper presents two memory organizations exploration for MP-SoC virtual platforms. A general purpose platform is present alongside its description in SystemC. The STORM platform allows one to describe architectures comprising processors, networks-on-chip and cache memories. The focus of this paper is to evaluate the impact of shared and distributed memories organizations in the design of multiprocessor systems. To evaluate such scenarios, specific instances of STORM were submitted to a oil reservoir simulation application.*

Resumo. *Este artigo apresenta a exploração de dois modelos de memória para plataformas virtuais MPSoC. Uma plataforma de propósito geral é apresentada em conjunto com sua descrição em SystemC. A plataforma STORM permite descrever arquiteturas incluindo processadores, redes-em-chip e memórias caches. O foco desse artigo é avaliar o impacto dos modelos de memória distribuída e compartilhada no projeto de sistemas multiprocessados. Para avaliar tais cenários, diferentes instâncias da plataforma STORM foram utilizadas para executar uma aplicação de simulação de reservatório.*

1. Introdução

O constante aumento de complexidade das aplicações demanda suporte de hardware cada vez mais poderoso do ponto de vista de sua capacidade computacional. Com a aproximação do limite de freqüência dos processadores, a solução mais explorada e, provavelmente uma das mais viáveis é o paralelismo. Sendo assim, a integração de múltiplos núcleos processantes em um único chip se torna interessante, dada a alta densidade de transistores em chip permitida pelas tecnologias de fabricação atuais. As principais pesquisas nesse sentido estão focadas no conceito de Sistemas em Chip Multiprocessados (MPSoC). No entanto, sistemas arquiteturalmente complexos como os MPSoCs apresentam diversos desafios aos projetistas. Muitos desses desafios, tais como, manutenção da coerência de cache e consumo de energia vêm sendo estudados há algum tempo. Contudo, muitas soluções arquiteturais são possíveis, dado o número de componentes disponíveis, o que evidencia um enorme espaço de projeto a ser explorado.

Este artigo tem como foco o problema da organização de memória em MPSoCs como forma de atender as restrições de desempenho de aplicações distribuídas. Neste

trabalho é abordado um cenário onde múltiplas tarefas podem ser alocadas em diferentes elementos processantes de um MPSoC. Para avaliar o desempenho, quanto ao tráfego injetado pelas tarefas, no subsistema de comunicação, dois modelos de arquitetura de memória são explorados em instâncias diferentes de um MPSoC baseado em Redes-em-Chip (NoC – Network-on-Chip).

O primeiro é o modelo de memória compartilhada. Neste modelo, o espaço de endereçamento é formado por um ou mais módulos de memória, distribuídos na rede-em-chip e compartilhada por todos os elementos processantes do MPSoC. O segundo é o modelo de memória distribuída. Neste modelo cada elemento processante do MPSoC tem seu próprio módulo de memória, com espaço de endereçamento privado. Enquanto no modelo de memória compartilhada é usada a comunicação baseada em variáveis compartilhadas, no modelo de memória distribuída a comunicação é baseada em troca de mensagens. Para avaliar o desempenho em termos de comunicação e processamento entre os dois modelos de memória, são comparadas: a latência dos pacotes, a carga injetada na rede, o número médio de ciclos por instrução e o número de instruções executadas.

Como estudo de caso, foi implementada uma aplicação da área de exploração de petróleo e gás natural, a simulação de reservatório [Fachi, Harpole e Bujnowski 1982]. Esse tipo de aplicação é conhecida por demandar alta capacidade de processamento.

Este trabalho está organizado da seguinte forma: na seção 2 é apresentada a plataforma virtual STORM (MPSoC Directory-Based Platform) [Girão et al. 2007]. A seção 3 detalha a aplicação usada como estudo de caso. A seção 4 mostra os experimentos e discute os resultados obtidos. A conclusão é apresentada na Seção 5.

