Scratchpad Memories for Parallel Applications in Multi-core Architectures *

Francis Birck Moreira, Eduardo Henrique Molina da Cruz,
Marco Antonio Zanata Alves, Philippe Olivier Alexandre Navaux
Grupo de Processamento Paralelo e Distribuído
Instituto de Informática
Universidade Federal do Rio Grande do Sul
{fbmoreira, ehmcruz, mazalves, navaux}@inf.ufrgs.br

Abstract

Scratchpad memories are largely used in embedded processors due to their reduced energy consumption and area compared to traditional cache memories. In multi-core architectures, these memories are an interesting solution for the storage of shared data and data which is used intensively. However, these memories present some challenges, such as the need for manual choice of the content. Furthermore, different sizes of scratchpad memories result in the need to modify the source code of the application. In this article, we propose the use of a scratchpad memory in a multi-core architecture which alleviates these disadvantages. We added the scratchpad to an architecture consisting of 4 cores, reducing the size of L2 cache in order to give chip area to the scratchpad memory. We evaluated our proposed design by executing the NAS Parallel Benchmark (NPB) applications in a simulator. We improved performance by up to 45% compared to the base architecture, reducing cache invalidations by up to 85%.

1 Introdução

Arquiteturas *multi-core* são o foco do estado da arte no desenvolvimento de novas arquiteturas de processadores. Nestas arquiteturas, há um aumento substancial na quantidade de transações de dados entre a memória primária e o processador devido ao crescente aumento no número de núcleos de processamento. Assim, a crescente diferença de desempenho entre a memória principal e o processador ressalta o sub-sistema de memória como um dos principais gargalos das novas arquiteturas *multi-core* [1]. Logo, a utilização de memórias *cache* vem sendo empregada de maneira a mascarar a latência da memória primária, valendo-se de vários níveis de hierarquia e mecanismos avançados, como *pipelining* e *prefetching*.

Entretanto, dados compartilhados por diferentes memórias *cache* podem acarretar perda de desempenho [4]. Um dos principais motivos para isto é a replicação descontrolada de dados em diferentes *caches*, o que diminui o espaço útil da memória *cache*. Os protocolos de coerência de memória *cache*, necessários para validação de acessos concorrentes, também impõem sobrecargas em sistemas com múltiplas *caches*. Isto ocorre porque escritas em linhas de *cache* geram invalidações em outras memórias *caches* que compartilham a linha sendo escrita [3].

A memória *scratchpad* (*Scratchpad Memory* - SPM) [6] é uma memória especializada visível ao processador e o sistema operacional. Nas arquiteturas *multi-core*, tal memória pode ser compartilhada entre os núcleos de processamento, provendo acesso eficiente a dados compartilhados. O *scratchpad* representa uma extensão ao espaço de armazenamento presente nas arquiteturas *multi-core*, sendo que o controle dos dados que o populam é feito via *software*.

Pesquisas com o *scratchpad* indicam ganhos principalmente em termos de área e energia [13]. Isto deve-se ao fato de implementações de memórias *scratchpad* não utilizarem *tags* e comparadores, que são necessários em memórias *cache* tradicionais. Além disso, uma memória *scratchpad* compartilhada não requer protocolos de coerência, já que todos os núcleos operam sobre os mesmos dados. Tais características tornam o uso do *scratchpad* interessante em sistemas embarcados, onde o consumo de potência e área do chip são muitas vezes mais importantes que o desempenho.

Neste contexto, torna-se interessante avaliar o uso da memória *scratchpad* em um sistema *multi-core* de propósito geral. Aliada à memória *cache* tradicional, o *scratchpad* representa uma solução eficiente para reduzir o consumo

*Trabalho parcialmente apoiado pelo CNPq e CAPES.

de energia da hierarquia de memória como um todo. Ele provê um espaço de armazenamento eficiente para estruturas de dados determinadas pelo programador, compilador ou sistema operacional, o que acarreta ganho de desempenho dada uma escolha de dados inteligente, priorizando dados frequentemente acessados.

O objetivo deste trabalho é propor e avaliar o desempenho de sistemas *multi-core* com o uso de memórias *scratchpad*. Para isto, desenvolveu-se uma simulação de memória *scratchpad* em um ambiente *multi-core* formado de 4 núcleos de processamento. O *scratchpad* foi implementado no simulador Simics [9], integrando-se como um componente da hierarquia de memória, modelado de maneira inovadora ao considerar um controle de páginas no *Translation Look-Aside Buffer*(TLB). As aplicações do *benchmark* NAS Parallel Benchmarks (NPB) [7] serviram de base para os experimentos, sendo que seus dados compartilhados e frequentemente acessados pelos múltiplos núcleos foram armazenados no *scratchpad*. A escolha de dados das aplicações a serem alocados no *scratchpad* foi feita com o auxilio da ferramenta VTune [12] da Intel, a qual pode ser utilizada em uma vasta gama de aplicações paralelas, assim auxiliando o programador na escolha dos dados mais importantes a serem mapeados no *scratchpad*.

