Um Algoritmo de Reconfiguração da Associatividade em Memórias Cache

Roberto B. Kerr Jr., Edson T. Midorikawa
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, 158, 05508-900
Cidade Universitária, São Paulo, SP
{roberto.kerr, edson.midorikawa}@poli.usp.br

Resumo

A aplicação de dispositivos reconfiguráveis em sistemas de computação de alto desempenho tem se difundido recentemente. Dentre as áreas de pesquisa com maior possibilidade de ganho de desempenho se destacam os projetos com caches reconfiguráveis. Trabalhos anteriores já mostraram a possibilidade de implementar reconfiguração em diversos campos da memória cache, como por exemplo, o tamanho da linha, a associatividade ou o algoritmo de substituição. Neste artigo analisamos uma proposta de um algoritmo de reconfiguração da associatividade. A análise foi conduzida com a utilização de traces do Spec2000 obtidos do BYU Trace Distribution Center. Resultados experimentais mostraram ganhos de desempenho em relação aos caches tradicionais e em relação a um algoritmo reconfigurável proposto na literatura.

1. Introdução

O aumento da capacidade computacional está entre uma das maiores preocupações dos desenvolvedores de processadores. Várias inovações têm sido introduzidas para obter melhoras de desempenho, seja com novas técnicas como os processadores superescalares e VLIW (*Very Long Instruction Word*) e as arquiteturas *multicore*, seja com o aumento no *clock*. Mas essas constantes evoluções acabam causando efeitos indesejados em sistemas computacionais, como é o exemplo da alta latência no acesso à memória criada pelo aumento na freqüência de funcionamento do processador, enquanto a freqüência de funcionamento das memórias não seguiu essa evolução.

Tentando resolver esta deficiência, no final da década de 60, foi proposta a utilização de uma memória de baixa latência junto ao processador, conhecida como memória cache [1]. Pelo seu custo elevado, a memória cache possui um tamanho bem limitado, mas mantendo sua base na heurística em que o processador gasta cerca de 90% do tempo de processamento em apenas 10% do código [2]. Desta forma, caches com tamanhos pequenos, tais como 32KB e 64KB, tem ganhos no desempenho muito convincentes.

Na busca do aumento no desempenho em operações específicas, alguns fabricantes de supercomputadores começaram a lançar computadores com FPGA (*Field Programmable Gate Array*). Incorporando ao sistema uma FPGA Virtex II Pro da Xilinx, a Cray lançou o XD1. Esta arquitetura reconfigurável se mostrou uma ótima solução para implementação de funções de busca, processamento de sinais, manipulação de áudio, vídeo e imagem, criptografia, codificação e decodificação, correções de erro e geração de números aleatórios para simulações de Monte Carlo, além de também poder ser utilizada em outras funções como processamento sísmico, aceleração gráfica, física quântica, simulação de tráfico veicular, entre outras [3].

Outras empresas que vêm utilizando esta técnica são a SGI, com o RASC [4], e a RSC Computers, que lançou máquinas usando a sua tecnologia *IMPLICIT+EXPLICIT*, onde *implicit* é basicamente composto por hardware de lógica fixa como ASICs e DSPs, e *explicit* é composto por *hardware* de lógica reconfiguráveis como as FPGAs [5].

Estes são apenas alguns dos possíveis usos da reconfiguração aplicável à computação, que faz com que um sistema possa melhor se adaptar aos seus atuais requisitos e obter um melhor desempenho [6][7]. Mas a idéia de reconfiguração computacional não está apenas no conceito do uso de *flowwares* e *configwares* [8], e sim também em uma melhor adaptação do sistema as características de uso, em busca de um desempenho melhor. A partir desta idéia, diversos trabalhos vêm

sendo realizados com a idéia de tentar implementar reconfigurabilidade nas memórias cache, com intuito principal de diminuir a taxa de faltas, aumentando assim o desempenho total do sistema.

O objetivo deste trabalho é a proposta de um algoritmo de reconfiguração da associatividade em memórias cache de acordo com a carga de trabalho, para obter o maior ganho de desempenho possível em relação a um mesmo sistema sem reconfiguração.

Este artigo está organizado da seguinte maneira: na seção 2 será feito um breve relato sobre a situação atual das memórias cache reconfiguráveis, e na seção 3 analisaremos uma situação específica de reconfiguração em memórias cache. Na seção 4 iremos apresentar nossa proposta de algoritmo de adaptação em memórias cache e na seção 5 apresentaremos os resultados experimentais comparando nossa proposta com outras arquiteturas, de forma qualitativa e quantitativa. Por fim, na seção 6, concluímos este artigo com nossas conclusões e propostas de trabalho futuros.

2. Caches Reconfiguráveis

Diversos trabalhos foram realizados com memórias cache reconfiguráveis, executando a reconfiguração nos diferentes campos de uma memória cache. Estes trabalhos propõem diversas técnicas para alterar o comportamento de uma cache durante seu modo normal de operação.