2. A Plataforma STORM

Nos últimos anos alguns trabalhos sobre o projeto de plataformas MPSoC foram publicados. A maioria deles com foco em plataformas baseadas em barramentos heterogêneos [Benini et al 2006], [Fummi et al 2004], [Suh, Blough e Lee 2004], enquanto que um número menor, mas crescente, tem foco em plataformas baseadas em NoC [Petrot, Greiner e Gomez 2006], [Forsell 2002]. A maioria dos trabalhos de projeto de plataformas baseadas em NoC focam o desenvolvimento e a execução de aplicações e a exploração arquitetural. No entanto, a organização e a otimização da memória devem ser consideradas durante a fase de projeto, visto que grande quantidade de aplicações requer estruturas de dados complexas. Não obstante a importância da escolha do modelo de memória, no projeto de sistemas em chip, de um modo geral e MPSoCs em particular. Outros aspectos também precisam ser levados em consideração para manter o funcionamento correto e consistente das operações de acesso à memória. Este trabalho leva em consideração a coerência e consistência de memória. Estas propriedades são necessárias para garantir a precisão das operações de leitura e escrita na memória e podem gerar impacto no desempenho de sistemas-em-chip.

STORM é uma plataforma virtual baseada em NoC, desenvolvida em SystemC. Por ser uma plataforma, STORM não possui uma arquitetura definida, mas sim um conjunto de módulos de hardware e especificações sobre como devem ser utilizados, além de bibliotecas de software, sendo possível para o projetista criar diversas instâncias de arquiteturas com diferentes características. Isto não só está de acordo com a atual tendência de projeto baseado em plataforma, como também permite a integração

de diferentes módulos e uma maior facilidade na exploração de espaço de projeto. Quanto aos módulos de hardware, STORM fornece um processador (SPARC V8), caches de instruções e dados, módulos de memória, módulo de diretório para manutenção da coerência das caches, um modelo de rede-em-chip (mesh 2D) e módulos de interface com a NoC. Estas características permitem ao projetista criar múltiplas instâncias de arquiteturas com diferentes propriedades para executar aplicações de propósito geral ou específico.

Atualmente, dois modelos de memória estão disponíveis. O primeiro é o modelo de memória compartilhada. Neste modelo um ou mais módulos de memória, fisicamente distribuídos entre os nós da NoC, fornecem um espaço de endereçamento único. Deste modo, todos os processadores no sistema podem acessar qualquer endereço de memória transparentemente, com diferentes custos de latência. Este modelo de memória oferece um modelo de programação baseado em variáveis compartilhadas. O segundo modelo de memória disponível na plataforma STORM é o modelo de memória distribuída. Neste modelo cada nó da NoC tem um processador com seu próprio módulo de memória. Um processador tem acesso apenas ao espaço de endereçamento fornecido pelo seu módulo de memória. A comunicação entre processadores é feita através de troca de mensagens.

As implementações dos nós do modelo de memória compartilhada e do modelo de memória distribuída podem ser vistas na Figura 1.a e Figura 1.b, respectivamente. Nessa figura é possível ver os módulos básicos de hardware da plataforma STORM, processador e NoC, sendo possível também ver os módulos que compõem a interface e fornecem as funcionalidades do protocolo de comunicação entre o processador, a NoC e a memória.

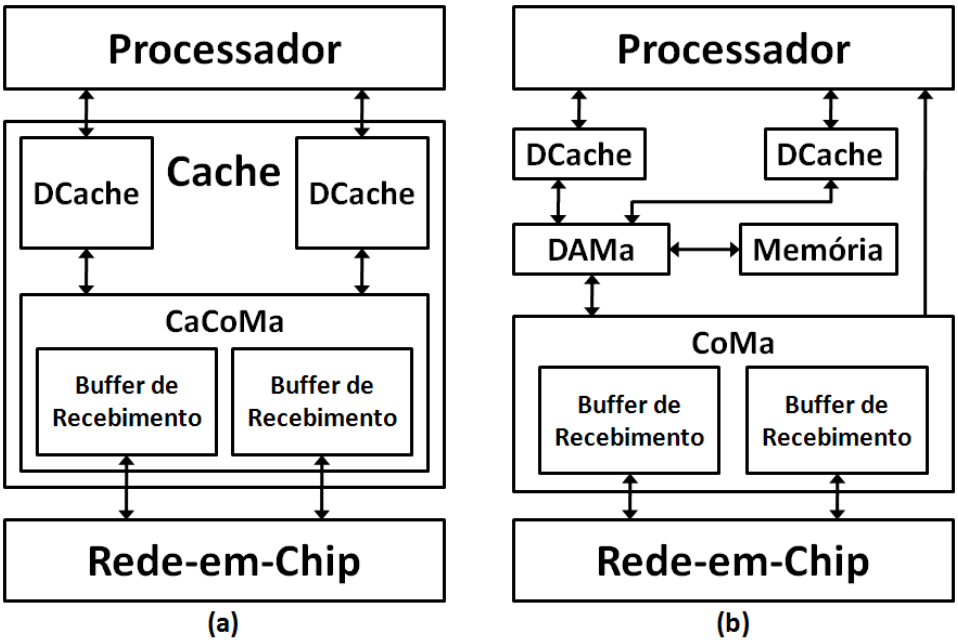

Figura 1. Nós da plataforma STORM: (a) memória compartilhada, (b) memória distribuída.

