Estudo Preliminar do Desempenho de Caches Multi-banking Inclusivos e Não-inclusivos em CMPs.

Filipe Montefusco Scoton e Mario Donato Marino
Departamento de Engenharia de Computação – Escola Politécnica – Universidade de São Paulo
[filipe, mario]@regulus.pcs.usp.br

Resumo

Atualmente a tendência do mercado tem sido a de se ter múltiplos cores no mesmo chip (Chip Multiprocessors – CMP). Cada core tem seu próprio processador, seu próprio cache L1 e pode ter seu próprio L2 ou mesmo compartilhar este com outros cores. Uma idéia é a de se ter o L2 dividido em bancos e estes bancos serem conectados através de uma rede intra-chip. Uma vez feita essa divisão em bancos, pode-se utilizar um modelo de cache inclusivo ou não-inclusivo. Baseado nessas idéias, um estudo de comparação de performance entre os modelos inclusivo e não-inclusivo foi realizado. Para isso foram modelados dois sistemas, cada um composto por um chip de 32 cores, executando alguns dos benchmarks do pacote SPLASH-2. Os resultados preliminares mostram a melhor performance do modelo não-inclusivo.

1. Introdução

O mercado de micro-processadores tem seguido a idéia de se ter múltiplos *cores* no mesmo *chip* (CMP) (figura 1). Alguns exemplos de fabricantes são a AMD [1] e a Intel [2], que possuem soluções *dual-core* que já estão no mercado. Outros exemplos são a Sun Microsystems [3], que já possui uma solução com oito *cores* para servidores *web* e a IBM [4], com processadores *dual-core* para servidores, um processador *tri-core* já disponível, e um com nove *cores* a ser lançado, sendo os dois últimos no mercado de jogos eletrônicos. Graças à necessidade de diminuição de consumo, de dissipação, aumento de escalabilidade e de performance, soluções CMP são preferíveis quando comparadas a processadores *single-core* mais poderosos.

A presença de múltiplos *cores* exige uma adaptação na hierarquia de memória utilizada nos modelos *single-core*, de modo a garantir a coerência das informações contidas nos *caches*. Para isso são utilizados protocolos de coerência de *caches.* Os *caches,* por sua vez, podem ser inclusivos ou não-inclusivos.

A configuração de *caches* L2 divididos em bancos (figura 1) tem sido a mais utilizada [7-12], aproveitando características de paralelismo no acesso às informações do *cache*. O principal objetivo do trabalho é o de se comparar o desempenho de modelos de *cache* inclusivo e não-inclusivo para duas configurações de *cache* L2, dividido em diferentes números de bancos. Através da simulação de um sistema completo, executando alguns dos *benchmarks* do pacote SPLASH-2 [5], foi modelado um CMP com *caches* inclusivos e não-inclusivos e variando o número de bancos que iriam compor o *cache* L2.

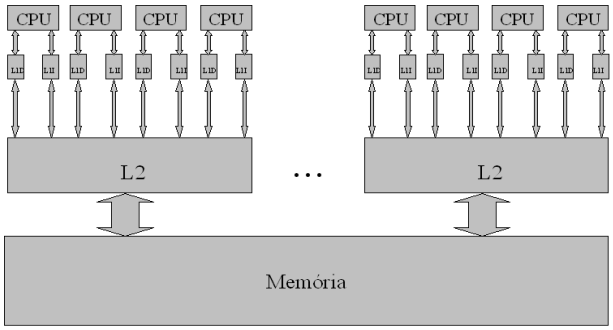

Figura 1: Chip contendo múltiplas células, cada uma possuindo 4 *cores* interligados a um *cache* L2 conectado à memória.

2. Cache Multi-banking, inclusividade e não-inclusividade

2.1 Cache Multi-banking

Um *cache* dito *multi-bank* (ou *interleaved*) (figura 2) faz uso de uma interconexão na forma de *crossbar* para dividir a cadeia de referência da memória entre

múltiplos bancos de *cache*. Essa separação em bancos permite a um banco servir de maneira independente a uma determinada requisição de *cache* a cada ciclo.

