PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós Graduação em Engenharia Elétrica

PROPOSTA E DESENVOLVIMENTO DE UMA ARQUITETURA DE MEMÓRIA CACHE RECONFIGURÁVEL

Milene Barbosa Carvalho

Belo Horizonte
2005

Milene Barbosa Carvalho

Proposta e Desenvolvimento de uma Arquitetura de Memória Cache Reconfigurável

Dissertação apresentada ao programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Carlos Augusto Paiva da Silva Martins
Co-orientador: Petr Iakovlevitch Ekel

Belo Horizonte
2005

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Carvalho, Milene Barbosa
C331p Proposta e desenvolvimento de uma arquitetura de memória cache reconfigurável. / Milene Barbosa Carvalho. Belo Horizonte, 2005.
105f. : il.

Orientador: Carlos Augusto Paiva da Silva Martins
Co-orientador: Petr Iakovlevitch Ekel
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica.

1. Memória Cache. 2. Arquitetura de Computador. 3. Sistemas de Memória de Computadores. I. Martins, Carlos Augusto Paiva da Silva. II. Ekel, Petr Iakovlevitch. III. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDU:681.3-11

AGRADECIMENTOS

Agradeço a todos que participaram de alguma forma no desenvolvimento e conclusão deste trabalho, pela paciência, pelos conhecimentos técnicos e científicos, mas principalmente pelo carinho e atenção não somente dedicados ao trabalho, mas a mim.

Neste grupo, é impossível não destacar a importância do Professor Carlos, meu Orientador em assuntos profissionais, mas também na minha vida pessoal, um exemplo de profissional e ser humano. Além disso, as contribuições de todos os amigos do GSDC e as secretárias e demais funcionários do PPGEE foram de extrema importância, não só para ajudar na dissertação, mas também nos momentos de relaxar da tensão.

Aos meus pais e minha irmã agradeço por entenderem meus momentos de ausência e apoiarem incondicionalmente minhas escolhas. Aos amigos que, mesmo distantes, estavam na torcida por mim.

RESUMO

Atualmente, nos sistemas computacionais tradicionais, para diminuir a diferença de desempenho, entre processadores e memória principal, existem pequenas memórias rápidas conhecidas como memória cache. O sucesso na utilização das memórias cache está relacionado à adequação de alguns parâmetros da memória cache a características da carga de trabalho que é executada nos sistemas computacionais. Alguns parâmetros da memória cache, como tamanho do bloco, tipo de organização e associatividade são definidos no momento da fabricação da cache sem que possam ser alterados posteriormente, na maioria das vezes. No entanto, como as características de uma carga de trabalho variam de aplicação para aplicação, a configuração definida na fabricação e montagem do sistema computacional não é a ideal para todas as cargas de trabalho. O objetivo dessa pesquisa é propor e desenvolver uma arquitetura de cache reconfigurável, isto é, uma memória cache que possa ter alguns parâmetros adaptados à carga de trabalho após a fabricação da cache. Para a verificação da proposta, foi necessário desenvolver uma política de adaptação da cache à carga de trabalho que permite a variação da associatividade da cache independente de slot. Como a verificação e implementação física de uma cache possuem custos e complexidade elevados e a adaptação da cache não pode ser realizada em memórias caches reais, um simulador foi desenvolvido para se analisar o desempenho de caches tradicionais com diversos valores para seus parâmetros e a cache reconfigurável com a política de adaptação proposta. Analisando-se os resultados da simulação realizada com os *benchmarks* SPEC 2000 CPU INT e FP e a política de adaptação implementada, na maioria das simulações realizadas, o desempenho da memória cache reconfigurável foi satisfatório, sendo melhor ou igual ao desempenho de uma memória cache associativa por conjunto com associatividade igual à associatividade máxima da memória cache reconfigurável. Através das análises realizadas é possível perceber que por ser uma política de adaptação simples, ela ainda possui algumas limitações na análise do comportamento da carga de trabalho e no processo de adaptação. No entanto, os resultados apresentados nesta dissertação foram capazes de confirmar nossa hipótese de que uma memória cache reconfigurável capaz de se adaptar à carga de trabalho executada em cada instante de tempo consegue atingir um desempenho melhor que as caches com organizações fixas.

Palavras-chave: memória cache, computação reconfigurável, memória cache reconfigurável.

ABSTRACT

Nowadays, in traditional computer systems, to reduce the performance difference between processors and main memory, there are fast small memories known as cache memory. The successful utilization of cache memories is related to adequacy of some cache memory parameters to workload characteristics executed in computational systems. Some cache memory parameters, as block size, organization and associativity, are defined in cache making time, not being changed latter, most of the time. However, as workload characteristics vary from application to application, the configuration defined in making and assembly time is not optimal for all workloads. The objective of this to research is to design and develop a reconfigurable cache architecture, i.e. a cache memory that can have some parameters adapted to workload after cache fabrication. To proposal verification, it was necessary to develop a policy that adapts a cache to the workload allowing a cache associativity variation in a slot independent manner. As cache verification and physic implementation has high cost and complexity and cache adaptation cannot be done in real cache memories, a simulator was developed to analyze traditional cache performance, with various parameters values, and reconfigurable cache with the proposed adaptation policy. Analyzing simulation results of SPEC 2000 CPU INT and FP benchmarks and the adaptation policy implemented, in most simulations, the reconfigurable cache memory was satisfactory, being better or equal to a set associative cache memory performance with associativity equals to reconfigurable cache memory maximal associativity. Through the analyses that were made, it is possible to realize that as the adaptation policy is simple, it has some limitations to analyze workload behavior and in adaptation process. However, the results presented in this master thesis was able to confirm our hypothesis of that a reconfigurable cache memory, able to self adapt to executed workload on each moment, achieves a better performance than cache memory with fixed organization.