No modelo de memória compartilhada, Figura 1.a, o CaCoMa (Cache Communication Manager) é responsável por enviar e receber pacotes através da NoC. Ele também é responsável pela tradução de endereços lógicos (endereços de memória) em endereços físicos (endereço de NoC do nó de memória correspondente ao endereço

de memória requisitado). O processo de tradução utiliza uma tabela de endereços (ATA – Address Table). A ATA associa um endereço físico ao último endereço lógico armazenado por um módulo de memória. Assim, para traduzir um endereço lógico X para endereço físico, o CaCoMa deve percorrer as entradas de sua ATA e retornar o endereço físico da primeira entrada cujo valor do último endereço lógico é maior que X. Um exemplo de ATA é mostrado na Tabela 1.

Tabela 1. Exemplo de ATA.

Endereço Lógico	Endereço Físico
0x000007FF	0,1
0x00000FFF	1,0
0x000017FF	1,2

No modelo de memória distribuída (Figura 1.b) a interface entre o processador, a memória e a NoC é feita pelos módulos DAMa e CoMa. O DAMa (Data Access Manager) fornece uma interface entre as caches (ICache e DCache) e os outros módulos no nó. Para cada módulo ligado ao DAMa é atribuída uma faixa de endereços de memória. Ao receber uma requisição da cache, o DAMa verifica a qual módulo pertence o endereço acessado, realiza a comunicação com o módulo correspondente e repassa os dados para a cache. Quando necessário, o DAMa pode enviar junto com os dados um sinal para avisar a cache que os dados não devem ser memorizados, mas apenas repassados ao processador. O COMa (Communication Manager) é responsável por enviar e receber pacotes através da NoC. Como pode ser visto na figura 1.b, o CoMa possui buffers de envio e recebimento. O tamanho de cada buffer é configurável e independente um do outro. Um sinal de interrupção é gerado sempre que o buffer de recebimento do CoMa atinge um determinado nível. Esse nível é configurável e deve ser ajustado de forma a evitar que o buffer de recebimento encha e acabe sobrecarregando os buffers dos roteadores da NoC.

2.1. Programabilidade da plataforma STORM

A plataforma STORM não conta atualmente com um sistema operacional, dessa forma, uma biblioteca em C foi implementada para fornecer comunicação e sincronização em nível de usuário. No modelo de memória distribuída, a comunicação entre os processadores ocorre por troca de mensagens. Assim, um subconjunto das funções do padrão MPI (Message Passing Interface) foi implementado. Este subconjunto (Tabela 2) é suficiente para implementar uma grande quantidade de aplicações uma vez que todos os elementos básicos necessários para comunicação entre processos estão inclusos. Apesar de não ser completo, esse subconjunto pode ser facilmente expandido, uma vez que todos os tipos básicos e constantes especificadas pelo padrão MPI foram implementadas. A utilização do padrão MPI permite também que aplicações existentes sejam portadas para a plataforma STORM com pouco ou nenhum esforço.

No modelo de memória compartilhada, a comunicação é realizada através de variáveis compartilhadas (variáveis globais) sincronizadas por mutexes. Dessa forma, toda sincronização deve ser realizada de forma explícita pelo programador, tornando a tarefa de portar aplicações para este modelo difícil e passível de erros.

Tabela 2. Operações do padrão MPI implementadas e suas descrições

Operação	Descrição
MPI_Comm_size	Retorna o número de processadores em um grupo
MPI_Comm_rank	Retorna o rank de um processador em um grupo
MPI_Init	Inicializa o ambiente MPI
MPI_Finalize	Finaliza o ambiente MPI
MPI_Recv	Recebe uma mensagem de outro nó da rede
MPI_Send	Envia uma mensagem a outro nó da rede
MPI_Bcast	Envia uma mensagem de um processador raiz para todos os outros processadores em um grupo
MPI_Reduce	Combina os dados fornecidos por cada processador de um grupo em um dado único
MPI_Pack	Empacota dados em uma região contígua de memória.
MPI_Unpack	Desempacota dados contidos em uma região de memória contígua de acordo com o tipo de dado fornecido
MPI_Pack_size	Retorna a quantidade de bytes necessários para empacotar um determinado dado
MPI_Op_create	Cria uma função de combinação definida pelo usuário.

