Arquitetura de Cache com Associatividade Reconfigurável

Milene B. Carvalho[1], Carlos A. P. S. Martins[2]
Laboratório de Sistemas Digitais e Computacionais (LSDC)
Programa de Pós-Graduação em Engenharia Elétrica, Pontifícia Universidade Católica de Minas Gerais, Av. Dom José Gaspar 500, Belo Horizonte, MG.
[1]milenebc@yahoo.com.br, [2]capsm@pucminas.br

Resumo

Neste artigo apresentamos uma arquitetura de cache com associatividade reconfigurável. Nossos objetivos principais são: propor e analisar uma arquitetura de memória cache com associatividade reconfigurável/variável. Apresentamos a taxa de erro da execução de algumas cargas de trabalho reais representadas por traces obtidos do BYU Trace Distribution Center. Analisamos o desempenho da arquitetura proposta através de comparação das taxas de erro obtidas através da simulação da arquitetura e de caches associativas por conjunto. Além disso, analisamos o espaço necessário para armazenar as tags na cache. Nossa principal contribuição é a proposta de uma arquitetura de memória cache com associatividade reconfigurável/variável capaz de se adaptar às diferentes cargas de trabalho.

1. Introdução

Idealmente, a memória principal de um computador deveria ser grande o suficiente para armazenar qualquer quantidade de bytes desejada e, ao mesmo tempo, veloz o suficiente, para que seu acesso não exigisse um tempo diferente de zero [3]. Como isso não é possível, os pesquisadores que participaram do projeto do ENIAC (*Electronic Numerical Integrator and Calculator*), considerado pela maioria dos pesquisadores o primeiro computador de propósito geral, sugeriram a criação de uma hierarquia de memória.

Dentro de uma hierarquia de memória, as camadas que estão entre a memória principal e a UCP (Unidade Central de Processamento), são chamadas de memória cache. Sua utilização tem o objetivo de melhorar o desempenho da memória do sistema computacional, baseada no principio de localidade de referência temporal e espacial. A utilização da cache visa diminuir o tempo de acesso médio à memória, criando para o processador a "ilusão" de uma memória principal rápida.

O projeto de uma memória cache é um problema de otimização, em que podem ser considerados diversos objetivos, restrições e parâmetros que podem ser variados. A otimização de uma memória cache está relacionada, principalmente, com a maximização da taxa de acerto (*hit*) e a minimização do tempo de acesso [14]. Considerando o problema de otimização no projeto de memória cache, foram desenvolvidas três organizações de cache: mapeamento direto, completamente associativa e associativa por conjunto [7].

Na organização do tipo mapeamento direito, cada bloco pode ser encontrado em exatamente uma posição da cache, um *slot* determinado, dessa maneira, basta um acesso à memória para verificar se a palavra procurada se encontra na cache. Sua principal vantagem está na facilidade de se detectar um cache *miss/hit*, no entanto, diversos blocos competem por uma mesma posição na memória cache, gerando conflitos. Este tipo de organização usa um único comparador de *tags*.

Em caches completamente associativas, as palavras (agrupadas em blocos) podem ser encontradas em qualquer posição da cache (entrada). Conflitos por regiões específicas, como ocorrem em caches do tipo mapeamento direito não ocorrem, mas, no pior caso (um único comparador), é necessário verificar todos os *slots* da cache para detectar um cache *miss/hit*. Nesta organização são usados mais de um comparador em paralelo, no entanto, é necessário verificar o custo da utilização de diversos comparadores.

A cache associativa por conjunto é uma organização intermediária entre as duas organizações descritas anteriormente. Os blocos podem ser encontrados somente em um *slot*, no entanto, cada *slot* tem *n* entradas, onde *n* é o valor da associatividade da

cache. Esta cache é chamada associativa por conjunto *n-way*. Nesse tipo de organização, é necessário checar *n* entradas de um *slot* específico para detectar um cache *miss/hit*. Geralmente neste tipo de organização são usados *n* comparadores em paralelo.

Na realidade, as duas primeiras organizações podem ser vistas como casos especiais da última. Uma cache mapeamento direto pode ser considerada uma cache associativa por conjunto *1-way*. Uma cache completamente associativa pode ser vista como uma cache associativa por conjunto com um único *slot n-way*, onde *n* é o número de blocos que podem ser armazenados na cache [7]. No entanto, geralmente não são utilizados *n* comparadores por causa do custo deles.