Este artigo está organizado da seguinte maneira. A seção 2 apresenta os trabalhos relacionados a área. Na Seção 3, é descrita uma modelagem para o uso da memória *scratchpad* em uma arquitetura *multi-core*. A Seção 4 explica a implementação da arquitetura no simulador. Os detalhes da carga de trabalho utilizada são mostrados na Seção 5. A Seção 6 traz a análise sobre os resultados obtidos nos experimentos. Finalmente, a Seção 7 traça as conclusões do artigo.

2 Trabalhos Relacionados

No trabalho de Banakar et al. [2], são apresentados comparativos entre utilização da memória *scratchpad* e a memória *cache* para sistemas embarcados. Os quesitos mais importantes apontados no estudo foram área e energia, onde demonstrou-se superioridade relevante da memória *scratchpad*. Entretanto, a programação dos aplicativos na arquitetura apresentada precisa ser feita em baixo nível, onde o programador necessita gerenciar diretamente o conteúdo da memória *scratchpad*.

O artigo apresentado por Poletti et al.[5] investiga modos eficazes de implementar a arquitetura de memória *scratchpad* em um ambiente embarcado *multi-core* sem memória *cache*. Neste contexto, foi avaliado o uso de um mecanismo de *Direct Memory Access*, o qual permite alocação eficiente da memória *scratchpad* em tempo de execução. Além disso, foi descrito também uma interface para o uso da memória *scratchpad* para ajudar na programação envolvendo o mesmo. Porém, neste trabalho, o endereçamento da memória depende do tamanho da memória *scratchpad* e é feito de modo direto, implicando em dificuldade para usar funções e bibliotecas pré-existentes.

O uso de uma memória *scratchpad* em um ambiente *multi-core* também foi explorado por Yanamandra et al. [14], utilizando uma memória *scratchpad* para cálculos envolvendo matrizes esparsas. O artigo demonstra a comparação de eficiência com dois experimentos base. No primeiro, parte da memória *cache* L1 é removida para dar lugar a uma memória *scratchpad*; no segundo experimento, a mesma estratégia é adotada na memória *cache* L2. Os resultados, em termos de desempenho e energia, se mostram favoráveis para a adoção da memória *scratchpad* no segundo nível da hierarquia de memória. Na descrição do método de simulação da memória *scratchpad*, é citado o uso do simulador Simics, as latências e as modificações de tamanhos nas *caches*. Entretanto, não são apresentados detalhes sobre como a memória foi simulada nem sobre como eram feitos os acessos ao *scratchpad*.

Nguyen et al.[11] demonstra uma técnica para resolução do problema de retrocompatibilidade ao criar um instalador de *software* customizado, o qual decide a alocação da memória *scratchpad* exatamente antes da primeira execução do programa. Determinado o tamanho da memória *scratchpad*, o instalador modifica o executável de acordo com a alocação escolhida. Esta técnica foi aplicada em sistemas embarcados, no qual o uso de um instalador é pertinente, já que atualizações de *softwares* embarcados são incomuns.

Comparando os trabalhos correlatos, pode-se ressaltar que este estudo considera um conjunto maior de aplicações paralelas, onde utiliza-se uma memória *scratchpad* compartilhada entre diversos núcleos, a fim de oferecer espaço de memória com acesso eficiente. Assim, reduz-se o tempo de comunicação entre as diversas *threads* das aplicações paralelas, bem como o tempo de acesso às estruturas mais utilizadas pelas aplicações.

3 Proposta de uma memória *scratchpad* em arquiteturas *multi-core*

A memória *scratchpad* foi modelada para oferecer uma memória rápida e paralela à hierarquia de memória *cache*. Para isso, antes de enviar uma transação para a hierarquia de memória *cache*, todos os endereços requisitados devem ser verificados para saber se o mesmo encontra-se no *scratchpad*. Caso o endereço esteja no *scratchpad*, o mesmo é responsável por tratar a transação. Caso contrário, a transação é repassada para a hierarquia de memória *cache*. Para a manipulação de quais estruturas de dados estão presentes na memória *scratchpad*, são usadas chamadas de sistema especificando quais faixa de endereços estão armazenados

nesta memória. Dessa forma, a análise de endereços presentes no *scratchpad* se resume em uma comparação de faixas de endereços.

Deste modo, utiliza-se a memória *scratchpad* como um armazenamento específico para estruturas de dados compartilhadas que são frequentemente usadas. Objetiva-se ganho de desempenho ao fazer uma seleção astuta dos dados mais usados, já que haverá diminuição de dados replicados e, por consequência, menor sobrecarga pelos protocolos de coerência de *cache*. Na manipulação do conteúdo presente na memória *scratchpad* por parte dos programas, os endereços são acessados como se fossem endereços normais, pois estes são traduzidos para o espaço da memória *scratchpad* automaticamente pelo *hardware*.