3. Estudo de caso

A aplicação escolhida para os experimentos é a simulação de reservatórios de petróleo, que consiste de um modelo matemático cujo objetivo é gerar previsões precisas do comportamento dos fluidos (água, gás e óleo) em um reservatório de petróleo (meio poroso) durante um determinado espaço de tempo e sob diferentes condições de operação [Ertekin, Abou-Kassen e King 2001]. Esse modelo tem como base a teoria de fluxo em meios porosos [Soares 2002]

Este trabalho utiliza o modelo de reservatório black-oil, um dos modelos mais comum entre simuladores comerciais de reservatórios [Fachi, Harpole e Bujnowski 1982], [CMG 1995], [Schlumberger 2009]. Este modelo leva em consideração três componentes e três fases: óleo, água e gás. As equações de fluxo para o modelo black-oil são obtidas através das equações de conservação de massa, equações de estado e a lei de Darcy, que descreve o fluxo de fluidos através de um meio poroso.

A precisão da simulação de reservatório é diretamente proporcional a quantidade de características do reservatório que foram utilizadas para obter o modelo matemático. No entanto, o uso de um grande número de características também aumenta o custo computacional da simulação. Assim é importante encontrar um equilíbrio entre precisão e o custo computacional.

As complexas equações resultantes do modelo matemático gerado não podem ser resolvidas analiticamente, sendo necessária a utilização de técnicas de discretização numérica para obter uma solução aproximada. No processo de discretização o intervalo de tempo é divido em subintervalos de valores fixos e a geometria do reservatório é, semelhantemente, divida em uma malha de blocos de tamanhos iguais. As técnicas de discretização do espaço e tempo geram, para cada bloco da malha, um sistema de equações lineares que precisa ser resolvido para cada subintervalo de tempo. De acordo com [Silva et al 2003], cerca de 80% do tempo total de execução de um simulador de reservatório é gasto na etapa de solução do sistema de equações lineares. Por isso, é de fundamental importância empregar métodos de solução eficientes, capazes de resolver o problema rapidamente, utilizando pouca memória, sem perder a precisão na solução.

O método de resolução de sistema de equações lineares utilizado em nossos experimentos é o succesive over-relaxation (SOR). Este método foi considerado por ter uma rápida convergência e ser adotado por simuladores comerciais de reservatório largamente utilizados, como o Black Oil Applied Simulation Tool [Fachi, Harpole e Bujnowski 1982].

As Equações 1 e 2 descrevem o método SOR:

$$x_i^{k+1} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j<i}^{n} a_{i,j}x_j^{(k+1)} - \sum_{j>i}^{n} a_{i,j}x_j^{(k)}\right) \qquad i = 1, 2, \dots, n \qquad \textbf{(1)}$$

$$x_i^{k+1} = \omega x_i^{k+1} + (1-\omega)x_i^k \qquad \textbf{(2)}$$

Onde:

- a é um elemento da matriz de coeficientes A ($A\bar{x} = \bar{b}$);
- $\bar{x}$ e $\bar{b}$ são elementos dos vetores do sistema de equações lineares;
- x_i^{k+1} é o componente da iteração $k+1$, encontrado em função dos demais elementos da iteração anterior, k;
- ω é o fator de sobre-relaxação, o uso do fator adequado acelera a convergência do método, reduzindo o número de iterações necessárias para resolver o sistema.

A cada iteração, as Equações 1 e 2 são usadas para encontrar o novo valor de cada uma das n variáveis do sistema. Assim o algoritmo pode ser paralelizado de maneira que cada processador p seja responsável por encontrar, a cada iteração, o novo valor de n/p variáveis do sistema.

4. Resultados

Os resultados apresentados aqui correspondem a *n* processadores executando um algoritmo de simulação de reservatório em instancias da STORM configuradas com os modelos de memória distribuída ou compartilhada.

No modelo de memória compartilhada os processadores se comunicam através de mutexes: cada vez que processador quer se comunicar com outro processador, ele bloqueia até que o mutex pertencente ao destino seja liberado. Por outro lado, cada vez que um processador termina suas operações de processamento ele libera seu mutex, permitindo que os outros processadores se comuniquem com ele.