O desempenho da memória cache está diretamente relacionado com sua organização e com a **carga de trabalho** (*workload*). As características arquiteturais da cache determinam quais são as cargas de trabalho que terão melhor desempenho quando executadas [10], ou em quais arquiteturas uma carga de trabalho terá um melhor desempenho.

A arquitetura das memórias cache presentes nos processadores atuais é fixa, isto é, quando a cache é projetada, sua configuração é definida e esta não pode ser modificada durante a execução de uma carga de trabalho. A escolha da arquitetura é baseada em uma configuração que possua um desempenho bom para uma média de cargas de trabalho, isto é, uma configuração que não é ideal para todas as cargas de trabalho do sistema, mas para uma média das mesmas.

A carga de trabalho de um processador de propósito geral é muito variável. Isso significa que em um mesmo processador são executadas tarefas com diferentes comportamentos de acesso a memória. Como a configuração da cache é fixa, algumas cargas de trabalho não são executadas com um desempenho satisfatório. Além disso, algumas que possuem um desempenho satisfatório ainda poderiam ter um desempenho melhor. Podemos concluir que o **problema** motivador de nossa pesquisa é que o desempenho das memórias cache utilizadas nos computadores não é otimizado para **todos** os *workloads*.

A **computação reconfigurável** é um paradigma que visa a flexibilidade do objeto que pode ser configurado [4] [12]. Para isso, o objeto reconfigurável assume uma das diversas configurações possíveis para ele. Cada uma delas é específica para uma determinada aplicação em um dado momento. Como a configuração é específica para cada aplicação, o **desempenho** das aplicações melhora, porque a configuração do objeto pode ser ideal ou próxima do ideal para cada aplicação, através da **adaptação** do objeto à aplicação. Além disso, através da adaptação, é possível otimizar não só o desempenho, como também os recursos do sistema [1]. Desta maneira, a computação reconfigurável permite que o objeto possua um comportamento flexível (variável) e conseqüentemente tenha um desempenho melhor que um mesmo objeto com comportamento fixo.

Uma das maneiras de obter uma otimização do funcionamento da cache é adaptar seu comportamento/estrutura/configuração à carga de trabalho do sistema computacional. Assim, alguns parâmetros da cache podem ser variados, como o número de *slots*, a associatividade, o tamanho do bloco, etc. No entanto, como as caches tradicionais atuais possuem um comportamento fixo, os parâmetros não podem ser variados de acordo com cada carga de trabalho. Por isso, neste trabalho, propomos uma arquitetura de cache que permite que os diferentes *slots* tenham associatividade diferenciada. Dessa maneira, a associatividade de cada *slot* pode ser alterada, fazendo com que o comportamento da cache possa ser adaptado à carga de trabalho do sistema.

Nossos objetivos principais neste artigo são: propor e analisar uma arquitetura de memória cache com associatividade variável, isto é, reconfigurável. Nossa principal meta é a proposta de uma arquitetura de memória cache com associatividade reconfigurável.

Este artigo está organizado da seguinte maneira: na seção 2 apresentamos os trabalhos correlatos à nossa pesquisa, na seção 3 apresentamos a proposta da arquitetura de memória cache com associatividade reconfigurável, na seção 4 apresentamos e analisamos alguns resultados obtidos com nossa arquitetura, na seção 5 apresentamos nossas conclusões.

2. Trabalhos correlatos

Nesta seção apresentamos trabalhos que permitem a modificação do comportamento de uma cache depois de seu projeto, adaptando dinamicamente sua estrutura ou organização de acordo com a carga de trabalho.

Uma cache armazena temporariamente uma porção da memória que se acredita que será utilizada considerado-se os conceitos de localidade temporal e espacial. Dessa maneira, podemos classificar os trabalhos sobre memória cache reconfigurável em dois conjuntos: aqueles que modificam a cache com base na localidade espacial e aqueles que a modificam com base na localidade temporal. No entanto, nada impede que técnicas destes dois conjuntos possam ser usadas simultaneamente em uma cache.