Dentre os possíveis campos que podem ser reconfigurados está o tamanho da linha, a exemplo do trabalho proposto por Veidenbaum e outros [9]. O conceito desta proposta é basicamente a implementação de uma grande memória cache, provida de linhas com blocos 8 *bytes* cada e utilizando mapeamento direto. Esta arquitetura de memória cache realiza a sua reconfiguração formando diversas linhas virtuais, que podem usar uma ou mais linhas concatenadas, variando sempre na potência de 2, com um mínimo de 8 *bytes* e o máximo de 256 *bytes*. Sua arquitetura se baseia no princípio que duas linhas adjacentes que possuam o mesmo *tag* (rótulo) possuem uma boa localidade espacial, podendo assim, ser criada uma linha unificada com as duas linhas concatenadas em uma só. Outro princípio desta arquitetura é que ao substituir uma linha, se for observado que metade dela não está sendo utilizada, ou esta linha não possui uma boa localidade espacial, ou ela está em conflito com outra linha. Sendo assim, a linha será particionada em duas linhas distintas para evitar poluição do cache com dados buscados da memória.

Outro campo de pesquisa para memórias cache reconfiguráveis é a política de substituição de linhas, onde dois trabalhos se destacam. O SF-LRU [10], que é uma variação do LRU (*Least Recently Used*). O algoritmo é dividido em duas partes distintas de operação, a operação de leitura da memória e a operação de escrita na memória. Ambas as operações utilizam um registrador que controla a freqüência de utilização, chamado RFCV (*Recency-Frequency Control Value*). Durante a operação de escrita, uma análise é feita sobre o conteúdo do RFCV do último dado da fila com o conteúdo do RFCV de seu antecessor, e se o conteúdo dele for maior, seu antecessor deverá ser eliminado e o conteúdo de todos os RFCV devem ser reiniciados (o algoritmo fornece apenas uma outra chance dos dados mais utilizados de permanecerem na cache), e o dado novo a ser escrito entra no topo da fila, como no LRU. Na operação de leitura, o conteúdo do RFCV do dado lido é incrementado, e o dado recém lido vai para o início da fila, assim como o LRU.

Um outro projeto na política de substituição de linhas utilizou o que se chamou de técnicas adaptativas [11]. A arquitetura deste trabalho não tem como base nenhum dos algoritmos de substituição conhecidos. Sua implementação é obtida com o uso de uma fórmula definida pelo autor que ajuda a identificar qual dos blocos deverá ser substituído. Esta fórmula leva em consideração a freqüência de acessos à linha e o número de *hits* (acertos) ocorridos para obter um resultado que será comparado a um registrador α que define se a memória cache no momento atual estará dando prioridade ao número de *hits* ou ao tempo entre um acesso e outro a uma mesma linha. Além da implementação da fórmula, serão necessários alguns *bits* por linha e registradores a mais para a implementação desta arquitetura. A técnica adaptativa neste algoritmo está na mudança do valor do registrador α com a carga de trabalho do sistema.

3. Reconfiguração na Associatividade

Um outro campo na reconfiguração em caches é a reconfiguração na associatividade. Usando este conceito, o algoritmo Carvalho e Martins [12][13][1] se destaca com sua simplicidade e facilidade de aplicar em *hardware*. Esta arquitetura de cache reconfigurável utiliza uma política de re-adaptação bem simples. A partir de uma memória cache inicialmente definida, a memória cache vai se readaptando à carga de trabalho utilizando intervalos de tempo pré-definidos, em busca da configuração ideal.

[1] Como o algoritmo proposto em [12][13] não foi nomeado pelos autores, iremos nos referir a ele por "algoritmo Carvalho e Martins", utilizando o sobrenome dos autores do trabalho.

O modo de funcionamento é bastante simples, utilizando-se inicialmente uma cache normal por associação em conjuntos com dois registradores em cada conjunto, um fará o registro dos acessos ao conjunto e o outro o registro das faltas. Estes registros serão realizados no que iremos chamar de modo normal de funcionamento. Existem dois modos distintos de funcionamento, o modo normal, referido anteriormente, onde esta memória cache funcionará basicamente como uma memória cache comum, e o modo de reconfiguração a ser chamado em intervalos de tempos pré-determinados[2], onde será executado o algoritmo com sua política de adaptação.

O algoritmo de adaptação inicialmente calcula a média de acessos e a média de faltas de todos os seus conjuntos, definindo assim, uma barreira para classificar os conjuntos pela sua quantidade de acessos e de faltas. Definidas as suas barreiras, cada um dos conjuntos irá receber um rótulo, a partir de sua classificação de acordo com o número de acessos e de faltas. Os possíveis rótulos a serem dados aos conjuntos estão definidos na tabela 1.

