Influência do Compartilhamento de *Cache* L2 em um *Chip* Multiprocessado sob Cargas de Trabalho com Conjuntos de Dados Contíguos e Não Contíguos

Marco A. Z. Alves, Henrique C. Freitas, Flávio R. Wagner, Philippe O. A. Navaux
Grupo de Processamento Paralelo e Distribuído
Laboratório de Sistemas Embarcados
Programa de Pós-Graduação em Computação, Instituto de Informática
Universidade Federal do Rio Grande do Sul
{marco.zanata, hcfreitas, flavio, navaux}@inf.ufrgs.br

Resumo

As diversas pesquisas e projetos de chips multiprocessados apontam na direção do uso da cache L2 compartilhada. No entanto, ainda está muito cedo para definir um modelo de compartilhamento à cache L2 que atenda às necessidades de desempenho dos futuros processadores multi-core. O objetivo deste artigo é apresentar resultados do projeto de um chip multiprocessado com vários agrupamentos de núcleos compartilhando a mesma cache L2 sob cargas de trabalho com conjuntos de dados contíguos e não contíguos. Os resultados apresentados mostram que a proximidade dos dados da aplicação paralela exerce forte influência sobre o desempenho nas diversas organizações de agrupamentos de processadores. Para a carga de trabalho com dados contíguos houve um aumento de desempenho de até 0,93% e uma redução na taxa de faltas da cache L2 de até 85,20% para leitura e aumento da taxa de faltas de até 59,07% para escrita. No conjunto de dados não contíguos houve uma degradação do desempenho de até 0,11%, mas com redução na taxa de faltas da cache L2 de até 82% para leitura e aumento na taxa de faltas para escrita de até 62,07%.

1. Introdução

Vários projetos de arquiteturas de processadores ao longo de décadas vêm adotando técnicas tradicionais [4][5][16][17][18] como *pipeline*, superescalaridade e *multithreading* para explorar o paralelismo de execução das aplicações e assim melhorar o tempo de resposta em computadores pessoais ou servidores.

Basicamente, em um *pipeline* superescalar [5], além do processamento das instruções serem divididos em estágios, é possível executar mais de uma instrução por ciclo, utilizando, para isso, o aumento do número de unidades funcionais e técnicas para solucionar falsas dependências entre as instruções, dentre outras. Desta forma, os processadores superescalares são capazes de aumentar o desempenho na execução de cargas de trabalho com alto paralelismo no nível de instruções.

O suporte a múltiplas *threads* [17][18] é uma técnica que foca a exploração de paralelismo de maior granularidade, explorando o paralelismo não apenas no nível de instruções, mas também no nível de fluxo de instruções (*threads*). Isto significa um aumento na vazão de *threads* (podendo mais de uma *thread* ser executada ao mesmo tempo) diferente da superescalaridade onde a vazão é de instruções de uma única *thread*. Diversas são as técnicas para exploração do paralelismo no nível de *threads*, sendo que a mais conhecida é a SMT (*Simultaneous Multithreading*) que é suportada por uma arquitetura superescalar.

Esta complexa abordagem de extração de paralelismo vem dando lugar a uma abordagem diferente. A fim de aumentar ainda mais o desempenho, e ainda algumas vezes diminuir a potência dissipada, o uso de processadores com múltiplos núcleos (*Chip Multiprocessors* - CMPs) [2][6][7][12][13] vem sendo consolidada como um bom método para aumento do desempenho de computação.

Com vários núcleos, um processador passa a ter suporte nativo a várias *threads*, sendo os núcleos superescalares ou não. Ou seja, se o CMP possui 8 núcleos escalares, é possível ter 8 *threads* simultâneas. Nada impede que sejam utilizadas superescalaridade e SMT em cada núcleo, mas nestes casos há um aumento considerável na área do CMP. Em função desta nova abordagem de projeto, os processadores com múltiplos

núcleos e que suportam múltiplas *threads* também são conhecidos como CMT ou *Chip Multithreading* [3][8].

Com o surgimento dos CMPs ou CMTs algumas pesquisas e processadores comerciais adotaram o compartilhamento da *cache* de nível 2 como uma alternativa para aumentar o desempenho de aplicações paralelas utilizando o modelo de programação por memória compartilhada.

No entanto em meio aos diversos tipos de aplicações, [15] destaca-se que a localidade dos dados em um programa exerce grande impacto sobre o desempenho. Dessa forma uma aplicação que utiliza um agrupamento de dados contíguos apresenta um comportamento diferente em relação a uma aplicação que utiliza dados não contíguos (alocados em blocos separados na memória).