Praticamente todos os trabalhos usam monitores [11] para obter informações e estatísticas da carga de

trabalho durante sua execução. Eles podem ser implementados, em software/aplicativo, no sistema operacional ou em hardware. Nos dois primeiros casos, existe um *overhead* no processamento causado pelo monitor e no terceiro caso, existe o custo do desenvolvimento do monitor em hardware. Um algoritmo que escolhe a nova configuração da cache para uma carga de trabalho usa as informações do monitor. Uma outra alternativa é a utilização de *tags* que identificam as características da carga de trabalho ou a configuração que deve ser utilizada durante sua execução. Nesse caso, a *tag* é atribuída à carga de trabalho antes de sua execução. Esta tarefa pode ser executada pelo próprio compilador da aplicação ou por um outro software dedicado a isso. Quando a carga de trabalho é executada, a *tag* é avaliada e de acordo com ela, o funcionamento da cache é alterado.

Considerando a localidade espacial, existem trabalhos que apresentam mudanças no tamanho do bloco/linha [17] e outros que modificam o tamanho de *fetch* [15]. Nesses trabalhos, o parâmetro (bloco ou tamanho de *fetch*) pode assumir um valor dentro de um limite. O valor ótimo é escolhido dinamicamente durante a execução. Essa abordagem é chamada por seus criadores de "adaptação".

Na abordagem que realiza adaptação do tamanho do bloco [17], a cache possui múltiplos tamanhos de bloco e cada tamanho individual é modificado de acordo com a localidade espacial inerente à aplicação. Com essa abordagem o tráfego de informações entre a cache e a memória diminui.

Enquanto em uma cache tradicional o tamanho de *fetch* é fixo, igual ao tamanho do bloco da cache, na abordagem adaptativa, quando ocorre um cache *miss*, podem ser buscados na memória múltiplos blocos. Esta abordagem pode obter as mesmas vantagens da abordagem que adapta o tamanho do bloco.

Considerando a localidade temporal, existem trabalhos que modificam a associatividade da cache. Muitos artigos apresentam a diminuição da associatividade da cache para minimizar a quantidade de energia consumida no uso da mesma, enquanto mantém o desempenho alto [9] [13]. Outros, no entanto, acreditam que o custo/benefício de uma diminuição no consumo de energia permita até uma diminuição no desempenho. Esses trabalhos não pretendem melhorar o desempenho da cache, eles trabalham com a redução do consumo de energia através da desativação de *ways* (todas as entradas dos *slots* correspondentes a uma coluna da cache) que não contêm o dado desejado. O desafio nesses trabalhos é reduzir a associatividade sem comprometer muito o desempenho. Eles usam monitores e algoritmos que podem prever se uma diminuição da associatividade é possível ou se um aumento é necessário.

A *Reactive-Associative Cache* (r-a cache) [2] proporciona flexibilidade de associatividade. Ela possui dois tipos de posição para armazenamento de dados: mapeamento direto e associativo por conjunto, a segunda possui um tempo maior para descobrir se a palavra está neste tipo de posicão que a primeira. Para proporcionar flexibilidade e alto desempenho, nessa organização, os blocos são armazenados nas posições de mapeamento direto e somente em caso de conflito que os blocos são armazenados nas posições associativa por conjunto. A organização possui dois tempos de *hit* diferentes, um para cada tipo de posição de armazenamento. Ela possui a vantagem das caches de mapeamento direto em relação ao tempo de *hit* se o dado desejado puder ser encontrado nessas posições. Além disso, ela não possui as desvantagens de conflito das caches mapeamento direto, no entanto, quando é necessário acessar dados conflitantes, o tempo de acesso é maior que o normal (mapeamento direto). A *Reactive-Associative Cache* ainda possui as mesmas desvantagens das caches associativas por conjunto quando ocorre conflito.

Até o momento não encontramos nenhum trabalho que descrevesse uma cache em que fosse possível realizar mudanças na associatividade de cada *slot* da cache para melhorar o desempenho e com o mesmo tempo de acesso para todas as posições.

3. Arquitetura da cache reconfigurável

A computação reconfigurável vem sendo aplicada especialmente em hardware, com os dispositivos reconfiguráveis. Estes dispositivos, incluindo os FPGAs (Field Programmable Gate Arrays), que contém um *array* de elementos de computação ou blocos construtivos. A funcionalidade/comportamento dos dispositivos é determinada por bits de configuração [4].

No entanto, os conceitos de computação reconfigurável podem ser aplicados nas diversas camadas arquiteturais [4][12]. Nosso grupo de pesquisa [19] vem trabalhando em uma arquitetura de cache reconfigurável que adapta seu comportamento dinamicamente de acordo com a carga de trabalho que é executada.