No modelo de memória distribuída, os processadores se comunicam exclusivamente por troca de mensagens, assim, cada vez que um processador precisa de informações de outros processadores, uma comunicação é estabelecida. Para isso, uma biblioteca MPI com funções bloqueantes é utilizada.

A Figura 2.a mostra os resultados de carga injetada na NoC para cada modelo de memória. Como esperado, o modelo de memória compartilhada produz muito mais tráfego na NoC quando comparada ao modelo de memória distribuída, já que todos os dados, inclusive os não compartilhados, são armazenados em módulos de memória distribuídos pela NoC.

Na Figura 2.b, a carga injetada para os dois modelos de memória pode ser observada em diferentes escalas. A escala do lado esquerdo da figura (Megabytes) corresponde ao modelo de memória compartilhada, enquanto que a escala do lado direito da figura (Kilobytes) corresponde ao modelo de memória distribuída. Nessa figura fica claro que, apesar de ser baixa, a carga gerada pelo modelo de memória distribuída se comporta de forma semelhante à carga gerada pelo modelo de memória compartilhada.

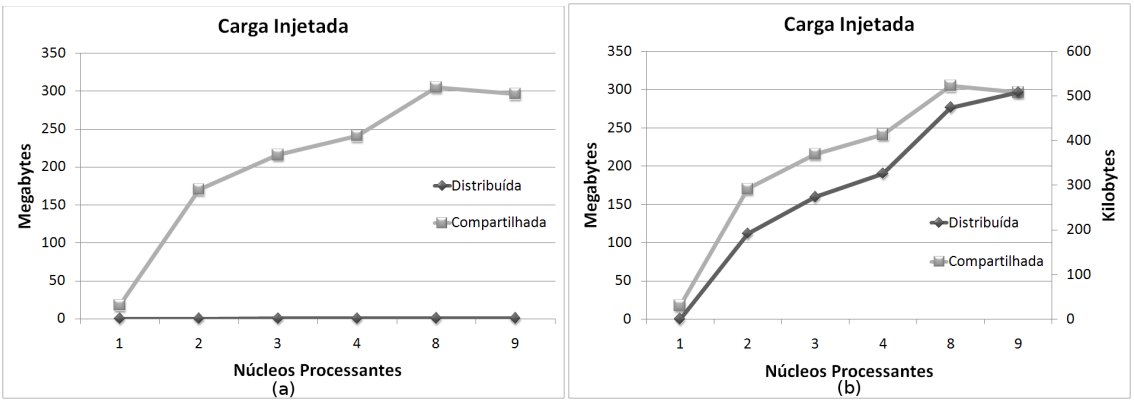

Figura 2. Carga injetada na NoC: (a) mesma escala, (b) escalas diferentes.

A Figura 3.a mostra a carga injetada apenas para o modelo de memória compartilhada. Essa carga está dividida em carga gerada por dados da aplicação e carga gerada para manter a coerência das caches do sistema. Como pode ser observado, a carga gerada por dados da aplicação supera em muito a carga gerada para manter a coerência das caches. Como mostrado em [Girão et al 2007], este resultado pode ser explicado pelo fato dos pacotes de coerência enviarem apenas pacotes de controle que são bem menores que os pacotes de dados transferido pela aplicação. Dessa forma, a linha que representa os dados injetados na NoC pela aplicação na Figura 3.b tem forma semelhante a linha da carga total (dados e coerência) injetada pelo modelo de memória compartilhada na Figura 2.a.

Na figura 3.b, o número de ciclos por instruções (CPI) é comparado para os dois modelos de memória. Como pode ser visto, o número de ciclos por instrução é consideravelmente menor no modelo de memória distribuída. Esse comportamento pode ser explicado pela concorrência no acesso aos dados que é característico do modelo de memória compartilhada.