Tabela 1 - Descrição de rótulos

Tipo	Faltas	Acessos	Descrição
GG	Grande	Grande	Tem alta prioridade em receber linha
GP	Grande	Pequeno	Tem baixa prioridade em receber linha
PG	Pequeno	Grande	Conjunto estável, não doa e não recebe
PP	Pequeno	Pequeno	Conjunto doador de linha

Os rótulos funcionam para definir a ação a ser tomada durante a adaptação da memória cache. Os conjuntos podem ser rotulados como GG, GP, PG e PP de acordo com sua posição nas barreiras de faltas e acessos.

Os conjuntos rotulados como GG obtiveram uma grande quantidade de acessos e uma grande quantidade de faltas durante o intervalo de tempo analisado. Com essas características, ele se torna um conjunto com alta prioridade para receber uma linha extra.

Os conjuntos GP obtiveram uma grande quantidade de faltas e uma baixa quantidade de acessos durante o intervalo de tempo medido, portanto, eles possuem características de um receptor, por haver uma alta quantia de faltas, mas por seus acessos serem baixos, têm uma baixa prioridade para receber uma linha nova.

[2] Os intervalos de tempo não podem ser nem muito baixos para gerar uma boa estatística para a reconfiguração, e nem muito altos tornando o sistema ineficiente. A escolha do intervalo de tempo ideal está na definição do que é muito pequeno e o que é muito grande.

Os conjuntos classificados como PG são os conjuntos ideais, pois possuem baixa quantidade de faltas e uma alta quantidade de acessos. Estes são considerados como conjuntos que não precisam receber linhas novas e nem doar linhas, apresentando um equilíbrio com relação ao número de faltas e acessos.

Por fim, conjuntos definidos como PP são os reconhecidos como doadores, por possuir uma baixa quantidade de acessos durante o intervalo de tempo medido, e também por possuir uma baixa quantidade de faltas, sendo doadores potenciais.

Tabela 2 - Exemplo de classificação

Conjunto	Faltas	Acessos	Rótulo
S0	10	100	GG
S1	7	30	PP
S2	15	110	GG
S3	8	40	PP
S4	8	50	PP
S5	12	20	GP
S6	3	120	PG
S7	17	100	GG
Média	10	70	

A tabela 2 mostra um exemplo funcional de uma classificação de conjuntos. Neste exemplo, uma cache fictícia com 8 conjuntos (S0, S1, S2, ..., S7) tem suas estatísticas analisadas durante um intervalo de tempo. Essas estatísticas que são descritas pelos valores de seus registradores, definidos pela coluna de faltas e acessos da tabela. A média de acessos e faltas é calculada pela soma de todos os registradores dos conjuntos dividida pelo número total de conjuntos. A partir das médias das barreiras, os rótulos dos conjuntos são definidos, analisando sua posição em relação ao número de faltas e acessos.

Após a obtenção das classificações dos conjuntos, o algoritmo de adaptação tem a função de realizar as doações propriamente ditas. Nesta fase, apenas três restrições são impostas aos conjuntos: um conjunto com apenas uma linha não poderá doar a sua última linha, pois os conjuntos não poderão deixar de existir, mudando a arquitetura final do sistema; a arquitetura dos sistemas será definida por ***x-y way***, onde o valor de *x* é o valor da associatividade inicial, e o valor de *y* é o valor da associatividade máxima, portanto, um conjunto que possui *y* linhas, não poderá receber mais nenhuma linha, pois esse valor *y* é o número de comparadores que o sistema possui; um conjunto pode fornecer apenas uma linha por vez, e o mesmo acontece com os conjuntos receptores, que podem receber apenas uma linha por rodada de adaptação.

Dada uma rodada de adaptação (execução do algoritmo de adaptação), o sistema voltará ao seu modo

normal de funcionamento, até a próxima adaptação. Ao finalizar o algoritmo, os valores dos registradores de faltas e acessos, de todos os conjuntos, serão zerados para obter estatísticas novas, referentes ao novo intervalo de operação.

A duração do modo normal de funcionamento é definida por um intervalo de tempo pré-determinado. Este intervalo de tempo não pode ser relativamente baixo, para poder obter uma estatística confiável, e nem muito grande, para que a função de readaptação seja chamada um número de vezes suficiente para obter uma melhoria do sistema.

A implementação deste sistema ocorre de uma forma bem simples, sendo necessário apenas dois registradores adicionais por linha para obter o número de acessos e de faltas por conjunto, e de um número de comparadores suficientes para analisar o valor máximo de linhas que cada conjunto poderá obter. A média dos acessos e das faltas pode ser calculada facilmente usando um somador de todos os registradores. Como o número de conjuntos é uma potência de 2, a operação de divisão pode ser realizada apenas com um deslocador.

A grande vantagem deste sistema é que ele apresenta ganhos de desempenho (aumento da taxa de acertos) bastante significativos em relação aos sistemas não adaptativos. Além disso, o modo simplificado de funcionamento e sua arquitetura fazem que o *hardware* exigido para a implementação desta memória cache seja bem elementar.