Segundo [7], uma função de seleção de bancos é necessária para mapear os endereços de referência da memória para o banco correspondente. Esta função pode afetar a largura de banda fornecida pela implementação *multi-bank* já que influencia a distribuição de acesso aos bancos. Ainda segundo [7], uma função ineficiente pode aumentar o número de conflitos nos bancos, reduzindo a largura de banda fornecida, já funções eficientes precisam contrabalancear a complexidade de implementação e a possibilidade de encurtar o tempo de acesso ao *cache*.

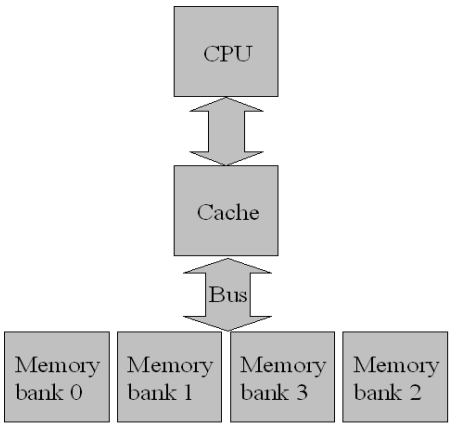

Figura 2: Organização de memória *interleaved*, retirada de [6].

2.2 Inclusividade e não-inclusividade

A inclusividade dos *caches* significa que os dados contidos no *cache* L1 também podem estar contidos no *cache* L2. Dessa maneira, quando há uma transferência de dados entre os dois níveis de *cache*, os dados são copiados de um para o outro (figura 3 - a).

Algumas implementações de *caches* inclusivos garantem que todos os dados contidos no *cache* L1 também estão contidos no *cache* L2. Uma vantagem dos *caches* estritamente inclusivos é a de que quando dispositivos externos ou outros processadores em um sistema multiprocessado querem remover uma linha de *cache* do processador, o processador precisa checar apenas o *cache* L2. Em hierarquias de *cache* que não obrigam a inclusão, o *cache* L1 precisa ser checado também. Como uma desvantagem, há uma correlação entre as associatividades dos *caches* L1 e L2: se o *cache* L2 não tem no mínimo tantas vias quanto todos os *caches* L1 juntos, a associatividade dos *caches* L1 torna-se restrita, uma vez que o conteúdo de todos os *caches* L1 devem estar contidos dentro do *cache* L2.

A não-inclusividade dos *caches* significa que dados contidos no L1 não estão contidos no L2. Dessa maneira, quando há uma transferência de dados entre os dois níveis de *cache*, os dados são movidos de um nível para o outro (figura 3 - b).

A vantagem de *caches* não-inclusivos é que eles podem armazenar mais dados. Esta vantagem se torna maior à medida em que cresce o tamanho dos *caches*. Quando ocorre um *miss* no *cache* L1 e um *hit* no *cache* L2, a linha de *cache* L2 encontrada é permutada com uma linha do L1. Esta permutação é um pouco mais trabalhosa do que apenas copiar a linha do L2 para o cache L1, que é o que o *cache* inclusivo faz, porém possibilita a expansão do *cache*, passando-se a ter uma capacidade de armazenamento igual à capacidade do *cache* L1 somada à do L2.

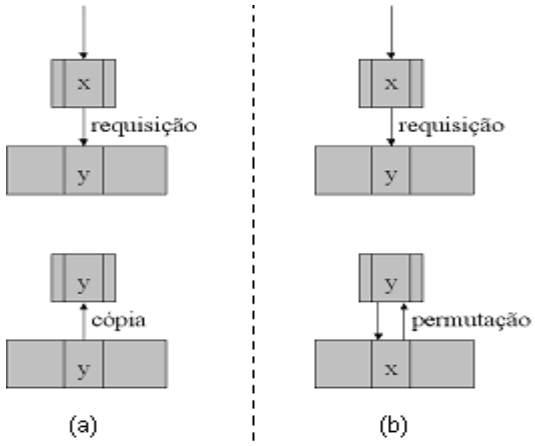

Figura 3: (a) Substituição no modelo de *caches* inclusivo. **(b)** Substituição no modelo de *caches* não-inclusivo.

3. Metodologia, Modelagem e Simulação

Seguindo a linha de outros trabalhos da mesma área [7-12], foi adotada uma metodologia fundamentada na simulação e avaliação de sistemas completos.