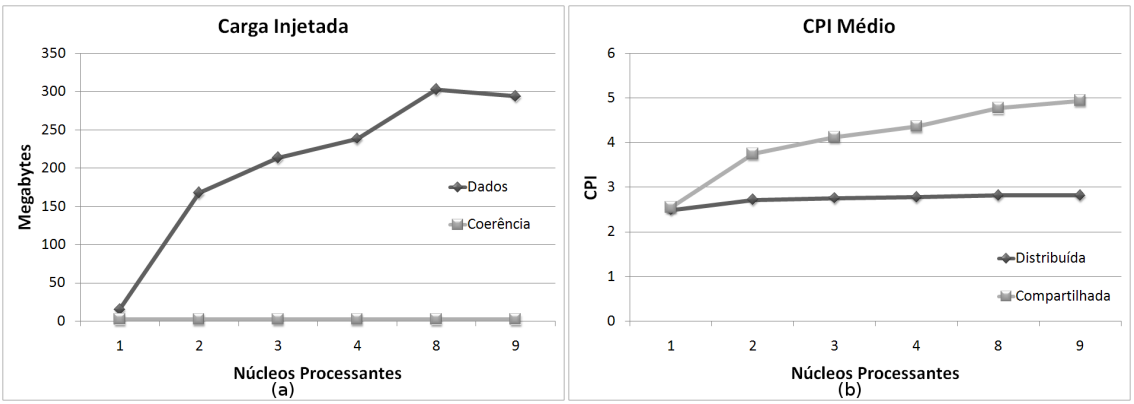

Figura 3. Carga Injetada (a) e CPI Médio (b)

A quantidade de ciclos que cada instância simulada levou para executar a aplicação é exibida na Figura 4.a. Mesmo os resultados de número de ciclos por instrução e carga injetada exibidos anteriormente sendo favoráveis ao modelo de memória distribuída, podemos notar que a quantidade de ciclos é semelhante em ambos os modelos de memória simulados. Isso pode ser creditado ao overhead causado pela execução da biblioteca de troca de mensagens (MPI), pois, como podemos observar na Figura 4.b, a quantidade média de instruções executadas por processador é maior no modelo de memória distribuída.

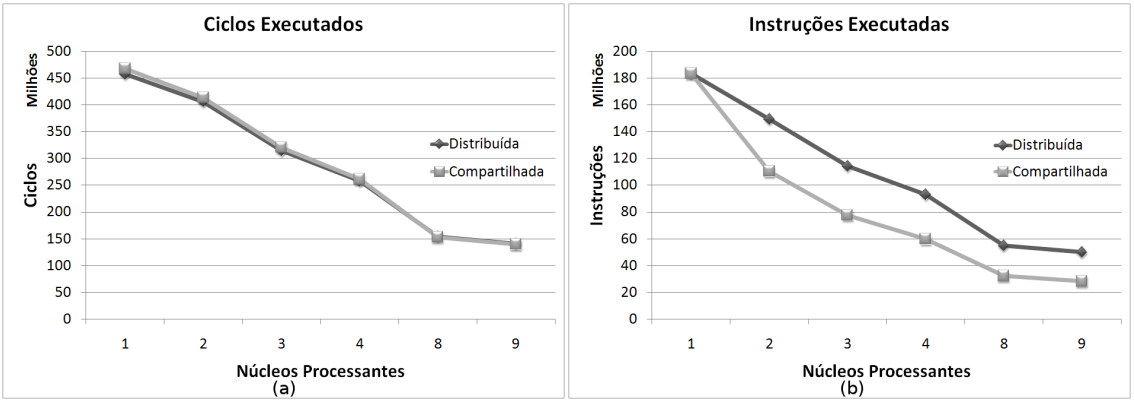

Figura 4. (a) Ciclos e (b) instruções executadas.

Os resultados de latência média dos pacotes são exibidos na Figura 5. Esses valores foram obtidos calculando a média da latência dos pacotes que chegam a cada roteador, e em seguida calculando a média desses valores. Podemos ver no gráfico que a latência média dos pacotes no modelo de memória distribuída é aproximadamente três vezes maior que a latência média do modelo de memória compartilhada. Esse fato ocorre por uma combinação de dois fatores: o chaveamento store-and-forward utilizado pela NoC, e o tamanho dos pacotes de NoC

No modelo de memória compartilhada, o tráfego é totalmente gerado pelos pedidos de dados das caches. Dessa forma, o tamanho médio do pacote de NoC é proporcional ao tamanho do bloco de cache. Já no modelo de memória distribuída, o tamanho do pacote depende do tamanho das mensagens enviadas pela aplicação. Como na aplicação simulada é necessária a troca de vetores entre os processadores do sistema, os pacotes de NoC tendem a ser maiores no modelo de memória distribuída. No

chaveamento store-and-forward os pacotes devem ser armazenados inteiramente nos roteadores intermediários. Assim, a latência tende a aumentar quando o tamanho do pacote aumenta.