Alguns problemas estão associados ao uso da *cache* compartilhada. Neste trabalho serão abordados dois destes problemas: Qual o desempenho para cargas com conjunto de dados contíguos e não contíguos? Qual o desempenho em função do aumento do número de processadores compartilhando a mesma *cache*?

Portanto, o objetivo deste trabalho é estudar o impacto de cargas com conjunto de dados contíguos e não contíguos para diferentes agrupamentos de núcleos de processamento compartilhando a *cache* L2 em um CMP homogêneo de até 32 núcleos.

A principal contribuição deste artigo está no estudo do desempenho do acesso à *cache* L2 sob as condições apresentadas no objetivo do artigo, além da ampliação dos resultados de outros autores [1][11].

Para a geração dos resultados foi utilizada a ferramenta de simulação de sistemas Simics [19], em conjunto com o pacote de aplicações para *benchmark* SPLASH-2 [15].

As seções seguintes apresentam os trabalhos correlatos, a proposta do trabalho, os resultados e as conclusões.

2. Trabalhos Correlatos

Ao longo de anos, diversos trabalhos abordam alternativas e estudam formas de diminuir os impactos das altas latências de memória em *chips* multiprocessados. Nesta seção, citamos alguns trabalhos desta linha de pesquisa.

O trabalho apresentado em [1] faz um estudo de diferentes organizações de memória em um computador multiprocessado. O trabalho apresenta a avaliação da influência das diferentes organizações de *clusters* de processadores sobre o desempenho do sistema. Os resultados obtidos mostraram que, para um sistema multiprocessado de até oito processadores, as contenções geradas pelo barramento são responsáveis por grande parte do tempo total de execução. Sendo assim, o agrupamento dos processadores pode eliminar parte dessa contenção. Além disto, este agrupamento de processadores tende a reduzir os tempos de espera por leitura dos dados, quando utilizado espaço de trabalho compartilhado.

Em [11], um dos focos do trabalho também foi testar diferentes organizações de memória. Porém, a simulação foi baseada em um sistema contendo um processador com múltiplos núcleos, onde foram simuladas diferentes organizações de agrupamento de núcleos por memória *cache* L2. Esta simulação foi a partir de um CMP de 32 núcleos, com agrupamentos de 1, 2 e 4 núcleos. Segundo o autor a principal conclusão foi que o compartilhamento da L2 por mais de 2 processadores contribuiu para o ganho de desempenho, sendo importante ressaltar a diferença com o trabalho de [1]. Neste caso, as interconexões entre os processadores com a memória *cache* L2 são internas ao chip, levando então a menores latências.

Em [10] foi apresentado o impacto do uso de agrupamentos de até 4 núcleos em um processador com múltiplos núcleos de processamento, com variações no tamanho da memória *cache* L2 compartilhada (1MB, 2MB e 4MB). Com relação aos resultados, o aumento do número de processadores por grupo compartilhando memórias *cache* maiores apresentou ganho de desempenho na maioria das aplicações de *benchmark* testadas.

A proposta apresentada por [20], se refere à extensão sobre o método de acesso não uniforme à memória, propondo uma política de reposicionamento e posicionamento dos dados baseado em associatividade por distância, para *chips* multiprocessados considerando o uso de memória *cache* L2 compartilhada ou privada. Esta proposta chamada de NuRAPID (*Non-uniform acess with Replacement And Placement usIng Distance associativity*), mostrou resultados significativos em uma máquina CMP de 4 núcleos, e 8MB de memória *cache*. Obtendo melhorias no desempenho de até 13% em sistemas com memória *cache* compartilhada, e até 8% em sistemas com *caches* privadas.

Em [14] são estudados diversos aspectos relacionados com as interconexões em *chip* multiprocessados. Neste trabalho, mostra-se a importância em sempre projetar as interconexões levando em consideração todos os outros aspectos do processador, como tamanho de memória *cache* e agrupamento de núcleos utilizado. Além disto, o artigo conclui que ao levar em conta os custos de interconexão, nem sempre o compartilhamento de *cache* L2 é vantajoso, sob o ponto de vista de ganho de

desempenho e também sob o aspecto do *overhead* sobre a área do projeto, gerada pela interconexão da memória *cache* com os núcleos.