Key-words: cache memory, reconfigurable computing, reconfigurable cache memory.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ms - milissegundos

KB - *kilobytes*

B - *bytes*

ns - nanossegundos

LISTA DE SIGLAS

BYU - *Brigham Young University*

CA - Completamente Associativa

CPU - *Central Processor Unit* (unidade central de processamento)

DSPs - *Digital Signal Processors* (processadores digitais de sinais)

ENIAC - *Electronic Numerical Integrator and Calculator*

FIFO - *First In First Out* (Primeiro a entrar, primeiro a sair)

ILP - *Instruction Level Parallelism* (paralelismo em nível de instruções)

LRU - *Least Recently Used* (menos recentemente usado)

MAC - *Multiply-and-Accumulator* (mutiplicador e acumulador)

MIPS - Milhões de Instruções por Segundo

SO - Sistema Operacional

SPEC - *Standard Performance Evaluation Corporation*

SPEC CFP - SPEC CPU *Float Point*

SPEC CINT - SPEC CPU *Integer*

UCP - Unidade Central de Processamento

SUMÁRIO

1 INTRODUÇÃO

Nesta pesquisa, propomos, desenvolvemos, verificamos, através de simulações, e analisamos a utilização de conceitos de reconfiguração no projeto de arquiteturas de memória cache e de adaptação da mesma à carga de trabalho (*workload*). A utilização da adaptação e dos conceitos de reconfiguração visa melhorar a adequação do funcionamento e da estrutura da cache à carga de trabalho específica, executada a cada momento em um sistema computacional. Quanto mais adequada a cache for à sua carga de trabalho, melhor será o desempenho da cache e, conseqüentemente, do próprio sistema computacional.

Neste primeiro capítulo apresentamos os seguintes tópicos: contexto, problema motivador, objetivos e metas, relevância, motivação, escopo da pesquisa, métodos e materiais, além da estrutura do restante da dissertação.

1.1 Contexto

Diversas inovações arquiteturais têm sido desenvolvidas com o objetivo de melhorar o desempenho dos sistemas computacionais (HENNESSY, 2003) (PATTERSON, 2005). Uma dessas inovações é a utilização do paralelismo em nível de instruções (ILP – *Instruction Level Parallelism*). Uma das técnicas desse tipo de paralelismo é o *pipeline*, muito presente na maioria dos microprocessadores atuais, que permitiu que crescesse muito o número de instruções executadas em um determinado intervalo de tempo. Outro conceito arquitetural presente na maioria dos microprocessadores atuais é a memória cache (SMITH, 1982) (HENNESSY, 2003). Sua utilização tem o objetivo de melhorar o desempenho da memória principal do sistema computacional, baseado nos princípios 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 mais rápida.

O desempenho da memória cache está diretamente relacionado com sua arquitetura e implementação e com a carga de trabalho (*workload*) sendo executada. As características arquiteturais da cache determinam quais são as cargas de trabalho (em função de suas características) que terão melhor desempenho quando executadas (JAIN, 1991). Por exemplo, caches com blocos/linhas grandes são indicadas quando a carga de trabalho possui alto grau de localidade espacial. Outro exemplo é a localidade temporal presente nos laços de repetição:

como as instruções internas ao laço são executadas diversas vezes, o armazenamento destas instruções em uma memória mais rápida melhoraria o desempenho do sistema computacional. As organizações, características e vantagens e desvantagens das caches são descritas no capítulo de Revisão da Literatura (Capítulo 2).

A computação reconfigurável é um paradigma que visa aumentar a flexibilidade e o desempenho do objeto que pode ser configurado (VILLASENOR, 1997) (DEHON, 2000) (COMPTON, 2002) (MARTINS, 2003). Para isso, o objeto reconfigurável assume uma das diversas configurações possíveis. Cada uma delas pode ser mais adequada para uma determinada carga de trabalho em um dado momento. Como a configuração pode ser 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. Uma das maneiras de mudar as configurações do sistema reconfigurável é fazer com que este se adapte à carga de trabalho para que a configuração seja adequada a esta carga. Através da técnica de adaptação, é possível melhorar não só o desempenho, como também os recursos do sistema (ALBONESI, 2003). Desta maneira, a computação reconfigurável permite que o objeto possua um comportamento flexível (variável) obtido através da adaptação e, conseqüentemente, tenha um desempenho melhor que um mesmo objeto com comportamento fixo para diversas aplicações.

1.2 Problema Motivador

A arquitetura das memórias cache presentes nos microprocessadores atuais é fixa, isto é, quando a cache é projetada, sua configuração é definida e a arquitetura da cache não pode ser modificada após a sua fabricação. A escolha da arquitetura e de seus parâmetros arquiteturais é baseada em uma configuração que possua um desempenho bom para uma carga de trabalho média, 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 cargas com diferentes comportamentos de acesso à memória. Como a configuração da cache é fixa, algumas cargas de trabalho não são executadas com um desempenho satisfatório. Além disso, das cargas de trabalho que são executadas com um tempo satisfatório, uma parte destas poderia ter um desempenho melhor.