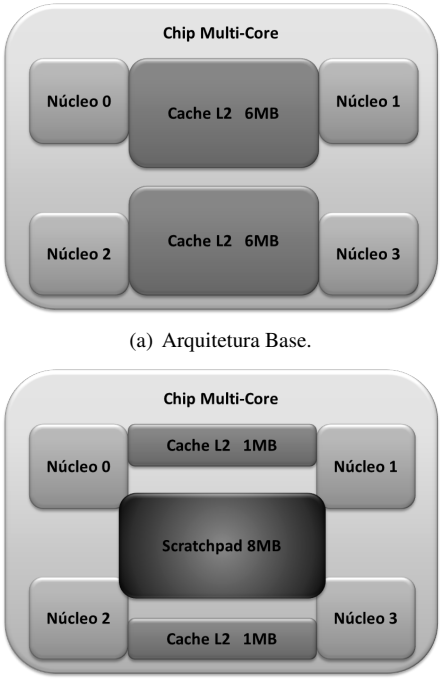
Assim é possível usar uma memória *scratchpad* que se adapta às necessidades de uma arquitetura *multi-core* genérica. Há retrocompatibilidade ao se mapear o espaço de endereçamento da memória *scratchpad* em faixas de endereçamento da memória primária. Como a tradução é implementada em *hardware*, poucas alterações são necessárias nos códigos das aplicações.

Com este modelo, em uma memória *scratchpad* compartilhada por múltiplos núcleos, verificamos a necessidade de especificarmos um mecanismo para gerenciar a chegada de múltiplas requisições. Para isto, foi utilizado um modelo simples com N portas paralelas, atrasando requisições caso nenhuma porta esteja disponível. Desta forma, foi possível experimentar diferentes números de portas (1-Porta, 2-Portas e 4-Portas de acesso), de modo a verificar o problema de contenção de acessos à memória *scratchpad*, e assim determinar um número de portas adequado para o sistema avaliado.

4 Simulação e Parâmetros

O simulador Simics foi empregado para simular um sistema *multi-core* com 4 núcleos de processamento e uma organização de hierarquia de memória semelhante à encontrada na arquitetura *Harpertown*, conforme ilustrado na Figura 1. Utilizando este sistema como base, duas arquiteturas foram simuladas: a primeira arquitetura base sem a memória *scratchpad*, (Figura 1(a)) e a segunda arquitetura com a memória *scratchpad* (Figura 1(b)). Os núcleos simulados no Simics são baseados em processadores UltraSPARC com o conjunto de instruções V9. Para uma comparação justa, foi reduzida a capacidade das memórias *cache* L2 no ambiente que utilizou a memória *scratchpad*. A Tabela 1 sumariza os parâmetros da hierarquia de memória, ressaltando que os parâmetros de latência das memórias foram gerados utilizando a ferramenta CACTI [10], considerando núcleos de processamento UltraSPARC com frequência de 1GHz.

A decisão de projeto que leva a uma memória *scratchpad* de grande capacidade (8 MB) se deve à possibilidade de uso desta por vários programas ou *threads*, sendo que, em sistemas de propósito geral de alto desempenho, diversas aplicações exigem grandes estruturas de dados. Do ponto de vista de implementação física, a diferença de latência e energia entre memórias *cache* e *scratchpad* em memórias pequenas (aproximadamente 16 KB) é próxima de zero, sendo que essa diferença se torna evidente conforme aumenta-se o tamanho dessas estruturas, conforme verificado no CACTI. Portanto, quanto maior o tamanho das memórias *cache* e *scratchpad*, maior a diferença na latência do acesso e no consumo de energia entre elas.



(a) Arquitetura Base.

(b) Arquitetura com SPM.

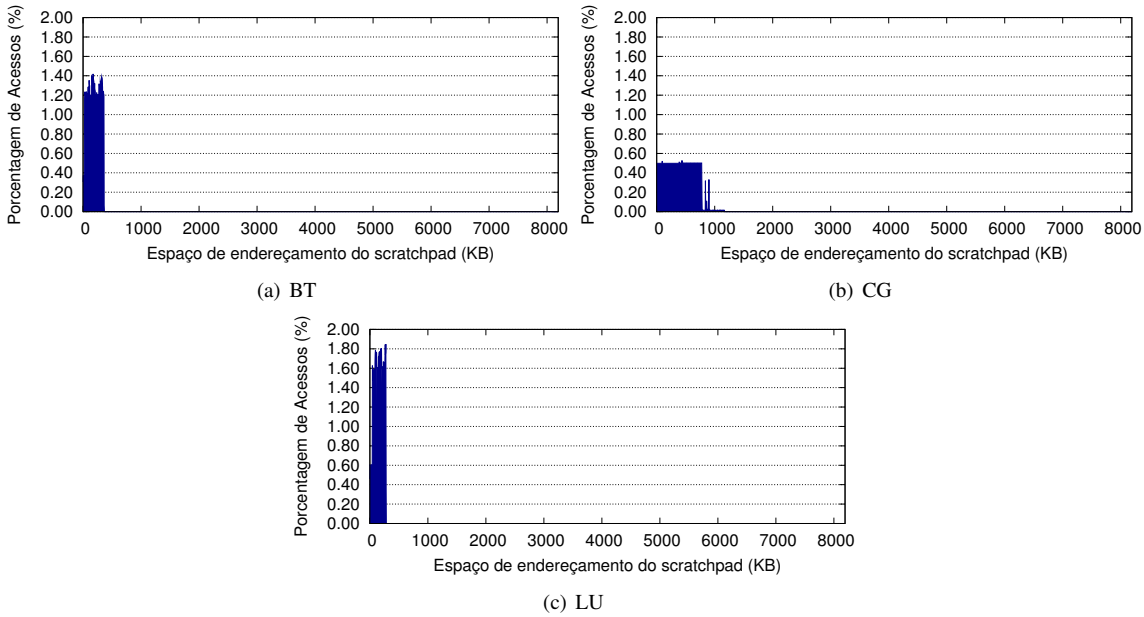
Figura 1. Ilustração das arquiteturas modeladas, 1(a) com apenas memórias *cache* e 1(b) com memórias *cache* e *Scratchpad*.