Para as simulações utilizou-se o Simics 2.0.16 [13] em conjunto com o módulo Ruby do Gems 1.2 [14], devido à possibilidade de simulação de um sistema completo, proporcionada pelo Simics, e pela maior facilidade de modelagem de hierarquias de memória, proporcionada pelo módulo Ruby. Foram simulados modelos com *caches* inclusivos e não-inclusivos para *chips* de 32 *cores*, cada *core* com seu *cache* L1 de dados e de instruções e um L2 dividido em bancos, compondo três novas configurações, com 16, 8 e 4 bancos cada. É assumida uma memória DRAM de 4GB e com 80 ciclos de latência. Cada CPU é representada por um UltraSparcIII de execução ordenada, com

pipelining ideal e CPI=1. O sistema simulado executa o sistema operacional Solaris 9, executando os aplicativos do SPLASH-2 compilados utilizando-se o gcc 3.3.0. Para o L1, assumiu-se um *cache* de 128KB (64KB de dados e 64KB de instruções).

No que diz respeito ao L2, assumiu-se um *cache* de tamanho total igual a 16MB, ou seja, 1MB para cada banco no modelo de 16 bancos, 2MB para o modelo de 8 bancos e 4MB para o modelo de 4 bancos.

Tabela 1: Resumo dos parâmetros dos modelos.

Componente	Parâmetro
Processador	UltraSparcIII, execução ordenada, CPI=1
Número de processadores	32
L1 – tamanho L1 – associatividade L1 – latência	128KB (64KB d+64KB i) 4-*way* 3 ciclos
L2 – tamanho total L2 – associatividade L2 – latência	16MB 4-*way* 6 ciclos
Latência da memória Latência *on-chip* Configuração da rede	80 ciclos 1 ciclo *Switch* Hierárquico

Foram utlizadas duas aplicações (Water e Ocean) do pacote de *benchmarks* SPLASH-2, que tem sido usado ao longo dos anos pela comunidade científica a fim de avaliar arquiteturas de memória compartilhada [7, 8, 11].

O Water tem como característica uma alta taxa de acertos no *cache* L1 (cerca de 90%), enquanto o Ocean tem uma taxa mais baixa (cerca de 65%). Os dois foram escolhidos devido ao fato de representarem categorias diferentes de aplicações, ou seja, com alta e baixa taxa de acertos no *cache* L1.

Todos os *benchmarks* foram compilados e executados para valores de entrada de classe A. O procedimento utilizado para avaliar os *benchmarks* foi sempre o de executá-los imediatamente após a inicialização do sistema operacional (Solaris 9) [8]. O tempo foi medido em número de ciclos.

4. Análise dos Resultados

A partir das medições realizadas, tem-se na tabela 2 alguns resultados comparando-se os tempos de execução medidos em ciclos para os diferentes *benchmarks* nas configurações alvo.

Observando-se a tabela 2, para o *benchmark* Water, pode-se notar uma diferença de cerca de 10% no tempo de execução favorecendo o modelo não-inclusivo.

Para o *benchmark* Ocean, a diferença nos tempos de execução é da ordem de 20% a favor do modelo não-inclusivo.

Analisando-se a latência média, medida em ciclos, para ambos os *benchmarks* (tabela 3), pode-se notar valores inferiores para o modelo não-inclusivo (cerca de 9%), o que explica sua maior performance. As réplicas de dados mascaram os acessos *off-chip* maiores.

Analisando-se os tempos de execução e as latências médias, percebe-se que há pouca variação desses valores comparando-se, em um mesmo modelo, diferentes configurações de número de bancos. Uma diferença menor que 5% não permite que se conclua que, para os valores utilizados, há alguma vantagem de uma dentre as três configurações utilizadas na divisão do *cache* L2 em bancos.

Os tempos de execução foram retirados a partir das saídas dos *benchmarks*, medidos através de rotinas inseridas no código fonte de cada um dos *bechmarks*. Para as medidas de latência média foram utilizadas estatísticas geradas pelo Ruby, que realiza medições de latência para cada um dos *misses* calculando no final o valor médio.

Tabela 2: Tempo de execução medido em ciclos.