O problema motivador de nossa pesquisa é que o desempenho das memórias cache utilizadas nos computadores não é otimizado para todas as cargas de trabalho. Isso acontece porque as caches não são flexíveis, pois possuem estrutura e comportamento fixos, isto é, possuem uma única configuração escolhida na fase de projeto da cache e ideal para uma carga de trabalho média.

Analisando nosso problema motivador e diversas situações do nosso cotidiano em que há a adaptação de objetos a fatores externos, acreditamos que o processo de adaptação poderia ser aplicado para a resolução do nosso problema. Conclui-se que uma das maneiras de obter uma melhoria no desempenho da cache é adaptar seu comportamento e estrutura à carga de trabalho do sistema computacional. Para isso, é necessário analisar métricas de desempenho dependentes tanto da arquitetura da memória cache quanto da carga de trabalho.

Os modelos de memória cache utilizados para a análise de desempenho de memória cache encontrados durante a revisão da literatura (REINMAN, 1999) (LENNERSTAD, 2000) (SEN, 2002) utilizam métricas extraídas da carga de trabalho. Neste caso, não é possível analisar o comportamento da memória cache isoladamente, porque são utilizadas métricas obtidas da execução total da carga de trabalho. Como não foi encontrado nenhum modelo de cache independente de qualquer carga (permitindo a análise da cache com uma carga de trabalho real), é necessário construir um modelo comportamental das organizações clássicas de memória cache e da cache proposta para ser usado em simulações para a análise de desempenho das memórias caches.

Portanto, nosso problema secundário é a falta de um modelo para avaliação do desempenho da cache independente da carga de trabalho. Com este modelo é possível usar a abordagem adaptativa sem ficar restrito a métricas que não representam a execução parcial de qualquer carga de trabalho. No entanto, como se deseja que a arquitetura se reconfigure dinamicamente, é necessário criar uma política de adaptação, capaz de analisar as características de uma carga de trabalho e, baseado nisso, determinar uma configuração para a memória cache. Com isso, temos mais um problema secundário: a necessidade de desenvolver uma ou mais políticas de adaptação de memória cache.

1.3 Objetivos e Metas

Baseado nos problemas apresentados anteriormente, o objetivo geral dessa pesquisa é propor e desenvolver uma arquitetura de cache reconfigurável que é capaz de se adaptar à

carga de trabalho em execução possibilitando a melhoria do desempenho na execução da mesma. A verificação e análise dessa arquitetura são feitas através de simulação. Para analisá-la, foram propostos uma organização de memória cache reconfigurável com associatividade variável independente de *slot* e uma política de adaptação dela à carga de trabalho.

1.3.1 Objetivos Principais

Baseado nos problemas levantados, destacamos os objetivos principais da pesquisa apresentada nesta dissertação, por ordem de importância:

a) Definir, propor e desenvolver uma arquitetura de memória cache reconfigurável;
b) Analisar o desempenho da arquitetura de memória cache proposta, por meio de simulação, usando uma política de adaptação definida;
c) Definir, propor e implementar uma política de adaptação da memória cache à carga de trabalho;
d) Definir um modelo de memória cache utilizado em simulações que permitam a análise da execução de uma determinada carga de trabalho e viabilize a verificação da arquitetura proposta;
e) Propor, desenvolver e implementar uma ferramenta de simulação de memória cache com organizações tradicionais e com organização reconfigurável (CacheSim).

1.3.2 Objetivos Intermediários

Dentre os objetivos intermediários da pesquisa apresentada nesta dissertação, destacamos os que julgamos mais importantes, por ordem cronológica:

a) Estudar e analisar o comportamento de arquiteturas de memórias cache relacionando as características das mesmas com as cargas de trabalho;
b) Estudar e analisar modelos para análise de desempenho de memórias cache;
c) Verificar o modelo proposto para simulação de memória cache para realizar a análise de desempenho de memória cache independente da carga de trabalho usada;

d) Estudar e analisar ferramentas de simulação de memória cache;
e) Verificar e testar a ferramenta de simulação de memória cache implementada (CacheSim);
f) Verificar e testar a política de adaptação proposta e implementada;
g) Verificar e simular a memória cache reconfigurável com associatividade variável independente de *slot* proposta.

1.3.3 Metas

Dentre as possíveis metas ou resultados esperados a serem alcançados na pesquisa apresentada nesta dissertação, destacamos os que julgamos mais importantes, por ordem cronológica:

a) Ferramenta de simulação CacheSim;
b) Política de adaptação da memória cache;
c) Arquitetura de memória cache reconfigurável;
d) Organização de memória cache reconfigurável com associatividade variável independente de *slot*;
e) Análise de desempenho da arquitetura de memória cache reconfigurável;
f) Publicação de artigos em congressos nacionais e internacionais com resultados parciais e finais da pesquisa.

1.4 Relevância

A relevância dessa pesquisa está na necessidade de melhoria do desempenho da memória cache. Apesar dos avanços tecnológicos de projeto de memórias, os avanços arquiteturais de memória não acompanharam os do projeto de processadores. Dessa maneira, o acesso à memória é um dos grandes gargalos no processamento (HENNESSY, 2003). Uma das maneiras de se melhorar o desempenho da memória cache, sem uma revolução tecnológica no projeto de memória, é adaptar a cache à carga de trabalho que é executada em cada computador em cada momento. Dessa maneira, utilizando-se os conceitos de

computação reconfigurável, a cache seria flexível e de desempenho superior ao de uma cache fixa.