Uma alteração na estrutura provoca uma alteração no comportamento do objeto. A computação reconfigurável permite que um objeto ajuste seu comportamento a uma situação específica. Então este objeto se torna flexível, permitindo um desempenho

maior quando comparado a um objeto com comportamento fixo.

Em nossa arquitetura de cache reconfigurável, representada de modo comportamental pela Figura 1, a configuração do comportamento da cache é determinado pela política adaptável de alocação de blocos. Esta política tem como parâmetros de entrada características da carga de trabalho do sistema e métricas de desempenho ou configurações da memória cache. Ela escolhe a configuração da cache que determinará seu comportamento, como se estivesse escolhendo uma entre as diversas configurações possível considerando as limitações da arquitetura. Uma política de alocação de blocos pode ser definida como a política que determina quais blocos de armazenamento da cache pertencem a cada *slot* de acordo com alguns parâmetros e restrições.

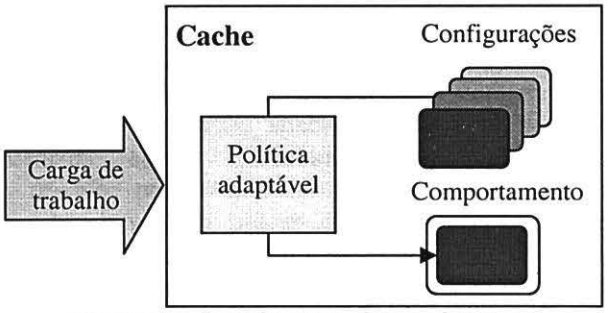

Figura 1. Arquitetura da cache com associatividade reconfigurável

Nossa arquitetura de cache funciona como uma cache associativa por conjunto, mas não há a restrição de ter todos os *slots* com a mesma associatividade. Ela possui uma associatividade inicial e uma associatividade máxima. Esta última indica o número máximo de entradas de um *slot* que podem ser verificadas simultaneamente, isto é, o número de comparadores que a cache possui. Durante a execução da cache, ela pode se adaptar à carga de trabalho, mudando o número de entradas de cada *slot*. Este número pode variar de um até a associatividade maxima (número de comparadores disponíveis). Apesar dessas modificações, nesta arquitetura reconfigurvel, o tamanho da cache é sempre o tamanho determinado no momento de projeto da mesma.

A política de adaptação pode ser implementada em qualquer uma das camadas da arquitetura de um computador: aplicação, sistema operacional, hardware, etc [7]. A camada em que a mesma é implementada, sua complexidade e a freqüência em que a reconfiguração é realizada determinam o *overhead* de reconfiguração.

Como exemplo, podemos implementar o comportamento de nossa cache com associatividade reconfigurável através de uma política adaptável que utiliza uma tabela como a da Figura 2. Esta tabela endereça entradas lógicas de uma memória cache para os endereços reais na memória cache. Estes endereços devem ser acessadas para verificar se uma palavra procurada está na memória cache. A Figura 2 representa uma tabela do segundo *quantum* de tempo de uma cache com associatividade reconfigurável com associatividade inicial igual a 2. Os bits V indicam se a entrada lógica existe. A reconfiguração é realizada ao fim de cada *quantum*, logo, esta figura representa o estado da tabela após a primeira reconfiguração.

Quando o processador requisita uma palavra, o gerenciador de memória consulta esta tabela obtendo os endereços reais das entradas dos *slot* indicado pelo *índice do slot* (Figura 3). Então, estes endereços são utilizados para localizar as entradas reais e cada *tag* deve ser comparada com a *tag* do endereço da palavra requisitada. Cada *tag* é analisada através de uma operação lógica *and* com seu bit V. Se ele é 1 (verdadeiro), a *tag* pode ser usada em um *cache hit*. No entanto, se ele é 0 (falso), o *slot* não possui mais esta entrada e a *tag* não pode ser utilizada. Enquanto a comparação de *tags* é realizada, o dado é acessado.

Como pode ser observado na Figura 2, a posição de memória cache real com endereço 9 do *Slot 1* foi redirecionada para o *Slot 0*. Isto explica porque o endereço 9 aparece nas entradas 1 e 2. O bit V igual a 0 no *Slot 1* indica que esta entrada não está acessível para este *slot*.