No. de banks	16 Banks		8 Banks		4 Banks	
Inclusividade	Inclusivo	Não-inclusivo	Inclusivo	Não-inclusivo	Inclusivo	Não-inclusivo
Water	4347160	3883241	4178147	3871747	4145146	3743360
Ocean	110681587	91810800	110007560	90638080	110882613	91811853

Tabela 3: Latência média medida em ciclos.

No. de banks	16 Banks		8 Banks		4 Banks	
Inclusividade	Inclusivo	Não-inclusivo	Inclusivo	Não-inclusivo	Inclusivo	Não-inclusivo
Water	926.198	858.206	928.223	856.727	932.019	855.485
Ocean	930.295	848.899	931.525	846.952	930.110	851.553

5. Conclusões

O principal objetivo do trabalho é o de avaliar qual a influência da inclusividade ou não-inclusividade dos *caches* no desempenho de *chips* com múltiplos *cores*.

Nos testes realizados, foi observado que modelos de *cache* não-inclusivo foram mais eficientes, apresentado tempos de execução menores, explicados pelos valores também menores da latência média.

Inicialmente o trabalho contribuiu ao demonstrar que, embora de implementação mais difícil, o modelo de *cache* não-inclusivo é mais vantajoso em termos de desempenho, ao possibilitar redução nos valores de latência média de aproximadamente 10% e com tempos de execução entre 10% e 20% menores, para as aplicações utilizadas.

Para aplicações com alta taxa de acertos no *cache* L1, é possível que haja melhor desempenho do modelo inclusivo sobre o não-inclusivo no que diz respeito ao tempo de execução, visto que os resultados obtidos com o Water, que tem uma taxa de acertos no L1 maior que o Ocean, são da ordem de 10% menores contra 20% no Ocean. Por outro lado, para aplicações com taxas de acerto ainda menores que as do Ocean, essa diferença nos tempos pode aumentar, demonstrando superioridade de desempenho do modelo não-inclusivo.

Como trabalhos futuros pretende-se fazer medições com outros *benchmarks*, a fim de garantir uma margem de resultados mais ampla bem como uma análise mais detalhada. Além disso pretende-se aumentar a diversidade de configurações, alterando-se o número de bancos de *cache* L2, o tamanho dos *caches* e o número de processadores.

Referências

[1] Advanced Micro Devices, AMD, http://www.amd.com, Acessado em: Maio/2006.

[2] Intel, http://www.intel.com, Acessado em: Maio/2006.

[3] Sun Microsystems, http://www.sun.com, Acessado em: Maio/2006.

[4] International Business Machines, IBM, http://www.ibm.com, Acessado em: Maio/2006.

[5] Woo S. C., Ohara M., Torrie E., Singh J. P., Gupta A.. "The SPLASH-2 programs: Characterization and methodological considerations". ISCA, S. Margherita Ligure - Itália, Julho/1995, pp. 24-36.

[6] Patterson D. A., Hennessy J. L., "Computer Organization and Design". Terceira Edição, Editora Morgan Kaufmann.

[7] Rivers J. A., Tyson G. S., Davidson E. S., Austin T. M., "On High-Bandwidth Data Cache Design for Multi-Issue Processors", IEE Computer Society, North Carolina - USA, Dezembro/1997, pp. 46-56.

[8] Marino M. D., "Preliminary evaluation of interconnection latency on a CMP with multisliced-L2", XXI SIM, Porto Alegre - Brasil, 2006.

[9] Zhang M., Asanovic K., "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors", ISCA, USA, 2005, pp. 336 - 345.

[10] Sohi G. S., Franklin M., "High-Bandwidth Data Memory Systems for Superscalar Processors", ASPLOS-IV, Santa Clara – USA, Abril/1991, pp. 53 - 62 .

[11] Strauss K., Shen X., Torrellas J., "Flexible Snooping: Adaptive Forwarding and Filtering of Snoops in Embedded-Ring Multiprocessors", IEE Computer Society, 2006, pp. 327 - 338.

[12] Chishti Z., Powell M. D., Vijaykumar T. N., "Optimizing Replication, Communication, and Capacity Allocation in CMPs", ISCA, 2005, pp. 357 - 368.

[13] Virtutech Simics, https://www.simics.net.

[14] Winsconsin Multifacet GEMS Simulator, http://www.cs.wisc.edu/gems , "ISCA Tutorial", ISCA, USA, 2005.