No Brasil, existem poucos trabalhos na área de melhoria de desempenho de memória cache. Além disso, a abordagem utilizada neste trabalho não foi encontrada em outro trabalho nacional ou internacional. Poucos trabalhos usam a adaptação ou reconfiguração da memória cache para melhoria de desempenho, a maioria dos trabalhos de cache adaptável ou reconfigurável procuram diminuir o consumo de energia. Os trabalhos que apresentam a adaptação ou reconfiguração como uma solução para melhorar o desempenho, normalmente, modificam somente o tamanho da linha. Os principais trabalhos correlatos ao nosso trabalho são apresentados no Capítulo 2.

A memória cache reconfigurável é flexível e permite que seu funcionamento seja adequado a cada carga de trabalho, aumentando o desempenho da mesma. Assim, uma cache flexível pode ser “ideal” para cada carga de trabalho, através da reconfiguração de sua organização à carga de trabalho. A adaptação permite que a configuração da cache seja alterada (reconfiguração) sem que sejam necessárias instruções específicas para a mudança de configuração. Isso significa uma grande facilidade para o usuário do sistema computacional, que não precisa adquirir diversos sistemas específicos para cada carga de trabalho para obter um bom desempenho e não precisa se contentar com um desempenho médio.

1.5 Motivação

Além do interesse pela área, especificamente por memórias caches, a principal motivação desta pesquisa está na importância e relevância da mesma, considerando-se que a exigência de aumento de desempenho dos sistemas computacionais é constante. Apesar dessa exigência, no caso do acesso à memória, as revoluções arquiteturais de memória não acompanharam as revoluções arquiteturais dos processadores. Para se obter o desempenho desejado sem revoluções tecnológicas, uma das soluções é adaptar a arquitetura da cache a cada tipo específico de carga de trabalho.

A adaptação ocorre a todo momento em nossa vida. Além disso, encontrar uma configuração ótima ou melhor para um determinado objeto é um problema que convivemos diariamente de uma forma tão natural que muitas vezes nem o notamos. Desde o momento em que escolhemos uma roupa para sair, até o momento que precisamos tomar decisões complicadas no trabalho, estamos escolhendo um “ponto ótimo” dentre os diversos

pontos/escolhas permissíveis. Inconscientemente utilizamos técnicas de otimização para tomar uma decisão. Essa decisão é tomada considerando-se ao menos um objetivo e algumas restrições.

Além disso, diversas pesquisas sobre computação reconfigurável vêm sendo realizadas dentro do Laboratório de Sistemas Digitais e Computacionais (LSDC) no Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da PUC Minas (FREITAS, 2003) (GÓES, 2004b) (RAMOS, 2004) (POUSA, 2005). Com essas pesquisas, podemos concluir que a computação reconfigurável permite que a flexibilidade de elementos computacionais melhore o desempenho dos mesmos.

1.6 Escopo da pesquisa

Esta pesquisa visa apenas fins acadêmicos e não industriais. Dessa maneira, não faz parte do escopo dessa pesquisa implementar uma memória cache real. O objetivo da pesquisa, num primeiro momento, até a finalização da dissertação, por limitações de tempo, é propor e desenvolver a arquitetura da memória cache reconfigurável, englobando todos os objetivos descritos na seção 1.3. No entanto, os resultados dessa pesquisa podem ser usados por outras pessoas que desejam implementar a arquitetura da memória cache fisicamente em hardware.

As ferramentas desenvolvidas e toda a documentação produzida podem ser usadas auxiliando disciplinas relacionadas com arquitetura de computadores, nos cursos de Engenharia da Computação, Engenharia Eletrônica, Ciência da Computação, etc.

Em termos técnicos e científicos, esta pesquisa se limita à análise de memória cache através de simulação do comportamento de memórias cache nível 1 (L1) de dados e instruções de máquinas monoprocessadas de propósito geral. As simulações serão realizadas com caches de organizações tradicionais (completamente associativa, mapeamento direto, associativa por conjunto) e com a cache reconfigurável com associatividade variável independente de *slot* proposta.

Estas são limitações impostas pelo tempo de desenvolvimento e escopo da pesquisa e não do objeto da mesma. Portanto, esta pesquisa visa verificar se uma arquitetura de memória cache reconfigurável pode ter desempenho melhor que uma arquitetura fixa. Para isso, utilizamos a arquitetura de uma memória cache com apenas uma política de adaptação. Muitas outras políticas podem ser desenvolvidas e os conceitos podem ser usados em outros objetos.

1.7 Estrutura da dissertação

O restante desta dissertação está organizado da seguinte maneira: no capítulo 2 são apresentados a revisão da literatura sobre memória cache, computação reconfigurável e simulação e também os trabalhos correlatos à nossa pesquisa; no capítulo 3 são apresentados a proposta, definição, e desenvolvimento da arquitetura de memória cache reconfigurável e a organização da memória cache reconfigurável com associatividade variável independente de *slot* e a política de adaptação propostas para verificação e análise da arquitetura proposta; no capítulo 4 são apresentados as ferramentas desenvolvidas para a experimentação de nossa proposta, os resultados obtidos através da experimentação e a análise dos mesmos; no capítulo 5, Conclusões, são discutidos os principais resultados obtidos e apresentados as principais contribuições e alguns possíveis trabalhos futuros; finalmente, são apresentadas as referências bibliográficas. Após as referências, apresentamos os apêndices formados pelas tabelas com os resultados das simulações e os anexos contendo artigos publicados e artigos submetidos aguardando aprovação.