5 Carga de Trabalho

As aplicações usadas nos experimentos constituem o *Numerical Aerodynamic Simulation Parallel Benchmark* (NAS-NPB), versão 3.3.1, paralelizada com OpenMP. Este *benchmark* é formado por aplicações relacionadas a métodos numéricos de simulações aerodinâmicas para computação científica. Tais aplicações foram projetadas para comparar o desempenho de computadores paralelos, sendo formadas por *kernels* de simulação de dinâmica de fluídos computacionais (CFD - Computational Fluid Dy-

Configuração	Memória *Cache* L1			Memória *Cache* L2			Memória *Scratchpad*		Memória Principal	
	Tamanho da *Cache*	Conjunto Assoc.	Latência (Ciclos)	Tamanho da *Cache*	Conjunto Assoc.	Latência (Ciclos)	Tamanho do SPM	Latência (Ciclos)	Tamanho da RAM	Latência (Ciclos)
Cache normal	32+32 KB	8 Vias	3	6+6 MB	24 Vias	15	-	-	1GB	78
Cache + SPM	32+32 KB	8 Vias	3	1+1 MB	16 Vias	8	8 MB	12	1GB	78
Cache 1MB	32+32 KB	8 Vias	3	1+1 MB	16 Vias	8	8 MB	12	1GB	78
Cache 1MB, 15 ciclos + SPM	32+32 KB	8 Vias	3	1+1 MB	16 Vias	15	8 MB	12	1GB	78

Tabela 1. Parâmetros utilizados na simulação dos experimentos.



(a) BT (b) CG

(c) LU

Figura 2. Histograma de uso das regiões da memória *scratchpad* para as aplicações que utilizaram menos que 4 MB da memória *scratchpad*.

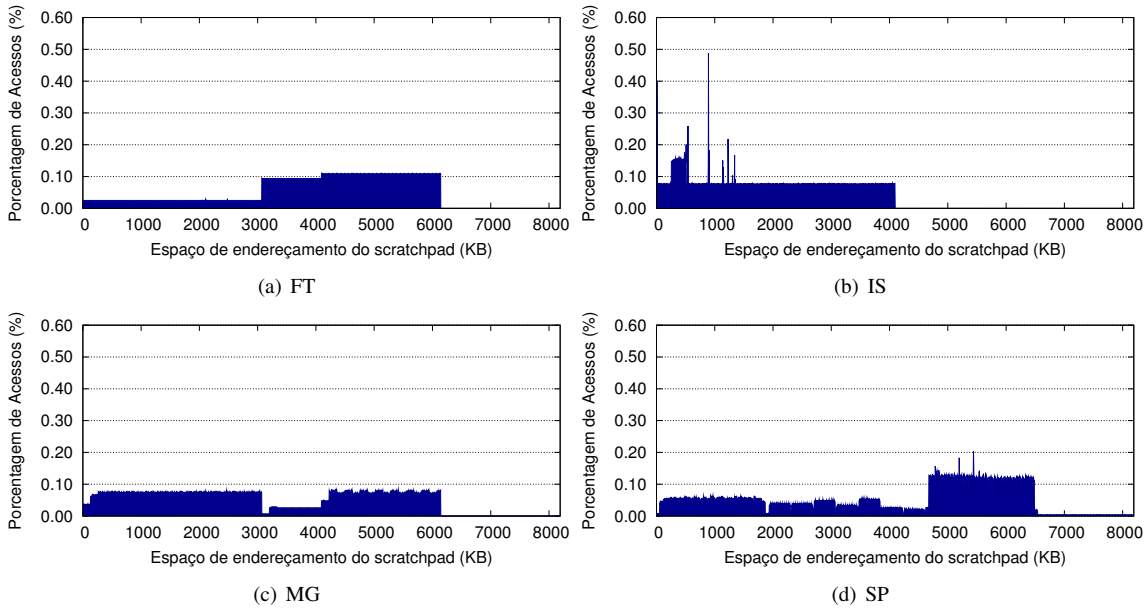
namics) derivados de importantes aplicações das classes aerofísicas, de maneira que as simulações reproduzem grande parte dos movimentos de dados e computações encontrados em códigos de dinâmica de fluídos completos [7].

As seguintes aplicações fazem parte deste estudo: BT (*Block Tridiagonal*), CG (*Conjugate Gradient*), MG (*Multigrid*), EP (*Embarassingly Parallel*), SP (*Scalar Pentadiagonal*), LU (*Lower and Upper Triangular System*), IS (*Integer Sort*), FT (*Fast Fourier Transform*), UA (*Unstructured Adaptive*). Todas essas aplicações executam operações de ponto-flutuante de precisão dupla (64 bits) e são programadas utilizando Fortran-90 (BT, CG, EP, FT, LU, MG, SP, UA), com exceção da aplicação IS, programada em C, efetuando operações sobre inteiros.