A diferença deste artigo em relação aos demais trabalhos correlatos está no agrupamento de um número maior de núcleos de processamento do chip para uma mesma memória *cache* L2 compartilhada. Além disto, é realizada a verificação e avaliação do desempenho para uma mesma carga de trabalho com dados contíguos e não contíguos. A carga de trabalho escolhida é a mesma utilizada nos demais artigos correlatos e está descrita na Seção 3.2.2.

3. Método e Proposta de Simulação

Ao longo desta seção serão apresentados o método utilizado e a proposta de simulação realizada para obtenção dos resultados.

3.1. Método

Para avaliar o desempenho de um determinado sistema computacional, onde está presente um CMP homogêneo, podem ser utilizados três modelos: i) analítico, ii) simulação, e iii) medição.

Conforme será descrito na Seção 3.2, o sistema proposto é muito complexo. Além do projeto do processador e da hierarquia de memória, estão envolvidos toda a arquitetura de um computador e do sistema operacional. Sendo assim, a utilização de um modelo analítico se torna inviável em função do tempo e da quantidade elevada de variáveis envolvidas. Como se trata de um CMP com vários núcleos de processamento e com uma variedade de alterações na organização de acesso a *cache* L2, uma medição também se mostra inviável, uma vez que não estão disponíveis CMPs no mercado para a realização das medidas. Sendo assim, a simulação foi escolhida por possuir as seguintes características: possibilidade de desenvolver e realizar os testes com mais rapidez, facilidade de alterações na arquitetura do projeto, e por ser um ambiente de alto controle de todas as variáveis envolvidas.

A definição do modelo foi a primeira etapa do método utilizado para realização dos testes apresentados neste trabalho. As demais etapas são:

- Definição do ambiente de simulação;
- Projeto das arquiteturas simuladas;
- Definição da carga de trabalho submetida;
- Obtenção dos resultados;
- Análise e avaliação dos resultados;

Na seção seguinte estão descritos em detalhes o ambiente usado para a simulação, as arquiteturas projetadas e a carga de trabalho utilizada para obtenção dos resultados.

A Seção 4 apresenta os resultados obtidos tal como as análises e avaliações em função do desempenho obtido e das comparações com os trabalhos correlatos.

3.2. Proposta de Simulação

Levando em consideração o objetivo deste trabalho, que é o estudo do impacto de diferentes agrupamentos de núcleos compartilhando uma mesma memória *cache* L2, foi simulado um *chip* multiprocessado com número de núcleos além da capacidade atualmente disponível no mercado.

Através de alguns exemplos de processadores comerciais como o Power5 [2] e o Niagara [12], podemos notar uma certa tendência das industrias de *chips* multiprocessados na adoção de agrupamento de núcleos para compartilhamento da *cache* L2.

Assim, formou-se a proposta de modelar uma arquitetura básica com 32 núcleos, e a partir daí, distribuí-los em diferentes organizações de memória *cache* L2 (Figura 1), a fim de estudar o desempenho e o comportamento das hierarquias em cada caso.

Esta arquitetura consiste em grupos de processadores, cada qual com uma memória *cache* L1 de dados e uma de instruções. Tais memórias estão conectadas à memória *cache* L2. As memórias *cache* L2, por sua vez, estão conectadas a *switches* de alta largura de banda, os quais estão conectados aos outros *switches* e também à memória principal do sistema.

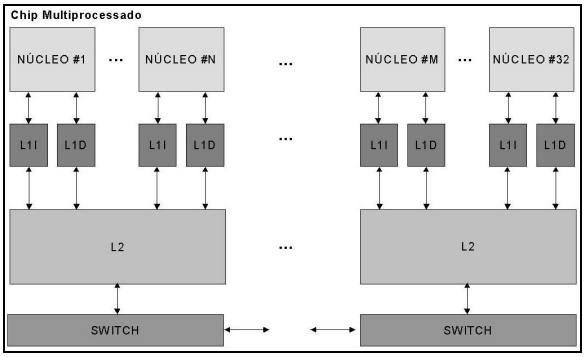

Figura 1. Arquitetura Simulada

Para isolarmos o impacto somente dos diferentes tamanhos de grupos de núcleos, foi estabelecido o número total de 32 núcleos com variação da quantidade de núcleos agrupados. As organizações propostas estão na Tabela 1, onde podemos ver que foram utilizadas configurações desde uma *cache* L2 privada por núcleo até o compartilhamento total da *cache* pelos 32 núcleos do processador simulado.