2 REVISÃO DA LITERATURA

Neste segundo capítulo, apresentamos uma revisão da literatura relacionada a memória cache, computação reconfigurável e simulação. Além destes tópicos, apresentamos os principais trabalhos correlatos à nossa pesquisa. Este capítulo inicia-se com uma introdução sobre hierarquia de memória. Em seguida, apresentamos as organizações tradicionais de memória cache. Depois disso, apresentamos algumas idéias comuns à análise de desempenho de cache e os tipos de conflitos que existem na execução de cargas de trabalho. Então, apresentamos os conceitos de computação reconfigurável e o objeto reconfigurável mais conhecido, o hardware reconfigurável. Além disso, apresentamos a simulação dirigida por fluxo. Estas primeiras seções têm como objetivo a apresentação de alguns conceitos essenciais ao entendimento do trabalho. No final do capítulo apresentamos os principais trabalhos correlatos à nossa pesquisa em relação ao problema e à solução proposta.

2.1 Hierarquia de memória

> Qualquer um desejaria uma capacidade de memória indefinidamente grande de tal forma que qualquer [...] palavra estaria imediatamente disponível [...]. Nós somos então forçados a reconhecer a possibilidade de construir uma hierarquia de memórias, cada uma com uma capacidade maior que a anterior, mas menos rapidamente acessível. (BURKS, 1962, p. 7, tradução nossa)[1]

Como podemos observar na citação anterior, dos pesquisadores envolvidos no projeto ENIAC (*Electronic Numerical Integrator and Calculator*), considerado pela maioria dos pesquisadores, o primeiro computador de propósito geral, a memória era e ainda é um dos limitadores dos sistemas computacionais tradicionais. Idealmente ela deveria ser grande o suficiente para armazenar qualquer coisa desejada e, ao mesmo tempo, rápida o suficiente, para que seu acesso não exigisse um tempo diferente de zero. Como isso não é possível, os pesquisadores sugeriram a criação de uma hierarquia de memória.

A hierarquia de memória (Figura 1) é composta por diversas camadas/níveis. Entre essas camadas, podem ser variados as tecnologias, o custo por bit, a área ocupada, a velocidade de acesso, a latência, etc. A camada superior da hierarquia é a mais próxima (não em termos físicos, mas em termos da hierarquia) da UCP - Unidade Central de Processamento

[1] […] one would desire an indefinitely large memory capacity such that any particular […] word would be immediately available […]. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

(CPU - *Central Processor Unit*), ela é a camada com menor capacidade de armazenamento, mas é a que possui o menor tempo de acesso. A medida em que as camadas da hierarquia vão se distanciando da UCP, a capacidade de armazenamento cresce e o tempo de acesso aumenta. Isso acontece porque o custo da memória, normalmente, é inversamente proporcional ao seu tempo de acesso, dessa maneira, utiliza-se menos memória de custo elevado em alguns níveis e é possível ter mais memória de custo inferior em outros (PATTERSON, 2005).

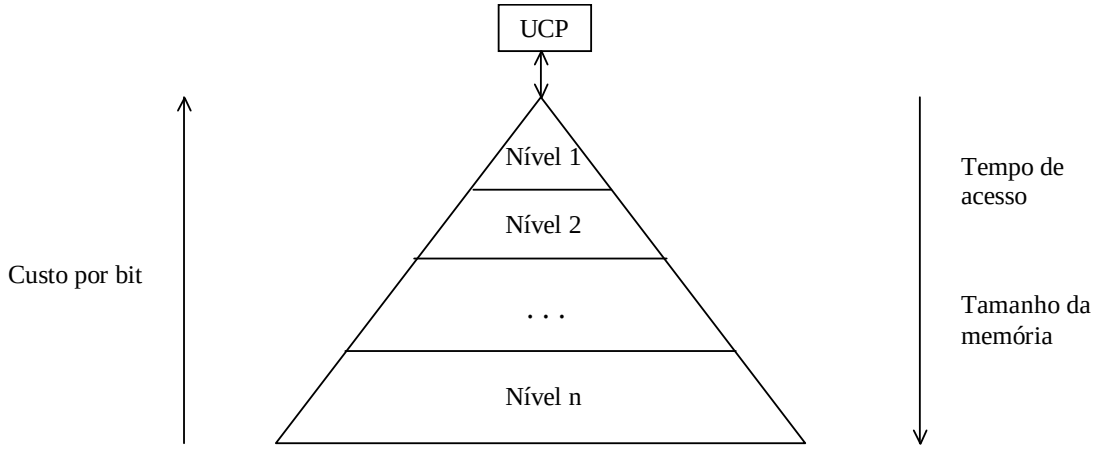

Figura 1 - Representação da hierarquia de memória de um computador. As setas laterais indicam a direção em que o custo por bit, o tempo de acesso e o tamanho da memória aumentam.
Fonte: Elaborada pela autora

Inicialmente, quando se criou o conceito de hierarquia de memória, cada informação contida em um endereço de memória de uma camada era uma cópia da mesma informação que estava contida em um endereço na camada inferior. Isso ainda é feito em muitas hierarquias de hoje, como a do processador Intel Pentium IV, no entanto, existem hierarquias que permitem que a informação seja uma cópia de uma camada inferior, e não da camada imediatamente inferior (não inclusividade), como o processador Athlon da AMD. Além disso, existem políticas que mantêm modificações somente nas camadas superiores, não alterando todas as camadas quando uma certa informação é modificada. No entanto, o uso dessas políticas não altera o objetivo da hierarquia. Desta maneira, com essa hierarquia, tenta-se criar para a UCP a ilusão de uma memória com capacidade equivalente à camada inferior da hierarquia e com velocidade de acesso equivalente à da primeira camada (PATTERSON, 2005).