As aplicações BT, SP e LU possuem seu paralelismo através da divisão de domínio espacial, com compartilhamento de dados nas bordas de cada subdomínio, apresentando acessos de forma linear aos dados. As aplicações CG e UA apresentam características de acesso randômico aos dados. A aplicação MG trabalha sobre dados contíguos, variando o padrão de acesso a cada etapa de computação, indo de acessos desalinhados até acessos lineares conforme a grade é refinada. Assim, esta aplicação pode ser considerada um mistro entre as aplicações de acesso regular (BT, LU e SP) e as de acesso irregular (CG e UA).

A aplicação EP é completamente paralela e possui poucos acessos à memória. O desempenho dessa aplicação pode ser utilizado como referência de desempenho computacional de pico de uma determinada máquina. A aplicação IS apresenta comportamento contrário a aplicação EP, já que executa muitos acessos a dados compartilhados, além dos acessos lineares aos dados armazenados. Por fim, a aplicação FT apresenta etapas de acesso linear com etapas de acesso compartilhado entre as diversas *threads*.

O NAS-NPB conta com vários tamanhos de entrada para os problemas, S, W, A, B, C (partindo do menor para o maior), sendo que para este trabalho foi utilizado o tamanho W. Este tamanho é o mais indicado para simulações, já que



(a) FT (b) IS

(c) MG (d) SP

Figura 3. Histograma de uso das regiões da memória *scratchpad* para as aplicações que utilizaram mais que 4 MB da memória *scratchpad*.

é classificado como um problema de tamanho real e, mesmo assim, possui um tempo viável de simulação.

6 Resultados experimentais

Para observar e comparar as execuções com a memória *scratchpad* e com a memória *cache* foram feitas algumas medidas. Mediu-se tempo de execução com objetivo de medir o desempenho obtido. Foram obtidas também medidas de uso da memória *scratchpad* para observar o quão efetivamente usada esta foi, podendo-se observar o padrão de acessos nas estruturas alocadas. O número de mensagens de coerência entre as memórias *cache* L2 foi analisado para visualizar o quanto a memória *scratchpad* ajuda a aliviar a sobrecarga destes, melhorando a performance das *caches*. E, por final, é feita uma comparação entre a memória *cache* L2 de tamanho reduzido, com latência de 8 ciclos, sem o uso do *scratchpad*, e memória *cache* L2 com latência de 15 ciclos, com o uso da memória *scratchpad*. O objetivo é comprovar que o uso da memória *scratchpad* é o responsável pela melhoria de performance, e não apenas a redução da latência da memória *cache* L2.

Para a alocação das estruturas de dados nos programas, foi necessário modificar o código dos *benchmarks* usados, de forma a fazer as chamadas ao Simics descrita nas seções 3 e 4 e armazenar as estruturas desejadas na memória *scratchpad*. Para selecionar tais estruturas, foi usado o programa Vtune [12] da Intel, o qual utiliza *hardware performance counters* presentes nos processadores da Intel para fazer *profiling* de aplicações. Com isto, é possível identificar quais estruturas de dados demandam mais tempo de execução, sendo estas as estruturas que foram armazenadas na memória *scratchpad*.

Todas as aplicações foram alteradas de maneira a se realizar uma alocação estática da memória *scratchpad* logo no início da execução. Foram obtidos resultados de todas aplicações do NPB, exceto da aplicação DC (Data Cube Operator), a qual não executaria em tempo viável em ambientes de simulação. São fornecidas as diferenças de tempo de execução para cada *benchmark* e um histograma apresentando a frequência de acesso a cada parcela de endereços dentro da memória *scratchpad*. Isto permite analisar o uso da memória e as diferenças entre as regiões de cada estrutura de dados armazenada.

Observa-se, no *benchmark* BT (*Block Tridiagonal*), o uso de uma parcela pequena da memória *scratchpad*, conforme a Figura 2(a), usando aproximadamente 368 KB da memória *scratchpad*. Ainda assim, os resultados apresentam ganhos de 21,75%, conforme a Figura 4, devido à alta concentração de acessos nesta parcela de endereços. Observando a estrutura do programa, é possível assumir que toda a estrutura principal foi alocada (para o tamanho W, uma matriz tridimensional de 33x33x33) na memória *scratchpad*. Porém, tal estrutura cabe nas memórias *cache* L2 de 1MB, e apenas as reduções de tamanho e latência da memória *cache* L2 provam-se necessárias para ganho de desempenho, como pode ser observado na Figura 5. Há ganho de desempenho ao se utilizar a memória *scratchpad*,