Porém, este algoritmo de adaptação tem algumas desvantagens. Primeiro, em relação à classificação de conjuntos pelo número de faltas e acessos, pelo fato de dividir os conjuntos em apenas dois grupos, acaba colocando na mesma classificação conjuntos com características bastante distintas. No exemplo da tabela 2, os conjuntos S0 e S7 são classificados como GG, então tem a mesma prioridade para receber uma linha. Desta forma é bem provável que o conjunto S0 receba uma linha antes que o conjunto S7. Contudo, o conjunto S7 deveria ter uma prioridade maior que o conjunto S0, por possuir um número de faltas bem mais elevado.

Uma outra situação desfavorável para o algoritmo Carvalho e Martins ocorre quando a carga de trabalho apresenta um padrão de acessos em que um conjunto muda seu padrão doador-receptor, isto é, um conjunto passa de doador de linhas para receptor durante a execução. Como o algoritmo permite apenas a doação de uma linha por vez, a transição do padrão doador-receptor será mais lenta. Este fato faz com que o algoritmo deixe de alcançar melhores ganhos de desempenho.

Trabalhando nestes dois aspectos, é possível obter um algoritmo que produza uma cache com melhor desempenho.

4. Um novo algoritmo de reconfiguração

Nossa proposta de um algoritmo de reconfiguração de memória cache trabalha realizando também a reconfiguração da associatividade da cache. Estamos propondo o algoritmo Kerr e Midorikawa, uma evolução do algoritmo Carvalho e Martins, mas com as melhorias necessárias para fazer que o sistema seja mais criterioso ao definir as prioridades dos conjuntos analisados.

Cada conjunto de linhas da cache também possui dois registradores, um para contar a quantidade de acessos ao conjunto e o outro para contar a quantidade de faltas ocorridas neste conjunto, durante um certo intervalo de tempo de execução normal da cache.

Assim como no algoritmo Carvalho e Martins, em intervalos pré-definidos o cache irá executar o algoritmo de reconfiguração. No algoritmo de adaptação, inicialmente serão definidos os valores de três barreiras de classificação dos conjuntos: a barreira central, a inferior e a superior. Dependendo da posição do conjunto quanto a essas barreiras, eles irão obter seus rótulos para classificação. Os possíveis rótulos são: 2P, 1P, 1G e 2G. Para encontrar o valor da barreira central das faltas, que divide os valores 2P e 1P dos valores 1G e 2G, basta fazer uma média de todos os registradores de faltas e dividir pelo número total de conjuntos. Para encontrar a barreira inferior, basta somar o valor dos registradores de falta dos conjuntos que possuem valor inferior a barreira central, e após isso, dividir pelo número de conjuntos que possuem menos faltas que a barreira central. A barreira superior funciona de maneira similar, somando o valor de todos os registradores dos conjuntos que possuem valor de faltas superior a barreira central, e dividir o resultado pelo número de conjuntos que possuem mais faltas que a barreira central. A figura 1 ilustra os rótulos para classificação de acordo com as barreiras. O mesmo procedimento é efetuado para definir os rótulos para classificação com relação ao registrador de acessos.

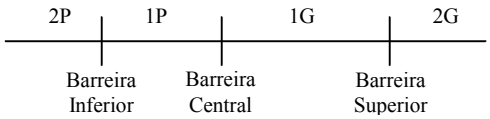


Figura 1 – Rótulos para classificação

Definidas as barreiras, os conjuntos serão classificados pela rotulação de seus registradores de

acordo com o número de faltas e de acessos, onde os conjuntos classificados com as prioridades de menor valor, possuem maior prioridade para receber linhas, e os conjuntos classificados com o maior valor possuem maior prioridade para doar linhas. As possíveis classificações dos conjuntos podem ser observadas na tabela 3.

As prioridades foram definidas assim seguindo o conceito que conjuntos com muitos acessos e muitas faltas necessitam de mais linhas e os conjuntos com poucas faltas e poucos acessos podem doar linhas extras.

Tabela 3 - Classificação

Prioridade	Faltas/Acessos	Ponteiros	Descrição
1	2G/2G		Recebe duas linhas se possível
2	2G/1G		
3	1G/2G		
4	1G/1G		
5	2G/1P	R2	Recebe uma linha se possível
6	2G/2P		
7	1G/1P		
8	1G/2P		
9	1P/2G	R1	Permanece estável
10	1P/1G		
11	2P/2G		
12	2P/1G	D1	
13	1P/1P		Doa uma linha se possível
14	1P/2P	D2	
15	2P/1P		Doa duas linhas se possível
16	2P/2P		