No entanto, para que a ilusão de uma memória grande e rápida seja possível, o dado/instrução buscado na memória pela UCP deve estar preferencialmente na camada superior da hierarquia. Para que isso aconteça, o princípio da localidade é usado (SMITH,

1982) (PATTERSON, 2005). O princípio da localidade é dividido em dois conceitos: o de localidade temporal e de localidade espacial.

A localidade temporal pode ser definida pela seguinte hipótese: se uma posição de memória foi referenciada, a tendência é que ela seja referenciada novamente em um curto espaço de tempo. Por exemplo, normalmente existem laços de repetição nos programas, provavelmente, as instruções e dados contidos nestes laços serão usados novamente em breve (PATTERSON, 2005). Este tipo de comportamento pode ser verificado em laços de repetição, funções ou métodos muito utilizados, um contador, uma variável que tem seu conteúdo lido ou modificado diversas vezes, etc.

A localidade espacial é a localidade no espaço, isto é, se uma posição de memória foi referenciada, a tendência é que os endereços próximos a essa posição sejam referenciados em breve (PATTERSON, 2005). Esse tipo de comportamento pode ser verificado em programas procedurais ou partes procedurais seqüências, acesso a vetores, matrizes, arranjos (*arrays*) em geral, etc.

Considerando os conceitos de localidade temporal e espacial, o conteúdo da memória é copiado de uma camada inferior para uma superior da hierarquia. Quando a UCP faz uma requisição de informação (conteúdo da memória), ela é procurada na camada mais próxima da UCP, se a informação é encontrada, dizemos que ocorreu um acerto (*hit*). No entanto, se ela não é encontrada em uma camada, dizemos que ocorreu uma falta (*miss*), e a informação é procurada na camada imediatamente inferior. Isso ocorre até que a informação seja encontrada. No pior caso, ela é encontrada na camada inferior da hierarquia. Dessa maneira, o tempo de acesso à memória é variável, quando a informação é encontrada na camada superior da hierarquia, o tempo de acesso é pequeno, no entanto, nas arquiteturas mais comuns, quando é necessário ir para as camadas inferiores, o tempo de acesso é igual à soma dos tempos de acesso de cada uma das camadas percorridas.

2.2 Memória cache

Dentro de uma hierarquia de memória, as camadas que estão entre a memória principal e a UCP, são chamadas de memória cache (SMITH, 1982) (TANENBAUM, 1998) (HENNESSY, 2003) (PATTERSON, 2005). Esse foi o nome dado na primeira máquina comercial que usou uma camada extra entre a UCP e a memória principal. No entanto, esse nome é usado hoje para qualquer gerenciamento de armazenamento que usa localidade de

acesso. Cache significa “um lugar seguro para esconder ou armazenar coisas” (PATTERSON, 2005, p. 358).

Para se compreender as memórias cache, alguns conceitos devem ser explicados. O primeiro deles é o conceito de bloco ou linha. Um bloco corresponde às palavras consecutivas na memória. Dessa maneira, quando falamos de um bloco de tamanho 8, significa que esse bloco contém oito palavras consecutivas da memória. Como cada processador lê palavras de um tamanho específico em bits, o tamanho de um bloco também pode ser medido em bits. Por exemplo, se o processador manipula palavras de 32 bits, um bloco de tamanho 8, possui 256 bits ou 32 bytes. Em uma memória cache, as palavras não são manipuladas separadamente, elas são armazenadas e manipuladas em blocos.

Cada bloco é identificado por parte do endereço das palavras que o compõe. Essa identificação é chamada *tag*. Quando um bloco é armazenado na cache, é armazenada também sua *tag*. O comparador é a parte da cache responsável por ler a *tag* e verificar se o bloco que está armazenado na cache contém a palavra procurada. A posição da palavra dentro do bloco é indicada pelo restante do endereço, o que é chamado de deslocamento ou *offset*. Assim, quando o comparador vai verificar se uma palavra está na cache, ele lê parte do endereço da palavra (*tag*) e compara com as *tags* armazenadas na cache, quando o bloco é encontrado, o restante do endereço da palavra é usado para a localização da palavra dentro do bloco. Uma entrada da cache é composta por um bit que indica se as informações contidas nela são válidas, os dados (bloco) e a *tag*.

Quando um bloco não é encontrado, há uma falta (*miss*), e o bloco deve ser buscado na camada inferior da hierarquia de memória (*memory fetch*). Nesse caso, para se determinar o tempo de acesso à memória, o tempo de acesso da(s) outra(s) camada(s) da hierarquia acessada(s) é somado. Algumas hierarquias fazem buscas paralelas em camadas diferentes, para minimizar o tempo total de acesso em caso de *miss*.

Certas organizações de cache (as organizações serão descritas na seção 2.2.1) possuem posições específicas na cache onde cada bloco pode ser encontrado. Essa posição pode possuir uma ou mais entradas.