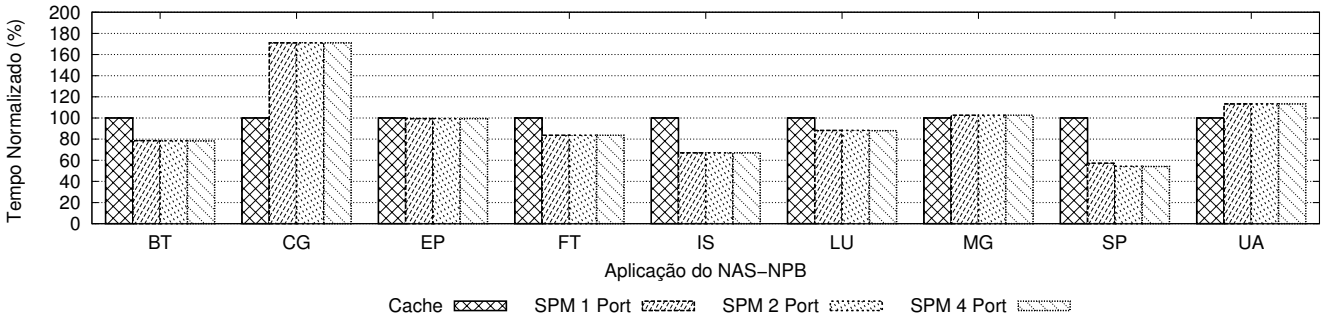


Figura 4. Tempo de execução dos *benchmarks* para configurações com memória *cache* e memória *scratchpad* com 1, 2 e 4 portas.

pois há redução no número de invalidaçõese mensagens de modificação de estado entre as memórias *cache* L2, conforme visto na Figura 6.

O padrão randômico de acessos à memória presente no *benchmark* CG (*Conjugate Gradient*) impõe uma maior sobrecarga no mecanismo de *caching* de dados. Como a *cache* L2 fora reduzida, houve perda de desempenho, pois a memória *scratchpad* não fora utilizada o bastante para compensar a memória *cache* L2 menor. Os tempos de execução do *benchmark* apresentam perda de 71,03% , na Figura 4. A frequência do uso de dados conforme a faixa de endereços alocada se encontra na Figura 2(b). Para o tamanho de entrada W, o *benchmark* utiliza uma matriz esparsa positiva de 7.000 posições e, como pode observar-se na Figura 5, uma arquitetura usando uma memória *cache* de tamanho e latência reduzidos também perde desempenho. Adicionalmente, a Figura 6 mostra que o número de mensagens de coerência entre as memórias *cache* L2 aumentou, comprovando que a faixa de endereços selecionada para a memória *scratchpad* não foi a mais frequentemente acessada, e que as memórias *cache* L2 com 1MB não tem condições de prover dados eficientemente.

O *benchmark* EP (*Embarassingly Parallel*) usa o método polar *Marsaglia* para gerar pares de números randômicos, o que necessita de muito pouca memória. Na Figura 4, é demonstrado que não existe diferença significante entre os tempos de execução. Foi armazenado no *scratchpad* uma faixa de endereços considerada relevante, mas que possui tamanho insignificante perante o tamanho total do *scratchpad*. Os dados alocados na memória *scratchpad* são pouco acessados, gerando nenhum ganho de desempenho. Como observado na Figura 5, o uso da memória *scratchpad* foi insignificante, pois o tempo de execução com uma memória *cache* L2 de 1MB e 15 ciclos de latência apresentou perda de desempenho, mesmo que o número de mensagens de coerência entre as memórias *cache* L2 tenha sido reduzido, conforme a Figura 6.

O *benchmark* FT (*Fast Fourier Transform*) realiza a transformada rápida de Fourier em uma grande estrutura matricial tri-dimensional, com dimensões de 128, 128 e 32. Os acessos são bem distribuídos e regulares, conforme a Figura 3(a). A alocação de grande parte da estrutura na memória *scratchpad* resulta em ganho de desempenho de 18,30%, como observado na Figura 4. Porém, conforme observado na Figura 5, a estrutura principal mais acessada cabe na memória *cache* L2 de 1MB, pois tal ambiente de simulação resulta no mesmo ganho de desempenho em relação ao ambiente usando uma memória *scratchpad*. A Figura 6 demonstra que houve uma grande queda no número de mensagens de coerência entre as memórias *cache* L2, embora o número de mensagens para este *benchmark* seja pequeno, fazendo com que a adição da memória *scratchpad* torne-se de pouca ou nenhuma relevância neste caso.

O *benchmark* IS (*Integer Sort*) realiza a classificação de um vetor de inteiros de $1.048.576$ posições, e apresenta acessos randômicos à memória tal qual o *benchmark* CG. Porém, como foi possível popular mais dados no *scratchpad* em relação ao CG, armazenando a estrutura inteira dentro da memória *scratchpad*, ganhos de desempenho mais expressivos foram obtidos, de 33,12% conforme a Figura 4. A frequência de acesso aos dados é inconstante na estrutura alocada, como pode ser observado na Figura 3(b). A memória *scratchpad* desempenha um papel essencial em tal *benchmark*, como pode ser observado na Figura 5, pois a estrutura principal não cabe nas memórias *cache*. A Figura 6 demonstra a queda nas mensagens de coerência, pois com a memória *scratchpad* não é necessário ficar substituindo partes da estrutura na memória *cache* L2.