	Entrada 0		Entrada 1		Entrada 2	
	V	Endereço	V	Endereço	V	Endereço
Slot 0	1	0	1	8	1	9
Slot 1	1	1	0	9	0	-
Slot 2	1	2	1	10	1	11
Slot 3	1	3	0	11	0	-
Slot 4	1	4	1	12	1	13
Slot 5	1	5	0	13	0	-
Slot 6	1	6	1	14	1	15
Slot 7	1	7	0	15	0	-

Figura 2. Tabela que endereça entradas lógicas da memória cache para endereços reais de memória cache

4. Resultados

Nesta seção apresentamos alguns resultados obtidos através da análise do espaço de armazenamento de *tags* da arquitetura de cache proposta e da simulação desta arquitetura com uma política adaptável simples. Com a

análise do espaço de armazenamento de *tags* avaliamos o espaço de memória física necessário para a implementação da cache. Este tipo de analise pode levar a outros, como o de custo da implementação considerando-se a quantidade de memória necessária. Por outro lado, as simulações com uma política permite uma análise de desempenho da arquitetura de cache reconfigurável com a política apresentada.

4.1. Espaço de armazenamento de *tags*

Quando uma memória cache recebe uma requisição de acesso a uma palavra, ela divide o endereço em porções que são analisadas separadamente. As porções usadas por uma cache associativa por conjunto e, conseqüentemente, pela nossa arquitetura também, estão representadas na Figura 3.

Os bits usados no *Deslocamento no bloco* indicam o deslocamento da palavra procurada no bloco em que ela pode ser encontrada. Os bits de *Índice do slots* são usados para endereçar o *slot* no qual o bloco pode estar armazenado. Os bits da *Tag do bloco* são armazenados na cache para que possam ser usados na comparação realizada quando uma palavra é procurada na cache.

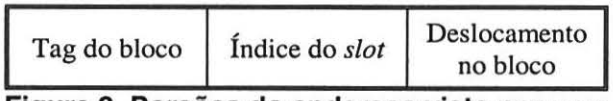

Figure 3. Porções do endereço visto por uma cache associativa por conjunto ou da arquitetura proposta

O número de blocos que podem ser armazenados em uma memória cache é definido pela seguinte equação, onde TamanhoCache representa o espaço da cache dedicado ao armazenamento de blocos:

$$NumeroBlocos = \frac{TamanhoCache}{TamanhoBloco}$$

O número de *slots* de uma cache associativa por conjunto e, conseqüentemente, de nossa arquitetura também, é dada pela equação:

$$NumeroSlots = \frac{NumeroBlocos}{AssociatividadeInicial}$$

O número de bits necessários para endereçar um *slot* é definido por:

$$IndiceSlots = \left| \log_2^{NumeroSlots} \right|$$

Para nossa análise consideramos caches com o mesmo tamanho (espaço de armazenamento de blocos) e blocos do mesmo tamanho. Como os blocos são do mesmo tamanho, o número de bits necessários para representar o deslocamento no bloco é o mesmo. Além disso, considerando-se estes dois parâmetros fixos, o único parâmetro que determina a variação no número de bits necessários para endereçar um *slot* (índice do *slot*) é a associatividade inicial da cache. Através das equações apresentadas anteriormente, podemos concluir que o número de bits do Índice do *slot* é inversamente proporcional à associatividade inicial.

Espera-se que o desempenho da cache com uma organização como a apresentada neste artigo seja similar ou superior a uma organização associativa por conjunto com o mesmo número de comparadores. O desempenho da cache será analisado na seção 4.2 onde mostramos que esta expectativa foi concretizada.

No caso de uma cache associativa por conjunto, a associatividade inicial é igual ao número de comparadores da cache, no entanto, em nossa arquitetura, a associatividade inicial é menor que o número de comparadores. Dessa maneira, considerando-se caches com o mesmo número de comparadores, o número de bits de *Tag do bloco* é maior na cache associativa por conjunto. Isto acontece porque a cache associativa por conjunto possui um número menor de *slots* (possui mais blocos em um mesmo *slot*), com isso, o número de blocos que podem ser armazenados em um mesmo *slot* é maior, então são necessários mais bits para diferenciá-los.