A partir da classificação de todos os conjuntos contidos na memória cache, dois tipos distintos de conjuntos doadores e receptores serão criados, os receptores que recebem 2 linhas, se possível, por reconfiguração e os receptores que recebem uma linha por reconfiguração, e o mesmo para os doadores. Em ambos os casos, internamente aos grupos o conceito de prioridade será seguido, tendo maior prioridade de doar os conjuntos com prioridade de maior valor, e maior prioridade de receber, os conjuntos com prioridade de menor valor. Estes tipos de conjuntos são definidos por quatro ponteiros, dois para os doadores (D1 e D2) e dois para os receptores (R1 e R2). Para a doação, conjuntos com valor de prioridade maior que D2 doam duas linhas se possível, e o valores maiores que D1, doam uma linha. Nos receptores, os conjuntos com prioridade menor que R2 recebem duas linhas se possível, e os menores que R1, recebem uma linha. Os conjuntos menores ou iguais a D1 e maiores ou iguais a R1, permanecem estáveis. Algumas ressalvas precisam ser feitas quanto aos conjuntos: um conjunto doador que possuir apenas uma linha não fará a doação; um conjunto receptor que possuir o valor máximo de *y* linhas em seu conjunto (*x-y way*) não receberá linha extra; conjuntos destinados a receber duas linhas extras, só poderão receber as duas linhas se ao recebê-las não ultrapassar o valor de *y*, e se possuir algum doador que possa doar duas linhas ao mesmo tempo; conjuntos doadores de duas linhas só irão doar duas linhas se possuírem pelo menos 3 linhas em seu conjunto, e se existir pelo menos um conjunto que possa receber as suas duas linhas ao mesmo tempo; em hipótese alguma haverá acúmulo de conjuntos doadores para doar para um mesmo conjunto ou acúmulo de receptores para receber de um mesmo conjunto, neste caso, receptor ou doador receberá apenas uma linha durante a execução do algoritmo, podendo receber ou doar outra linha apenas na próxima reconfiguração.

5. Resultados experimentais

Para realizar uma avaliação experimental da nossa proposta, foi desenvolvido o simulador BobSim [14]. Com este simulador, diversas análises comparativas foram feitas em alguns modelos de memória cache (tradicionais e reconfiguráveis) para se obter uma análise de desempenho. Inicialmente, simulações foram realizadas utilizando modelos de cache tradicionais com mapeamento direto, associativo por conjuntos e totalmente associativo para fins de comparação de desempenho. A seguir foi analisado o algoritmo Carvalho e Martins [12][13], que serviu como parâmetro de comparação de algoritmos reconfiguráveis. Um exemplo destas simulações pode ser observado na figura 2. Essa figura apresenta uma comparação dos desempenhos das organizações tradicionais de memória cache (mapeamento direto, associativo por conjuntos e totalmente associativo) com a configuração *4-8 way* do algoritmo Carvalho e Martins para o *trace* 164 gzip graphic.

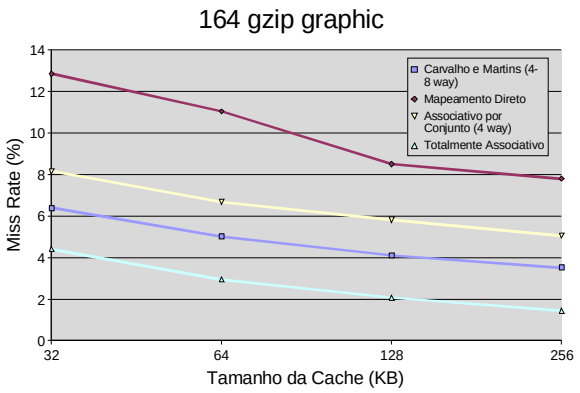


Figura 2 – Desempenho de diferentes caches

Os resultados obtidos mostram que o algoritmo Carvalho e Martins apresenta um desempenho melhor

que as organizações mapeamento direto e associativo por conjuntos, contudo, com um desempenho ainda pior que a organização totalmente associativa. Isto mostra que ainda é possível obter melhorias de desempenho.

5.1. Configuração das simulações realizadas

As simulações foram realizadas a partir de 47 *traces* da BYU Trace Distribution Center [15] obtidos com todos os *benchmarks* SPEC CPU2000 [16], utilizando tanto os *benchmarks* de inteiros quanto os de ponto flutuante. Estes *traces* variam desde operações de compressão usando gzip e bzip2, compilações com gcc usando diversas cargas de entrada, a simulações de propagação de ondas sísmicas e jogos de xadrez. Os *traces* escolhidos foram obtidos em máquinas PIII 733MHz com Windows 2000. Alguns dos *traces* utilizados nas simulações podem ser observados na tabela 4.