2.2.1 Organizações de memória cache

As organizações típicas de cache são mapeamento direito, completamente associativa e associativa por conjunto (SMITH, 1982) (TANENBAUM, 1998) (HENNESSY, 2003)

(PATTERSON, 2005). Na organização do tipo mapeamento direito (Figura 2), cada bloco pode ser encontrado em exatamente uma posição da cache, um *slot* determinado. Dessa maneira, basta um acesso à cache 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, os blocos podem ser encontrados 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, seqüencialmente, 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 (SMITH, 1982) (TANENBAUM, 1998) (HENNESSY, 2003) (PATTERSON, 2005).

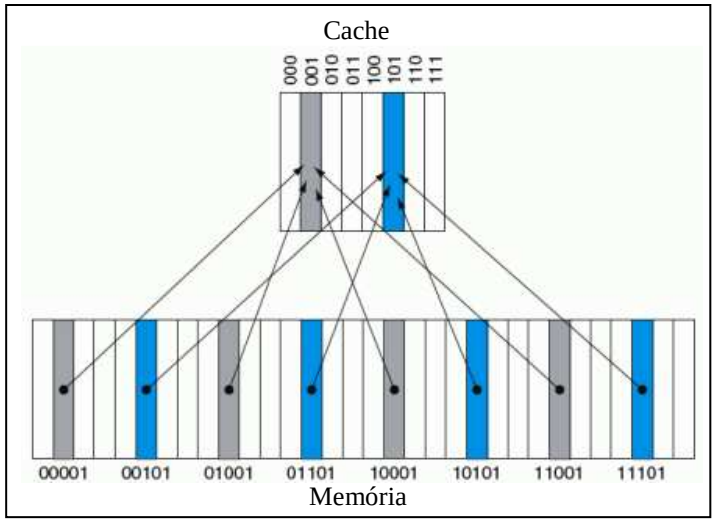

Figura 2 - Estrutura de uma cache do tipo mapeamento direto. Diversas posições da camada inferior (memória) são mapeadas para regiões específicas na memória cache.
Fonte: PATTERSON, 2005

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 *Nway*. Nesse tipo de organização, é necessário checar *n* entradas de um *slot* específico para detectar um cache *miss*/*hit*. Geralmente neste tipo de cache são usados *n* comparadores em paralelo (SMITH, 1982) (TANENBAUM, 1998) (HENNESSY, 2003) (PATTERSON, 2005).

Na realidade, as duas primeiras organizações podem ser vistas como casos especiais da última, como pode ser observado na Figura 3. Uma cache mapeamento direto pode ser considerada uma cache associativa por conjunto *1way*. Uma cache completamente associativa pode ser vista como uma cache associativa por conjunto com um único *slot Nway*, onde *N* é o número de blocos que podem ser armazenados na cache (SMITH, 1982) (TANENBAUM, 1998) (HENNESSY, 2003) (PATTERSON, 2005).

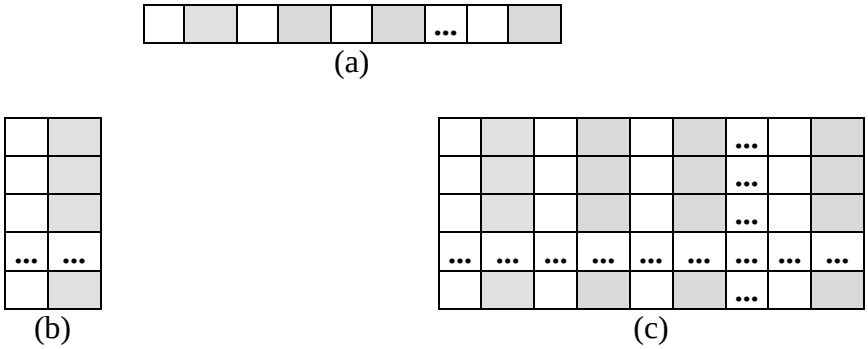

Figura 3 - Organizações de cache: (a) completamente associativa, (b) mapeamento direto e (c) associativa por conjunto.
Fonte: Elaborada pela autora

2.2.2 Desempenho de memória cache

O tempo de execução de uma carga de trabalho, simplificadamente, pode ser definido pela equação 2.1 (PATTERSON, 2005).

$$Tempo = (CiclosGastosPelaUCP + CiclosDeEsperaPelaMemória) \times TempoDeUmCiclo \quad \textbf{(2.1)}$$

Para melhorar o desempenho da execução de uma carga de trabalho, é necessário diminuir o número de ciclos gastos esperando pela memória. A espera pela memória está diretamente relacionada ao desempenho da memória cache, de acordo com a equação 2.2.

$$TempoDeEsperaPelaMemória = Acertos \times TempoDeAcesso + Faltas \times PenalidadeDeFalta \quad \textbf{(2.2)}$$

O número de acertos na memória cache é dado pela equação 2.3.

$$Acertos = Acessos - Faltas \quad \textbf{(2.3)}$$

Geralmente, a penalidade de falta é maior que o tempo de acesso à memória cache, portanto, espera-se que o número de acertos seja maior que o de faltas. Analisando-se as equações 2.2 e 2.3, podemos concluir que, para reduzir o tempo de espera pela memória e,

conseqüentemente, aumentar o desempenho da execução da carga de trabalho, é necessário ao menos uma das ações a seguir:

a) Diminuir o número de faltas (e conseqüentemente aumentar o número de acertos);
b) Diminuir o tempo de acesso à memória cache;
c) Diminuir a penalidade de falta.