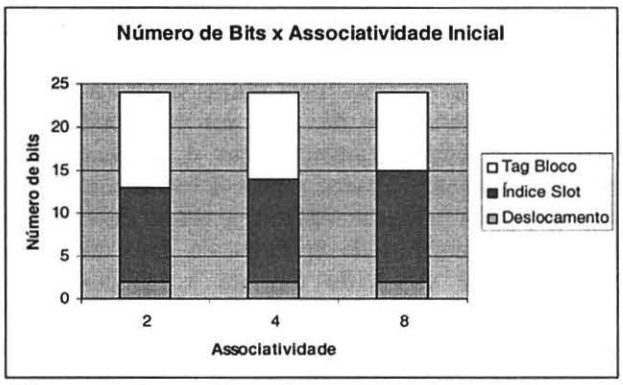

Figura 4. Tamanho das porções do endereço de uma palavra em relação à associatividade inicial da cache

A diferença de tamanho nas porções (número de bits) do endereço de uma palavra está representada na Figura 4. Nesta figura, está representado um endereço de uma memória principal de 512MB e cache de 64KB. Cada bloco possui 4 palavras e cada palavra possui 32 bits. O número de bits de endereçamento é o

mesmo porque o número de palavras que pode ser armazenado é igual, independentemente da associatividade. A diferença não parece ser significativa quando comparamos os bits de uma única palavra, no entanto, quando consideramos todas as *tags* que são armazenadas na cache, esta diferença de poucos bits representa uma grande diferença no espaço de armazenamento de *tags*.

4.2. Análise de desempenho

Para analisar o desempenho de nossa arquitetura de cache com associatividade reconfigurável, criamos uma política adaptativa simples e simulamos nossa arquitetura e caches associativa por conjunto. Para simular as caches, desenvolvemos um simulador que é um módulo do ClusterSim [6] que utiliza simulação dirigida por fluxo [16].

A política adaptativa implementada é baseada em estatísticas obtidas da carga de trabalho e uma configuração de cache. Durante cada *quantum* de tempo uma tabela estatística é preenchida. Esta tabela contém o número de acessos e de *misses* por *slot* durante um *quantum* e suas médias. O número de acessos ou de *misses* é considerado grande se ele é maior que média respectiva de todos os *slots*, senão ele é considerado pequeno. Um *slot* GG (grande-grande) possui um grande número de *misses* e de acessos, então podemos considerar que é um *slot* muito acessado com um número de blocos insuficiente. No entanto, se ele tivesse um número pequeno de acessos, seria um *slot* GP (grande-pequeno), que possui um número insuficiente de blocos, podendo ser melhorado, mas não é muito acessado. Um *slot* PG (pequeno-grande) tem uma condição ideal, porque é muito acessado e mantém uma taxa pequena de *misses*. Um *slot* PP (pequeno-pequeno) possui um número pequeno de acessos e de *misses*, então ele pode dar blocos para *slots* mais importantes (com grande número de acessos) ou com uma taxa de *miss* maior.

Inicialmente, a política adaptativa classifica todos os *slots* em uma das possíveis combinações. Então, ela cria uma lista de doadores e uma de receptores. Na lista de receptores, os *slots* GG vêm em primeiro lugar e são seguidos pelos *slots* GP. Então podemos distribuir os blocos de maneira FCFS (*first-come-first-served*), isto é, o primeiro *slot* da lista de receptores recebe um bloco do primeiro *slot* da lista de doadores, então os dois *slots* são retirados das listas. Esta operação é repetida até que uma das listas esteja vazia.

Para analisar nossa arquitetura de cache, utilizamos alguns traces de memória obtidos no Brigham Young University Trace Distribution Center [16], disponíveis na Internet. Para obter estes traces já disponíveis, foi utilizado o BACH [4], um hardware que coleta referencias de memória (endereços) ocorridas em uma execução, incluindo as referencias realizadas pelo sistema operacional.

Em nossas simulações utilizamos somente acessos à memória de dados. Utilizamos seis traces do BYU Trace Distribution Center correspondentes a execuções de benchmarks SPEC [8][20] executados utilizando-se o Windows 2000. Os traces simulados estão descritos na Tabela 1. A configuração da máquina usada é com uma memória principal de 512MB e cache de dados de 64KB. Cada bloco possui 4 palavras e cada palavra possui 32 bits. A associatividade inicial da cache com associatividade reconfigurável é 2 e a máxima é 4. Consideramos um *quantum* que representasse o número de acessos à memória entre uma troca de contexto de um sistema operacional como o Windows.