Tabela 1. Organizações do Sistema

Nome	Conjuntos	Processadores em cada conjunto	Tamanho em cada *cache* compartilhada
L2S1	32	1	1 MB
L2S2	16	2	1 MB
L2S4	8	4	1 MB
L2S8	4	8	1 MB
L2S16	2	16	1 MB
L2S32	1	32	1 MB

Ainda podemos ressaltar que não houve aumento do tamanho de cada memória *cache* L2, sendo que esta ficou em todos casos com tamanho igual a 1MB por *cache*. A mudança no tamanho de memória *cache* poderia influenciar nos resultados alterando os valores de *cache hit* e *cache miss*.

Para facilitar a representação das simulações, foi escolhido o nome L2Si [i = 1, 2, 4, 8, 16, 32] para nomear os sistemas com memória *cache* L2 não compartilhada (i = 1), ou compartilhada entre 2, 4, 8, 16 ou 32 núcleos do processador.

Assim como [1][9][10][11], todos os parâmetros de latência foram considerados em ciclos, a fim de simplificar os resultados. Desta forma, a freqüência do processador simulado não interfere nos resultados apresentados na Seção 4.

Como descrito na Tabela 2 de configurações do sistema, cada núcleo do multiprocessador é um UltraSparc III, com execução em ordem, com um *pipeline* ideal e IPC = 1. Esta simplificação na configuração do processador acelera a simulação e a obtenção dos resultados sem influenciar na avaliação realizada na *cache* L2.

Para as configurações de memória *cache* L1, consideramos uma memória de tamanho igual a 64 kB, sendo 32 kB para a memória de instruções e 32 kB para dados. O tamanho da linha utilizado foi de 64 Bytes, a política de substituição de dados adotada foi LRU (*Least Recently Used*), além de uma latência de 3 ciclos para esta memória.

Nas configurações de memória *cache* L2, consideramos o tamanho de 1MB por memória L2, utilizando linhas de tamanho igual a 128 Bytes também com a política de substituição LRU. As latências de leitura e escrita foram definidas para 10 ciclos, e a latência para acessar cada *cache* vizinha tem um custo adicional de 4 ciclos, em cada salto.

Se um processador não achar uma determinada informação em sua memória *cache* L2, essa informação poderá ser encontrada em alguma das outras *caches*. Assim, a latência da falta de dados em uma *cache* L2 também levará em consideração o número de saltos necessários, entre as *caches* até encontrar os dados procurados.

Tabela 2. Configurações do Sistema

Componente	Parâmetro	Configuração
Processador	Chip Multiprocessado	32 núcleos
	Núcleos	UltraSparcIII
	IPC	1
	Execução	Em ordem
Cache L1	Tamanho da linha	64 B
	Cache de dados	32 kB
	Cache de instruções	32 kB
	Tamanho total	64 kB
	Política	LRU
	Associatividade	4-*way*
	Latência	3 ciclos
Cache L2	Tamanho da linha	128 B
	Política	LRU
	Associatividade	4-*way*
	Latência de Leitura	10 ciclos
	Latência de Escrita	10 ciclos
	Latência Leitura Próxima	4 ciclos
	Latência Escrita Próxima	4 ciclos
Memória	Latência	200 ciclos

Assumimos que a memória principal do sistema tem tamanho infinito e latência de 200 ciclos para acesso.

Após ser definida toda a proposta, foi escolhido o simulador Simics [19] para ser utilizado neste trabalho. Ele é uma poderosa ferramenta onde podemos simular sistemas inteiros, sendo esta, uma abordagem bem próxima de um sistema real. Uma breve apresentação sobre a modelagem feita no ambiente de simulação está presente na Seção 3.2.1.

Em cada máquina simulada foi feita uma instalação do sistema operacional da Sun Microsystems, "Solaris 9", permitindo assim a execução de *benchmarks*.

As aplicações de teste escolhidas fazem parte das aplicações de *benchmark*, SPLASH-2. No caso, devido a restrições de tempo, foram escolhidas somente duas implementações da aplicação Ocean para serem executadas em nossa simulação. Porém, estas implementações escolhidas possuem algumas características interessantes a serem estudadas, pois uma possui porções de dados alocados em forma contígua, sendo que na segunda implementação os dados são alocados de forma não contígua. Outras informações a respeito do *benchmark* encontram-se na Seção 3.2.2.

3.2.1. O ambiente de simulação SIMICS

O Simics [19] é um simulador que fornece ferramentas de modelagem e configuração para que possam ser obtidos resultados de diversos sistemas.