Tabela 4 - Alguns *benchmarks* utilizados nas simulações

Benchmark	Tipo	Descrição
164.gzip	Int	Compressão de dados
171.swim	FP	Modelamento de fluidos
175.vpr	Int	Place e Route em FPGA
176.gcc	Int	Compilador C
177.mesa	FP	Biblioteca Gráfica 3D
186.crafty	Int	Jogo de Xadrez
200.sixtrack	FP	Física Nuclear: Projeto de Aceleradores
252.eon	Int	Visualização computacional
253.perlbmk	Int	Programação PERL
254.gap	Int	Teoria de Grupos, Interpretador
255.vortex	Int	Banco de Dados Orientados a Objetos

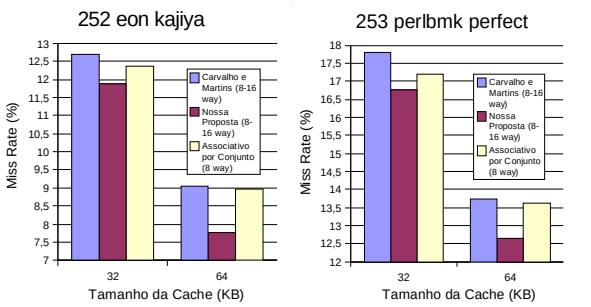
Atualmente, diversos tipos de memória cache podem ser encontrados, variando tanto seu tamanho total, como sua associatividade, tamanho de linha, algoritmo de substituição e até mesmo a arquitetura, mesmo que raramente. Para este trabalho, utilizamos simulações com cache de tamanho variando entre 32KB a 256KB, associatividade inicial variando de 2 a 16 e associatividade final variando de 4 a 32. O tamanho da linha escolhido foi de 32 *bytes*, por ser um valor bastante comum. A arquitetura de memória escolhida foi a de Von-Neumann, onde as memórias de dados e instruções são integradas.

5.2. Análise dos resultados

Os resultados obtidos com essas simulações, em sua maioria, foram como esperado, tendo a memória cache totalmente associativa com o melhor desempenho, e o algoritmo Carvalho e Martins obtendo um melhor desempenho em relação à organização associativa por conjuntos[3]. Porém, em alguns casos, o algoritmo Carvalho e Martins obteve piores resultados se comparados com a AC.

As pioras se deram devido ao fato que os *traces* não seguem um padrão doador-receptor estável, onde os melhores conjuntos estão sempre dispostos a receber linhas, e os piores conjuntos, dispostos a sempre doar linhas. Dado este fato, este algoritmo se mostrou frágil porque conjuntos previamente definidos como conjuntos receptores ou conjuntos doadores levam mais tempo para se readaptarem para um novo padrão de acessos onde os mais prováveis a receber linhas se tornam mais prováveis a doar e vice-versa.

O algoritmo proposto busca resolver este tipo de problema permitindo a doação ou recepção de até duas linhas por reconfiguração. Como pode ser observado na figura 3, nosso algoritmo conseguiu melhoras de desempenho sobre o algoritmo Carvalho e Martins e sobre a AC em diferentes casos.



(a) **(b)**

Figura 3 – Influência do padrão doador-receptor nas simulações realizadas

A figura 3.a apresenta os resultados obtidos para o *trace* Spec2000 252 eon kajiya. De uma maneira geral o algoritmo Carvalho e Martins apresenta resultados piores que a AC. Por exemplo para o tamanho de cache de 32KB a taxa de falta é 0,3% superior a AC. Nesta configuração o nosso algoritmo apresentou resultados melhores que ambas as organizações com uma taxa de faltas de 11,9%, correspondendo a uma melhora de 0,81% em relação ao algoritmo Carvalho e Martins. Um ganho maior foi obtido para um cache de 64KB com uma taxa de faltas 1,29% menor que a taxa do algoritmo Carvalho e Martins.

Resultados semelhantes foram obtidos com o *trace* Spec2000 253 perlbmk perfect. O nosso algoritmo apresentou taxas de faltas menores de cerca de 1% que

[3] Como o termo organização associativa por conjuntos é muito usado nesta seção, iremos adotar a sigla "AC" para designar esta organização de memória cache.

o algoritmo Carvalho e Martins para ambos os tamanhos de cache.

Esta comparação mostra que o nosso algoritmo se adapta mais facilmente às mudanças no padrão doador-receptor das aplicações, que é uma das deficiências do algoritmo Carvalho e Martins.

Nas situações em que o algoritmo Carvalho e Martins apresenta uma melhoria comparado à memória cache AC, nossa proposta busca obter melhoras adicionais de desempenho com a definição de um número maior de tipos de conjuntos. Isto permite diferenciar melhor os vários conjuntos de acordo com o número de acessos e faltas atribuindo um número maior de prioridades. Na figura 4, podemos ver alguns exemplos de casos onde o algoritmo Carvalho e Martins possui um bom desempenho sobre a AC, e uma comparação com a nossa proposta.