O *benchmark* LU (*Lower and Upper Triangular System*) consiste de um algoritmo de resolução de sistemas tal qual o BT, e resolve um sistema de equações não lineares de uma matriz 33x33x33. Ele apresentou ganho de desempenho de 12,15%, conforme a Figura 4, mesmo usando-se apenas uma parcela pequena da memória *scratchpad*, como pode ser observado na Figura 2(c). Igualmente ao BT, a

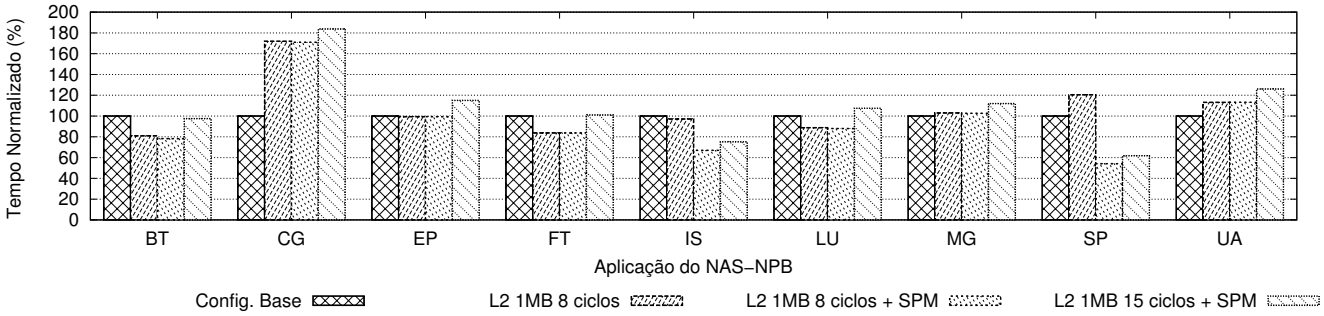
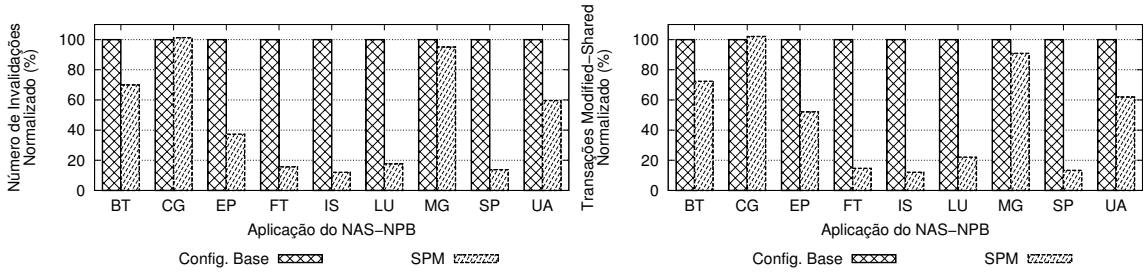


Figura 5. Comparação do *scratchpad* com *cache* normal 6MB, *cache* de 1MB e latência de 8 ciclos, *cache* de 1MB e latência de 8 ciclos com scratchpad e *cache* de 1MB e latência de 15 ciclos com *scratchpad*



(a) Invalidações

(b) Modified to Shared

Figura 6. Quantidade de invalidações e mudanças de estado de *modified* para *shared* detectadas no protocolo MESI.

estrutura inteira cabe nas memórias *cache* L2 de 1MB, o que resulta em ganho de performance semelhante sem o uso de memória *scratchpad*, conforme a Figura 5. A vantagem obtida com a memória *scratchpad* é a redução de mensagens de coerência, observada na Figura 6, a qual gera a diferença de desempenho entre memórias *cache* L2 sem uso de *scratchpad* e com uso de *scratchpad*.

O *benchmark* MG (*Multigrid*) resolve uma equação discreta de *Poisson* tridimensional com dimensões de 128x128x128, com acessos iniciais esparsos onde a distância entre os acessos reduz até torna-los regulares, facilitando o uso de grande parcela da memória *scratchpad*. Pode-se observar na Figura 3(c) que os acessos distribuem-se uniformemente em duas grandes faixas de dados na memória *scratchpad*. Entretanto, não obtêm-se ganho de desempenho, pois o *scratchpad* não tem capacidade para armazenar todos os dados do MG, que aparenta possuir estruturas mais usadas que a matriz principal. Desta forma, a diminuição da memória *cache* de nível 2 impõe uma perda de desempenho superior aos ganho obtidos com o *scratchpad*, como observado na Figura 5. Adicionalmente, pode-se observar pela pequena redução do número de mensagens de coerência na Figura 6, que a seleção das estruturas de dados não foi ideal.

Tal qual os *benchmarks* BT e LU, o *benchmark* SP (*Scalar Pentadiagonal*) resolve de forma paralela um conjunto de sistemas não lineares de uma matriz de dimensões 33x33x33, com acessos regulares. Entretanto, nessa aplicação, a maior parcela do espaço de endereçamento do *scratchpad* foi aproveitado (Figura 3(d)), assim, ganhos mais expressivos, de 45% foram obtidos. Diferentemente dos *benchmarks* BT e LU, o SP aloca três grandes vetores utilizados para manipulação dos dados, os quais foram armazenados na memória *scratchpad*, tornando a memória *scratchpad* essencial ao ganho de desempenho, conforme mostrado na Figura 5. Neste *benchmark*, o número de mensagens de coerência é reduzido em aproximadamente 90%, como visto na Figura 6. Devido ao número intensivo de mensagens trocadas, é notado que a memória *cache* L2 de 1MB, sem o uso de *scratchpad*, sofre perda de desempenho notável tanto por falta de espaço de armazenamento quanto por mensagens de coerência.