Dessa maneira, este é um ambiente de simulação de sistema completo, em nível de conjunto de instruções. Assim, o propósito dessa ferramenta é de modelar o sistema alvo no nível de instruções individuais, executando estas instruções uma por vez.

Para simulação de *cache*, o Simics provê ferramentas para configurar dispositivos bastante flexíveis. Em uma simulação de *cache* o Simics permite tanto observar informações a respeito de tempo, como a respeito das informações contidas na memória. O simulador possui dois modelos pré-modelados de cache: *g-cached* e *g-cached-ooo*.

O modelo *g-cached* fornece todas as condições para modelagem de uma *cache* ligada a um processador executando as instruções em ordem e fornecendo relatórios sobre as atividades realizadas.

Já o modelo *g-cached-ooo* além das funcionalidades apresentadas pela *g-cache*, provê ainda, a possibilidade de ser utilizada em simulações de processamento fora de ordem.

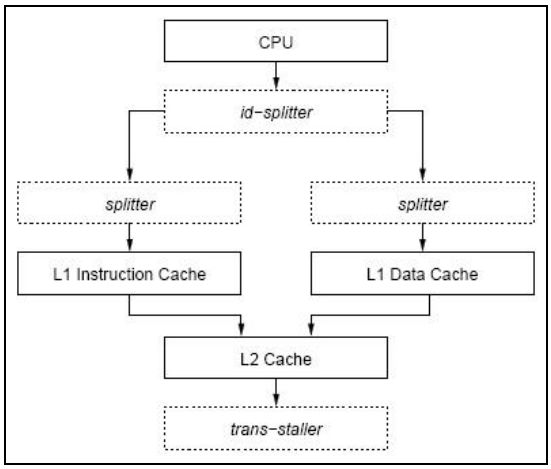

Figura 2. Modelo *g-cache* [18]

Mesmo o modelo *g-cache-ooo* sendo o mais completo para uma simulação de *cache*, este modo é o mais lento do Simics, e não foi utilizado, devido a restrições de tempo, pois para a metodologia utilizada, este poderia tornar os experimentos inviáveis.

O funcionamento genérico do modelo *g-cache* é ilustrado na Figura 2, onde podemos ver os seguintes componentes:

- *Id-splitter*: Utilizado para fazer a separação entre instruções e dados para a memória *cache* correta.
- *Splitter*: Módulo responsável em particionar os dados, afim de só alocar a quantidade de dados coerente com o tamanho de memória.
- *Trans-staller*: Dispositivo simples, que simula a latência da memória.

3.2.2. *Benchmarks* utilizados

A carga de trabalho utilizada foi a Ocean, a qual faz parte do conjunto de *benchmark* SPLASH-2. Este conjunto é composto por diversas aplicações paralelas [15], cada uma contendo diferentes características computacionais.

Este *benchmark* tem sido cada vez mais utilizado em teste de desempenho de arquiteturas paralelas, pois suas aplicações representam diversos contextos de utilização do processamento.

A aplicação Ocean que foi utilizada possui duas implementações diferentes, uma com o conjunto de dados contíguo (*contiguous partitions*) e a outra com as estruturas de dados não contíguas (*non contiguous partitions*).

A aplicação Ocean estuda movimentos de grande escala de um oceano. As grades utilizadas pela aplicação são particionadas de forma quadrada. A aplicação é implementada em linguagem C e utiliza *threads* para a paralelização da aplicação.

Na primeira implementação (contígua), todos os dados são alocados em uma matriz tri-dimensional. Dessa forma o primeiro índice da matriz, refere-se ao processador pertencente. Assim, todos os dados ficaram alocados em porções contíguas de memória.

Na segunda implementação, cada conjunto de dados pertencente a um dado processador é alocado em uma matriz bi-dimensional, evitando dessa forma que os dados de diversos processadores sejam alocados de forma contígua na memória.

As implementações da aplicação Ocean foram compiladas com o compilador gcc versão 3.4.6 e executadas com os parâmetros iniciais do problema conforme descritos na Tabela 3.

Tabela 3. Parâmetros da Aplicação Ocean

Parâmetro	Valor
Processadores	32
Tamanho da grade	258x258
Resolução da grade (metros)	20000.00
Tempo entre as relaxações (segundos)	28800
Erro máximo tolerado	1e-07

4. Resultados

Nesta seção são apresentados os resultados obtidos através das simulações realizadas com as aplicações Ocean contígua e não contígua.

A Figura 3 apresenta o gráfico de tempo de execução para as duas implementações da aplicação Ocean. A unidade de medida está dada em ciclos de relógio.