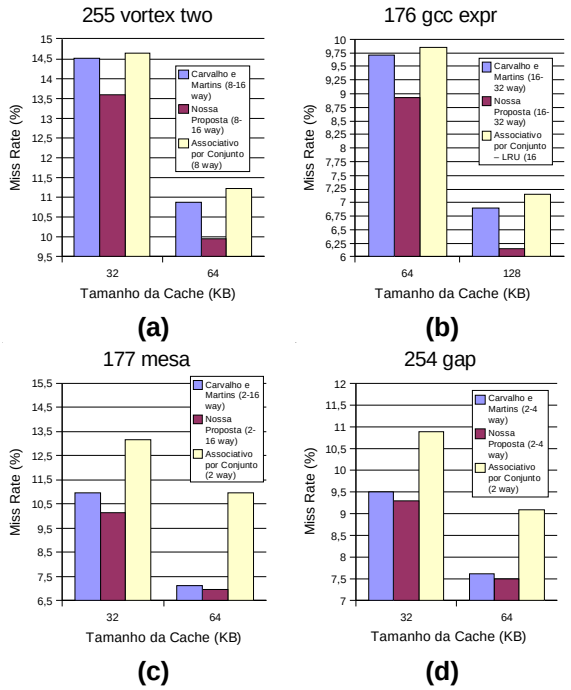


Figura 4 – Influência da prioridade na doação e recepção de linhas

Na figura 4.a, com o *trace* Spec2000 255 vortex two, para tamanhos de cache de 32KB e 64KB, o algoritmo Carvalho e Martins tem um ganho no desempenho, diminuindo a taxa de falta em aproximadamente 0,15% e 0,35%, comparado com o valor obtido com a AC. Os resultados obtidos para o nosso algoritmo mostra uma diminuição da taxa de faltas em relação ao algoritmo Carvalho e Martins em cerca de 0,95% em ambos os tamanhos. Resultados semelhantes foram obtidos para os *traces* Spec2000 176 gcc expr (figura 4.b), o Spec2000 177 mesa (figura 4.c) e o Spec2000 254 gap (figura 4.d) onde o nosso algoritmo apresentou uma taxa de faltas menor que o algoritmo Carvalho e Martins em 0,8%, 0,85% e 0,2% respectivamente.

Estes resultados mostram que nosso algoritmo conseguiu diferenciar melhor os conjuntos com maior necessidade de receber linhas atribuindo uma prioridade maior em relação aos outros conjuntos.

O próximo estudo realizado foi verificar a existência de uma configuração de cache reconfigurável que sempre apresentasse ganhos de desempenho em todas as aplicações. Desta forma esta configuração poderia ser utilizada como uma configuração padrão para as memórias cache reconfiguráveis em sistemas reais.

As simulações realizadas apontaram para uma configuração especial com nossa política de adaptação: caches utilizando a configuração *4-8 way* obtiveram, em todas as simulações, excelentes resultados comparados ao algoritmo Carvalho e Martins e ao associativo por conjuntos. A Figura 5 mostra alguns resultados obtidos por esta configuração.

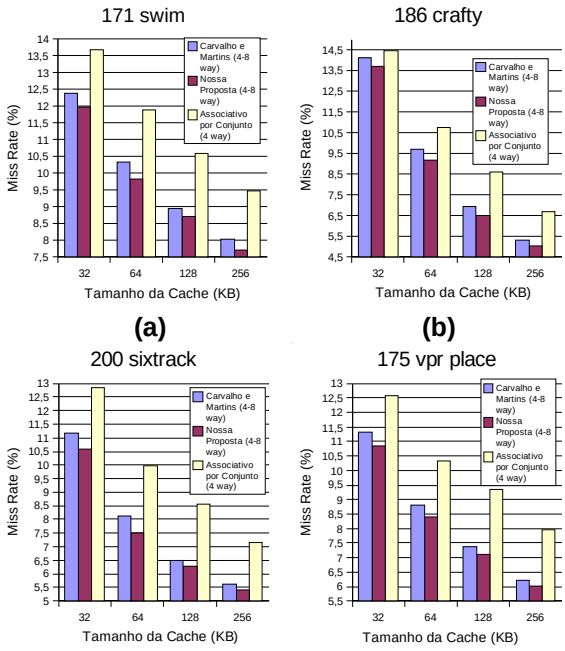


Figura 5 – Configuração 4-8 *way*

Na figura 5.a, o *trace* de ponto flutuante Spec2000 171 swim é utilizado, apresentando uma taxa de faltas de 13,7% no associativo por conjuntos com cache de 32KB, uma taxa de 12,4% no algoritmo Carvalho e Martins e de 12% em nossa proposta. Os resultados seguiram esse padrão de queda para caches de 64KB, 128KB e 256KB, com uma taxa de faltas de 11,9%, 10,6% e 9,5% no associativo por conjuntos, e caindo

aproximadamente 1.5% em todos os tamanhos para o algoritmo Carvalho e Martins, e em nosso algoritmo, caindo mais 0,5% para 64KB, e 0,3% para 128KB e 256KB. As figuras 5.b a 5.d mostram melhoras de cerca de 0,3% a 0,8% nos *traces* Spec2000 186 crafty, SpecFP2000 200 sixtrack e Spec2000 175 vpr place. Resultados semelhantes foram obtidos com todos os outros *traces* analisados do Spec2000.