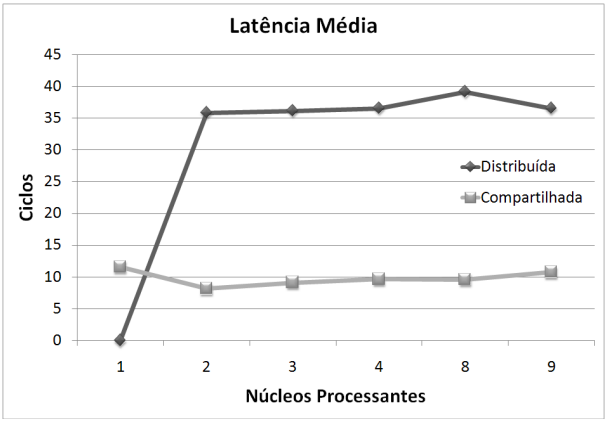

Figura 5. Latência média dos pacotes da NoC.

5. Conclusões e Trabalhos Futuros

Este artigo apresentou o projeto de uma plataforma MPSoC virtual que propõe a utilização de um conjunto de requisitos funcionais, módulos de hardware e bibliotecas de software. Tal projeto provê flexibilidade quando novas instâncias arquiteturais precisam ser desenvolvidas. Diferentemente de muitos artigos sobre MPSoC na literatura, este apresentou uma aproximação de propósito geral. A aplicação utilizada como estudo de caso foi uma simulação de reservatório computacionalmente intensiva. Esta aplicação foi simulada utilizando um sistema com 200 equações lineares

A aplicação foi executada com os modelos de memória compartilhada e distribuída disponíveis na plataforma STORM. As simulações foram realizadas considerando a carga injetada na NoC, o CPI médio, o número de ciclos e instruções executadas e a latência dos pacotes de comunicação da NoC.

Os resultados mostraram que o modelo de memória distribuída apresenta melhores resultados quando a carga injetada e o CPI médio são considerados. Por outro lado, o modelo de memória compartilhada foi melhor quanto a latência dos pacotes na NoC. Esse último resultado pode ser explicado pelo tamanho dos pacotes injetados na NoC pela aplicação. Enquanto no modelo de memória compartilhada os pacotes são do tamanho aproximado do tamanho dos blocos da cache, no modelo de memória distribuída os pacotes são proporcionais ao tamanho das mensagens enviadas pela aplicação. Esses resultados mostraram a relevância de considerar a organização da memória durante a concepção de um MPSoC, uma vez que ela impacta na comunicação (latência) e processamento (CPI, número de instruções executadas).

Como trabalhos futuros nós planejamos testar outras aplicações e observar o impacto em outras caracteristicas, como consumo de energia, quando arquiteturas de processamento e comunicação são combinadas com diferentes organizações de memórias para criar MPSoCs.

Referências

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.

CMG Ltd. "IMEX Reference Manual", version 96.00, [S.l.: s.n.], 1995.

Ertekin, T., Abou-Kassem, J. H. e King, G. R. "Basic applied reservoir simulation". SPE Textbook Series Vol. 7. Richardson: SPE, 2001.

Fanchi, J. R., Harpole, K. J. e Bujnowski, S. W. "BOAST: A three-dimensional, three-phase black oil applied simulation tool". Oklahoma: [s.n.], 1982.

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

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

Girão, G. et al. "Cache Coherency Communication Cost in a NoC-based MPSoC Platform". In: Symposium on Integrated Circuits and Systems Design, 20, 2007, Rio de Janeiro. Proceedings... New York : Association for Computing Machinery, 2007. v. 1. p. 288-293.

Petrot, F., Greiner, A. e 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.

Schlumblerger Ltd. "ECLIPSE Reservoir Engineering Software". Disponível em: http://www.slb.com. Abril.

Silva, F. A. et al. "Parallelizing Black Oil Reservoir Simulation Systems for SMP Machines". In: Annual Symposium on Simulation, 36, 2003, Proceedings... Washington: IEEE Computer Society, 2003, p. 224 - 230.

Soares, A. A. M. "Simulação de reservatórios de petróleo em arquiteturas paralelas com memória distribuída". 2002. Dissertação (Mestrado em Ciências e Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil. UFPE, Recife, 2002.

Suh, T., Blough, D. M. e 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 pp.1150 - 1155 Vol.2