Tabela 1. Benchmarks SPEC usados nas simulações

Nome	Descrição	Número de acessos
256.bzip	Compressão	3.746.058
186.crafty	Jogador de xadrez	3.861.895
164.gzip	Compressão	3.647.919
254.gap	Teoria de grupo, interpretador	4.058.463
197.parser	Processamento de texto	4.099.236
181.mcf	Otimização combinatorial	3.874.037

A taxa de erro de algumas cargas executadas em caches associativa por conjunto 2-way e 4-way e em nossa arquitetura com associatividade reconfigurável (AR) pode ser observada na Figura 5. Nestas simulações, a reconfiguração ocorre a cada *quantum*. Considerando a diferença da taxa de erro entre as três organizações, podemos dizer que elas podem ser representadas pelos três *traces* apresentados na Figura 5, porque os pares: 256.bzip e 164.gzip, 186.crafty e 197.parser, 254.gap e 181.mcf possuem comportamentos semelhantes.

Analisando-se mais detalhadamente o trace 256.bzip, podemos observar que inicialmente, o número de acessos no primeiro *quantum* é menor na cache com associatividade reconfigurável como era de se esperar, já que ela é inicialmente semelhante a uma cache associativa por conjunto 2-way (Figura 6).

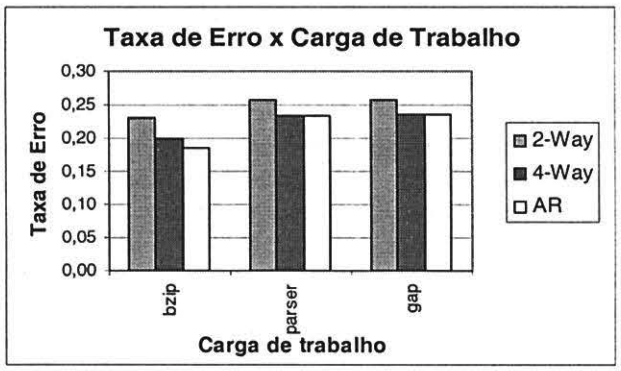

Figura 5. Taxa de erro de algumas cargas executadas em diferentes organizações de cache

No entanto, depois de algum tempo (Figura 6), o número de acessos da cache com associatividade reconfigurável fica maior que a da cache associativa por conjunto 4-way, porque ela possui uma taxa de erro menor (Figura 7), mostrando que a cache se adaptou à carga de trabalho. A redução na taxa de erro permitiu que a execução na cache com associatividade reconfigurável terminasse com um quantum a menos (Figura 6), deixando a execução mais rápida.

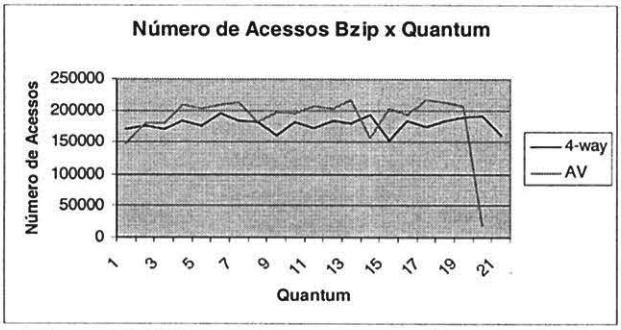

Figura 6. Número de acessos do trace bzip por quantum

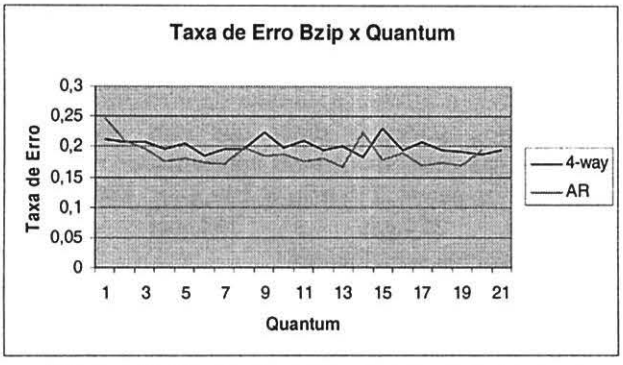

Figura 7. Taxa de erro do trace bzip por quantum

5. Conclusões

Neste trabalho apresentamos a arquitetura de uma cache com associatividade reconfigurável. Analisamos o espaço de memória utilizado para armazenar as *tags*. As cargas utilizadas foram obtidas no BYU Trace Distribution Center. Através das simulações pudemos obter a taxa de erro das cargas de trabalho (benchmarks) em execução em caches com configurações semelhantes as existentes em computadores reais.