6. Conclusão e Trabalhos Futuros

Neste artigo fizemos uma proposta de um algoritmo de reconfiguração da associatividade em memórias cache de acordo com a carga de trabalho. A partir de resultados obtidos em nossas simulações conseguimos concluir que nossa proposta consegue obter melhorias no desempenho em relação às organizações de caches tradicionais e ao algoritmo Carvalho e Martins .

Um caso inicial foi analisado onde o algoritmo Carvalho e Martins não conseguia obter um desempenho melhor que o associativo por conjuntos, e nossa proposta obteve melhores desempenhos devido ao fato que nosso algoritmo possui uma característica de melhor recuperação quando ocorre uma inversão no padrão doador-receptor dos conjuntos da memória cache.

Outro caso analisado foi a melhoria do desempenho mesmo nos casos em que o algoritmo Carvalho e Martins já possuía bons resultados comparado ao associativo por conjuntos. Essa melhoria ocorreu devido ao fato que nossa proposta possui uma política de prioridades mais detalhada para classificaçãos de doadores e receptores.

O terceiro estudo realizado mostrou que existe uma configuração de cache reconfigurável que sempre apresenta ganhos de desempenho, a configuração *4-8 way*. Esse resultado é muito importante pois pode ser utilizado como configuração padrão de caches reconfiguráveis.

Uma proposta de trabalho futuro se refere à melhora na obtenção do valor das barreiras central, inferior e superior. Atualmente para se obter os valores das barreiras, é necessário calcular o valor médio dos registradores de acessos e de faltas, o qual apresenta um custo muito elevado para sua implementação em *hardware*. Uma possível alteração seria utilizar um contador para obter inicialmente a barreira central, fazendo com que 50% dos conjuntos estejam antes desta barreira e o resto após. O mesmo seria realizado com a barreira inferior e a barreira superior. Esta mudança precisa ser bem analisada, pois poderia gerar simplificar o *hardware*.

A adaptatividade dos valores dos ponteiros D1, D2, R1 e R2, de modo que o algoritmo tenha um número equilibrado de doadores e receptores, pode levar a uma melhora adicional de desempenho.

7. Referências

[1] Jim Handy. The Cache Memory Book. Morgan Kaufman, 2th Edition, 1998.

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

[3] Application Acceleration with FPGA-Based Reconfigurable Computing. Website: http://www.cray.com/products/xd1/acceleration.html

[4] SGI RASC Technology Website: http://www.sgi.com/products/rasc/

[5] SRC Computers Implicit+Explicit Architecture Website: http://www.srccomp.com/ImplicitExplicitArch.htm

[6] Katherine Compton, Scott Hauck: Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, Vol. 34, No. 2, June 2002, pp. 171-210

[7] C. A. P. S. Martins, E. D. Ordonez, J. B. T. Corrêa, M. B. Carvalho. Computação Reconfigurável: conceitos, tendências e aplicações. In: XXII Jornada de Atualização em Informática (JAI), SBC2003, Vol. 2, p.339-388, 2003.

[8] R. Hartenstein. Why we need Reconfigurable Computing Education: Introduction. 1st International Workshop on Reconfigurable Computing Education (RCeducation), 2006.

[9] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau and X. Ji. Adaptating Cache Line Size to Application Behavior. In Int'l Conf. on Supercomputing, pages 145-154, 1999.

[10] J. Alghazo, A. Akaaboune, N. Botros. SF-LRU Cache Replacement Algorithm. In: The 2004 International Workshop on Memory Technology, Design and Testing (MTDT'04), pages 19-24, 2004.

[11] D. Dasarathan, S. Kulandaiyan. Adaptative Cache Replacement Technique. In: The 9th International Conference on High Performance Computing (HiPC 2002).

[12] M. B. Carvalho e C. A. P. S. Martins. Arquitetura de Cache com Associatividade Reconfigurável. In: V Workshop em Sistemas Computacionais de Alto Desempenho, pp. 50-57, 2004.

[13] M. B. Carvalho e C. A. P. S. Martins. Arquitetura de Memória Cache Reconfigurável. In: VII Workshop em Sistemas Computacionais de Alto Desempenho, pp. 149-156, 2006.

[14] R. B. Kerr Jr., E. T. Midorikawa. Introdução da Computação Reconfigurável e o Uso de Ferramentas no Ensino de Arquitetura de Computadores. Submetido para o II Workshop sobre Educação em Arquitetura de Computadores (WEAC 2007).

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

[16] The Standard Performance Evaluation Corporation Website: http://www.spec.org/