Com relação aos tempos de execução, notamos que houve uma redução de até 0,93% entre a organização L2S1 e L2S32 executando a implementação de dados contíguos. A organização L2S1 foi a de pior desempenho, variando gradativamente até a de melhor desempenho, L2S32.

Já na implementação de dados não contíguos houve um comportamento diferente, onde notamos na máquina L2S1 uma diferença de aproximadamente 0,11% em relação à máquina de pior desempenho. Partindo da implementação L1S1 e chegando na L2S4, o sistema obteve uma melhora no desempenho, chegando a uma redução de tempo de até 0,80% com relação ao pior desempenho. E, partindo da configuração L2S4, chegamos gradativamente ao pior de todos os desempenhos (0,90%), que foi da máquina L2S32.

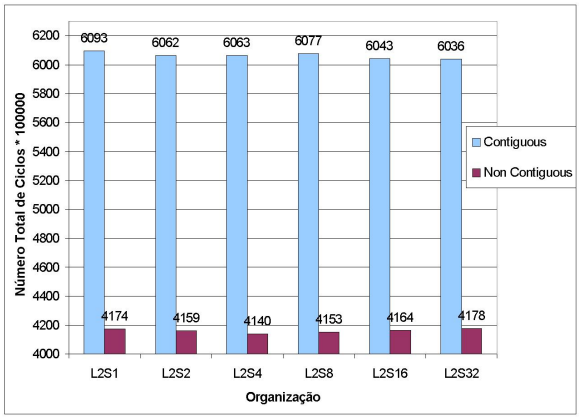

Figura 3. Quantidade de ciclos de execução

Na Figura 4 está ilustrado o gráfico com os resultados das porcentagens de faltas de dados (*misses*) na *cache* de dados L1.

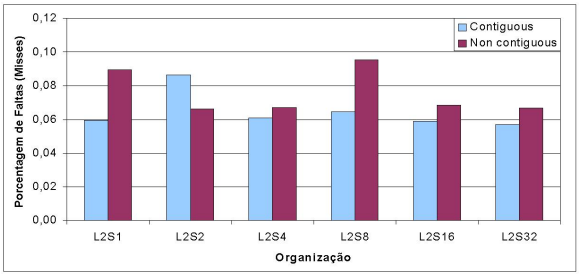

Figura 4. Taxa de faltas durante a leitura da *cache* L1 de dados

Neste gráfico podemos notar uma variação de pouca intensidade entre as diversas organizações, tanto para a implementação contígua quanto na não contígua, sendo que esta pequena variação pode ser considerada como sendo aleatória, resultante do próprio processo de simulação.

Na Figura 5, onde estão os gráficos com valores referentes às porcentagens de faltas na escrita na *cache* L1 de dados, podemos observar um comportamento um pouco diferente do comportamento desta mesma *cache* para leitura.

Na implementação contígua, o número de faltas teve um aumento na organização L2S2, porém retornando ao mesmo patamar da organização L2S1 na máquina L2S32. Portanto, nestes casos testados não se obteve um comportamento bem definido, seja de crescimento ou diminuição.

Observamos uma variação dos resultados de aproximadamente 10,67% na implementação não contígua, onde a menor taxa ficou por conta da organização L1S1, tendo variações até alcançar a organização L2S32, que obteve o maior número de *misses* neste caso.

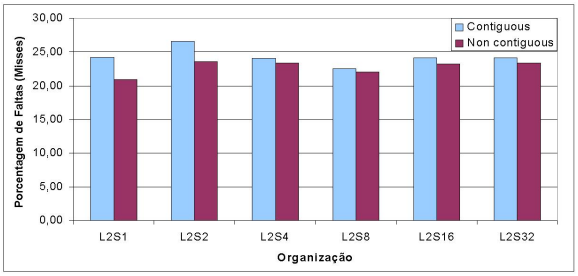

Figura 5. Taxa de faltas durante a escrita na *cache* L1 de dados

O gráfico da Figura 6 (*Read Misses*) representa as taxas de faltas durante o processo de leitura de dados desta *cache*.

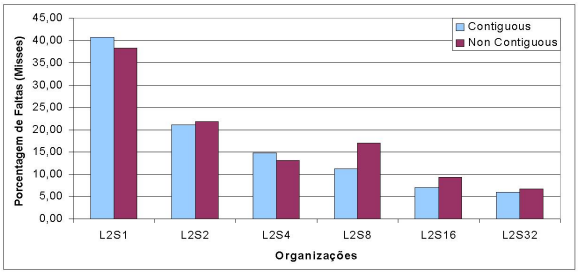

Figura 6. Taxa de faltas durante a leitura da *cache* L2

Houve uma redução entre as organizações de até 85,20% nas quantidades de faltas durante a leitura de dados na implementação contígua. Neste caso a organização L1S1 foi a que apresentou o maior número de faltas e a L2S32 apresentou a menor taxa de faltas.