O *benchmark* UA (*Unstructured Adaptive*) realiza cálculos de transferência de calor e, assim como os *benchmarks* CG e IS, possui acessos a memória de forma randômica. Porém, este algoritmo faz alocação dinâmica da memória, sendo inapropriado para o método de alocação estática utilizado no presente trabalho. Obteve-se perda de

desempenho de 13,29%, conforme observado na Figura 4, pois alocou-se apenas 4 KB de memória. A Figura 5 comprova a necessidade de memória *cache* L2, e, mesmo a redução de mensagens de coerência, vista na Figura 6, não é suficiente para compensar a perda de desempenho. No caso da aplicação UA, os resultados negativos eram esperados, pois além da falta de estruturas a serem alocadas no *scratchpad*, esta aplicação foi desenvolvida de maneira a se limitar ganhos através da manipulação de conteúdo de memória por compiladores e *hardware* especializado [7].

7 Conclusões

Este trabalho avaliou o potencial do uso de memórias *scratchpad* em arquiteturas *multi-core*. Observa-se ganho de desempenho proporcional ao uso efetivo da memória *scratchpad*. Isto deve-se ao fato de que consegue-se acessar dados compartilhados de forma mais eficientemente do que em memórias *cache*, já que as últimas ocasionam replicações e sofrem com invalidações por protocolos de coerência. O *scratchpad* também apresenta menor latência que a memória *cache* tradicional, pois não necessita de comparação de *tags*.

Conforme os resultados dos vários *benchmarks*, nota-se que a memória *scratchpad* se comporta como uma solução interessante para a grande maioria destes. Ganhos de até 45% foram obtidos, com uma média de 7,47%. Aplicações com pouco uso do *scratchpad*, como o EP e LU, não obtiveram ganho de desempenho, como esperado. Neste trabalho, uma ferramenta de *profiling* de aplicações foi utilizada para facilitar a tomada de decisões sobre quais estruturas de dados seriam armazenadas no *scratchpad* simulado. Foi notável a facilidade de modificação do código, pois a hierarquia de memória modelada no artigo considera a facilidade de programação como um fator importante para o sucesso de uma melhoria em uma máquina *off-the-shelf*, devido ao uso mais variado de tal tipo de máquina.

Para obtenção da melhor performance com os *benchmarks*, estamos analisando a possibilidade de se implementar uma mudança no algoritmo do compilador demonstrado em [8], para utilizar a memória *scratchpad* da maneira mais efetiva possível sem aumentar o esforço do programador. Também é possível a aplicação das ferramentas de *profiling* descritas em [3], que facilitariam a alocação para todos *benchmarks*, em especial o *benchmark* CG. Outro foco de futura pesquisa seriam métodos de definição do conteúdo do *scratchpad* apropriados para alocações dinâmicas de memória, como presente na aplicação UA.

Referências

[1] M. A. Z. Alves, H. C. Freitas, and P. O. A. Navaux. Investigation of shared l2 cache on many-core processors. In *Proceedings Workshop on Many-Core*, pages 21–30, Berlin, 2009. VDE Verlag.

[2] R. Banakar, S. Steinke, B. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, pages 73–78, New York, NY, USA, 2002. ACM.

[3] E. Cruz, M. Alves, and P. Navaux. Process mapping based on memory access traces. In *11th Symposium on Computing Systems*, WSCAD-SCC, pages 72–79, 2010.

[4] E. Cruz, C. P. Ribeiro, M. Alves, A. S. Carissimi, J. Mehaut, and P. Navaux. Process mapping based on memory access traces. In *13th Workshop on Advances in Parallel and Distributed Computational Models*, APDCM / IPDPS, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[5] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J. M. Mendias. An integrated hardware/software approach for run-time scratchpad management. In *Proceedings of the 41st annual Design Automation Conference*, DAC, pages 238–243, New York, NY, USA, 2004. ACM.

[6] B. Jacob, S. Ng, and D. Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann Publishers Inc., 2007.

[7] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. *NASA Ames Research Center, Technical Report NAS-99-011*, 1999.

[8] L. Li, L. Gao, and J. Xue. Memory coloring: a compiler approach for scratchpad memory management. In *Parallel Architectures and Compilation Techniques, 2005. PACT 2005. 14th International Conference on*, pages 329–338, 2005.

[9] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35:50–58, 2002.

[10] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[11] N. Nguyen, A. Dominguez, and R. Barua. Memory allocation for embedded systems with a compile-time-unknown scratch-pad size. *ACM Trans. Embed. Comput. Syst.*, 8:21–32, April 2009.

[12] J. Reinders. *VTune performance analyzer essentials: measurement and tuning techniques for software developers*. Intel Press, 2005.

[13] V. Suhendra, A. Roychoudhury, and T. Mitra. Scratchpad allocation for concurrent embedded software. *ACM Transactions on Programming Languages and Systems*, 32:13–47, April 2010.

[14] A. Yanamandra, B. Cover, P. Raghavan, M. Irwin, and M. Kandemir. Evaluating the role of scratchpad memories in chip multiprocessors for sparse matrix computations. In *IEEE International Symposium on Parallel and Distributed Processing.*, IPDPS, pages 1–10, 2008.