Através dos experimentos realizados (simulações), podemos concluir que a cache com associatividade reconfigurável possui desempenho (através da análise da taxa de erro) melhor que uma cache associativa por conjunto com o mesmo número de comparadores sem considerar os *overheads* de configuração. A taxa de erro da cache com associatividade reconfigurável foi menor ou próxima da taxa da cache associativa por conjunto com o mesmo número de comparadores mesmo utilizando-se uma política simples.

Além dos resultados que foram obtidos, podemos dizer que a proposta de uma cache em que é possível realizar mudanças na associatividade de cada *slot* para melhorar o desempenho e com o mesmo tempo de acesso para todas as posições é mais uma contribuição, já que não achamos trabalhos com tais características.

Nossa principal contribuição descrita neste artigo é a proposta da arquitetura de uma cache com associatividade reconfigurável capaz de se adaptar à carga de trabalho e sua análise, tanto em relação o desempenho como em algum tipo de custo (área necessária para armazenar as *tags*).

Como trabalhos futuros podemos citar a implementação de outras políticas de alocação de blocos, a análise de outros fatores da cache, como potencia consumida, estimativas de *overhead* de reconfiguração em diversas camadas arquiteturais e a implementação da cache, considerando a camada arquitetural que apresentar melhor desempenho.

6. Referências

[1] D. H, Albonesi. Et al. Dynamically tuning processor resources with adaptive processing. IEEE Computer, 12(36):49-58, 2003.

[2] B. Batson, T. N. Vijaykumar. Reactive-associative caches, Proceedings of IEEE International Conference on Parallel Architectures and Compilation Techniques, pages 49-60, September 2001.

[3] A. W. Burks, H. H. Goldstine, J. von Neuman. Preliminary discussion of the logical design of an electronic

computing instrument Disponivel em: http://www.cs.unc.edu/~adyilie/comp265/vonNeumann.html

[4] K. Compton, S. Hauck. Reconfigurable Computing: A Survey of Systems and Software, ACM Computing Survey, 34(2):171-210, 2002.

[5] J.K. Flanagan, B. Nelson, J. Archibald, K. Grimsrud. BACH: BYU Address Collection Hardware, the Collection of Complete Traces, Proceedings of the 6th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Edinburgh U.K., September 1992, pp. 128-137.

[6] L. F. W. Góes, C. A. P. S Martins. ClusterSim: A Java Parallel Discrete Event Simulation Tool, IEEE International Conference on Cluster Computing, 2004. (a ser publicado)

[7] J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufman, 3th Edition, 2003.

[8] J.L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium, IEEE Computer, v.33, n.7, July 2000, pp. 28-35.

[9] K. Inoue, T. Ishihara, K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption, Proceedings of the ACM 1999 international symposium on Low power electronics and design, pages 273-275, August 1999.

[10] R. K. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling, John Wiley & Sons, 1991.

[11] T. L. Johnson, D. A. Connors and W. W. Hwu. Run-time adaptive cache management, Proceedings of the Thirty-First Hawaii International Conference on System Sciences, 7(6-9): 774 -775, January 1998.

[12] C. Martins, E. Ordonez, J. Corrêa and M. Carvalho. Reconfigurable Computing: concepts, tendencies and application. In: XXII Jornada de Atualização em Informática (JAI), SBC2003, Vol. 2, 2003, p.339 - 388. (In Portuguese)

[13] M. D. Powell, A. Agarwall, T. N. Vijaykumar, B. Falsafi and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping, Proceedings of 34th ACM/IEEE International Symposium on Microarchitecture, pp. 54-65, December 2001.

[14] A. J. Smith. Cache Memories, ACM Computing Surveys, 14(3):473-530, September 1982.

[15] W. Tang, A. Veidenbaum, A. Nicolau and R. Gupta. Cache With Adaptive Fetch Size, Technical Report ICS-00-16 of University of California, Irvine, April 2000.

[16] R. A. Uhlig, T. N. Mudge. Trace-driven memory simulation: a survey, ACM Computing Surveys, 29(2):128-170, June 1997.

[17] A. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, X. Ji. Adapting Cache Line Size to Application Behavior, Proceedings of the 13th ACM International Conference on Supercomputing, pp. 145-154, Rhodes, 1999.

[18] Brigham Young University Trace Distribution Website: http://traces.byu.edu/

[19] Laboratório de Sistemas Digitais e Computacionais (LSDC) Website: http://www.ppgee.pucminas.br/lsdc/

[20] SPEC - Standard Performance Evaluation Corporation Website: http://www.spec.org/