No caso da implementação não contígua, também houve uma diminuição do número de faltas de até 82%. Porém a organização L2S8 apresentou um comportamento anômalo com relação à tendência das outras organizações. Isto pode ter tido influência de aspectos inerentes ao particionamento ou localidade dos dados do problema.

A Figura 7 contém o gráfico de taxa de faltas durante o processo de escrita de dados da *cache* L2.

Neste gráfico, podemos ver na implementação contígua e na implementação não contígua um crescimento na taxa de faltas de até 59,07% e de 62,07%, respectivamente, conforme aumenta-se o número de processadores compartilhando a memória *cache* L2.

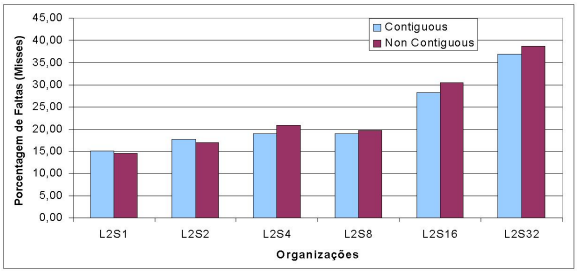

Figura 7. Taxa de faltas durante a escrita na *cache* L2

Após a geração dos resultados podemos fazer certas análises sobre o tempo de execução das duas implementações da mesma aplicação. O número de ciclos total para execução dos dados contíguos teve um comportamento de diminuição conforme aumentou-se o número de processadores compartilhando a mesma memória *cache* L2. Isto era esperado, uma vez que os dados necessários para os diversos processadores nesse caso estavam próximos uns dos outros, fazendo com que a latência de acesso à memória tendesse a diminuir. Já na implementação de dados não contíguos, o número de ciclos para completar o trabalho obteve uma redução conforme aumentou-se o agrupamento de processadores, porém essa aceleração existiu até o agrupamento de 4 processadores, enquanto a partir da organização L2S8 até L2S32 a quantidade de ciclos para completar a tarefa teve um aumento. Isto mostra que para esta aplicação é vantajoso o compartilhamento de *cache* para até 4 processadores. Além desse agrupamento, um número maior de núcleos compartilhando a *cache* começa a degradar no desempenho.

Embora o gráfico de taxa de faltas na leitura da memória *cache* L1 tenha mostrado pequenas variações, os gráficos de taxa de falta das memórias *cache* L1 durante a escrita mostraram um aumento na taxa de faltas para a implementação não contígua. Conforme foi-se aumentando o tamanho dos agrupamentos de núcleos chegando no caso da L2S32 a um aumento de aproximadamente 10,67% acima da configuração L2S1, foi possível verificar que, ao aumentar o agrupamento de núcleos, as penalidades ocasionadas por esta memória também aumentaram.

Os gráficos a respeito da memória *cache* L2 e as taxas de faltas em cada organização testada demonstram claramente que, ao aumentar o número de processadores compartilhando a memória *cache* L2, esta apresenta uma menor quantidade total de faltas na leitura. Porém, na escrita, a taxa de faltas aumenta à medida que aumenta-se os agrupamentos de núcleos de processamento.

Pode-se perceber através dos resultados apresentados que a questão de compartilhamento de memória *cache* L2 é bastante relativa ao tipo de aplicação paralela que será executada. Ficou claro que a implementação com proximidade dos dados obteve nas simulações adotadas um melhor desempenho conforme aumentou-se o agrupamento dos núcleos de processamento. Porém, a implementação de maior distanciamento dos dados usados não obteve um melhor desempenho para todos os casos de maior compartilhamento da memória *cache* L2.

5. Conclusões

A tendência para os novos processadores de propósitos gerais é a utilização de múltiplos núcleos de processamento homogêneos. Como conseqüência, o aumento das *threads* em execução, simultaneamente, faz com que o desempenho de acesso à memória *cache*

L2 seja cada vez mais importante para o desempenho do próprio *Chip Multiprocessor*.

Alguns estudos estão sendo realizados para avaliar o desempenho da *cache* L2 compartilhada em CMPs homogêneos. Neste artigo, verificou-se o desempenho em função dos diferentes comportamentos ao utilizar conjuntos de dados contíguos e não contíguos, demonstrando que em aplicações planejadas, a fim de tirar vantagem da memória *cache* compartilhada (dados contíguos), é possível alcançar boas melhorias no desempenho aumentando o número de núcleos agrupados. Porém, em aplicações paralelas que possuem baixa proximidade nos dados, o agrupamento de um grande número de processadores pode não levar à melhoria do desempenho.

Entre os trabalhos futuros de avaliação do desempenho do compartilhamento da *cache* L2 em CMPs homogêneos estão: execução de uma maior variedade de cargas de trabalhos, cargas de trabalho diferentes e com múltiplos espaços de endereçamento, e o projeto de uma hierarquia de memória que privilegie o desempenho neste contexto.

Agradecimentos

Agradecemos ao Grupo de Processamento Paralelo e Distribuído, ao Laboratório de Sistemas Embarcados, ao Programa de Pós-Graduação em Computação e ao Instituto de Informática da UFRGS. À PUC Minas, à CAPES e ao CNPq que também contribuíram com o apoio no desenvolvimento deste trabalho.

Referências

[1] B. A. Nayfeh, et al., "The Impact of Shared-Cache Clustering in Small-Scale Shared-Memory Multiprocessors", 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA), 1996.

[2] B. Sinharoy, et al., "POWER5 system microarchitecture", IBM J. RES. & DEV, Vol.49 No. 4/5 July/September 2005

[3] Freitas, H. C., Navaux, P. O. A., *Chip Multithreading: Conceitos, Arquiteturas e Tendências*, Trabalho Individual (TI:1253), Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre RS, Brasil, 58p., 2006.

[4] Hennessy, J. L., D. A. Patterson, *Arquitetura de Computadores Uma Abordagem Quantitativa*, Editora Campus, 3a edição, 2003.

[5] J. E. Smith, G. S. Sohi, "The Microarchitecture of Superscalar Processors", IEEE, v. 83, No. 12, pp.1609-1624, 1995.

[6] K. Olukotun, et al., "The Case for a Single-Chip Multiprocessor", International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp.2-11, October 1996.

[7] L. A. Barroso, et al., "Piranha: a scalable architecture based on single-chip multiprocessing" - 27th International Symposium on Computer Architecture (ISCA), pp.282–293, 2000.

[8] L. Spracklen, S.G. Abraham, "Chip Multithreading: Opportunities and Challenges", International Symposium on High-Performance Computer Architecture (HPCA), pp.248-252, February 2005.

[9] M. D. Marino, "Evaluating the Interconnection Latency Costs on the Performance of a CMP with Multisliced L2", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, USA, 2006.

[10] M. D. Marino, "L2-cache hierarchical organizations for multi-core architectures", International Symposium on Parallel and Distributed Processing and Applications (ISPA), Sorrento, Italy, pp.74-83, 2006.

[11] M. D. Marino, "32-core CMP with multi-sliced L2: 2 and 4 cores sharing a L2 slice", 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Ouro Preto, MG, Brasil, pp.141-150, 2006.

[12] P. Kongetira, et al., "Niagara: a 32-way multithreaded Sparc processor", IEEE MICRO, v. 25, Issue 2, p. 21-29, March-April 2005.

[13] R. Kumar, et al., "Heterogeneous chip multiprocessors", IEEE Computer, v. 38, Issue 11, p. 32-38, November 2005.

[14] R. Kumar, V. Zyuban, D.M. Tullsen, "Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling", 32nd International Symposium on Computer Architecture, pp.408-419, June 2005.

[15] S. C. Woo, et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations", International Symposium on Computer Architecture (ISCA), Santa Margherita Ligure, Italy, 1995.

[16] Stallings, W., Arquitetura e Organização de Computadores, Pearson / Prentice Hall, 2005.

[17] T. Ungerer, et al., "A Survey of Processors with Explicit Multithreading", ACM Computing Surveys, Volume 35, Issue 1, pp.29-63, March 2003

[18] T. Ungerer, et al., "Multithreaded Processors", The Computer Journal, British Computer Society, v. 45, n. 3, p. 320-348, 2002.

[19] Virtutech, "Simics 3.0 – User Guide for Unix", Revision 1376, http://www.simics.net, 2007

[20] Z. Chishti, et al., "Optimizing Replication, Communication, and Capacity Allocation in CMPs", International Symposium on Computer Architecture, 2005.