Existem inúmeras técnicas para alcançar um ou mais destes objetivos (HENNESSY, 2003), no entanto, neste trabalho estamos interessados em solucionar o problema do número de faltas e, por isso, somente descreveremos as técnicas usadas na redução do número de faltas na seção de trabalhos correlatos (seção 2.5).

2.2.3 Tipos de conflito em memória cache

As faltas que acontecem durante a execução de uma carga de trabalho com a utilização de uma memória cache podem ser subdivididas em três categorias: faltas compulsórias, faltas por capacidade e faltas por conflito (HILL, 1989). Neste modelo, todas as faltas são classificadas como uma dessas categorias, os três Cs (PATTERSON, 2005).

As faltas compulsórias existem em todas as execuções e são também conhecidas como falhas de partida a frio (PATTERSON, 2005). Elas ocorrem quando é a primeira vez que se deseja acessar uma palavra de um bloco que ainda não esteve na cache durante a execução. Este tipo de falta só pode ser reduzido aumentando-se o tamanho do bloco ou utilizando-se técnicas de pré-busca (*prefetching*) (HILL, 1989).

As faltas por capacidade ocorrem quando não existe mais espaço na memória cache onde o bloco possa ser armazenado. Então um bloco é substituído e, quando este bloco é novamente acessado, e ele não está mais na memória cache, ocorre a falta por capacidade. Este tipo de falta pode ser reduzido aumentando-se o tamanho da memória cache (HILL, 1989).

As faltas por conflito, também chamadas de falta de colisão (PATTERSON, 2005), ocorrem somente em caches em que um bloco possui um determinado conjunto de posições onde ele pode ser armazenado. Portanto, este tipo de falta não ocorre em caches completamente associativa. Nas caches em que diversos blocos competem por um conjunto de posições determinadas, ocorre uma falta por conflito quando um bloco que está na cache é substituído por outro, sem que toda a capacidade da memória cache tenha sido utilizada, e

depois o primeiro é acessado novamente. Com um aumento no tamanho da cache sem uma alteração em sua associatividade, aumentando o número de *slots*, ou aumentando-se a associatividade sem alterar o tamanho da cache, é possível diminuir o número de faltas por conflito (HILL, 1989).

2.3 Computação Reconfigurável

O crescente aumento do uso da computação e a conseqüente aplicação nas mais diversas áreas do conhecimento vêm gerando um aumento de demanda computacional. Essa exigência por mais recursos estimula constantes pesquisas em busca de melhorias nos sistemas computacionais. Para se atender toda essa demanda de recursos computacionais, muitas técnicas foram desenvolvidas e são aplicadas tanto em software como em hardware. Um desses esforços deu origem à computação reconfigurável (MARTINS, 2003) (COMPTON, 2002) (DEHON, 1999).

O objetivo da computação reconfigurável é permitir que os sistemas computacionais ou parte deles sejam alterados para estados não previstos no momento da criação e construção dos mesmos para que sejam específicos para uma determinada aplicação num dado momento. Teoricamente, os princípios da computação reconfigurável podem ser aplicados em qualquer nível da arquitetura de um sistema computacional, isto é, pode-se ter softwares reconfiguráveis, sistemas operacionais reconfiguráveis e hardware reconfigurável, por exemplo (MARTINS, 2003).

Os novos estados dos objetos (sistema computacional, software, hardware, etc.) podem ser obtidos através de uma mudança na estrutura do objeto em questão, modificando seu comportamento. O comportamento é determinado pela configuração dos blocos lógicos, que podem ser reorganizados de diversas maneiras. Além dos blocos lógicos, existem ligações entre eles (interconexões) que também podem ser reconfiguradas. Isso quer dizer que a única restrição que os sistemas reconfiguráveis têm está sobre a forma e a quantidade de blocos lógicos, não estando limitados as idéias que o desenvolvedor teve no momento do projeto ou da fabricação/implementação do objeto.

O comportamento do sistema reconfigurável pode mudar dinamicamente de acordo com a sua forma ou conexão espacial (DEHON, 1999). Dessa maneira, através da flexibilidade, é possível alcançar um desempenho maior que o uso de um sistema fixo,

geralmente configurado para uma situação média e não uma ideal para cada uma das possíveis soluções.

Um sistema computacional tradicional é dividido em camadas: aplicação, algoritmo, linguagem, compilador, sistema operacional, arquitetura, conjunto de instruções e micro-arquitetura (PATTERSON, 2005). Os conceitos de reconfiguração podem ser aplicados não somente ao hardware, mas em qualquer uma dessas camadas.

Nosso grupo de pesquisa está trabalhando para desenvolver os conceitos de computação reconfigurável também em software. Já conseguimos bons resultados relacionados ao desempenho de escalonamento paralelo de tarefas (GÓES, 2004a) (GÓES, 2004b), funções de comunicação coletiva (RAMOS, 2004), consistência de objetos (POUSA, 2005) e esperamos aplicar o conceito de computação reconfigurável nas diversas camadas. Em nossos trabalhos, conseguimos determinar três camadas que compõem um sistema reconfigurável: Camada Básica, Camada Reconfigurável e Camada de Controle de Configuração. Estas camadas estão representadas na Figura 4, exemplificadas para uma arquitetura de algoritmo reconfigurável. No entanto, elas são válidas para sistemas reconfiguráveis de qualquer camada arquitetural.

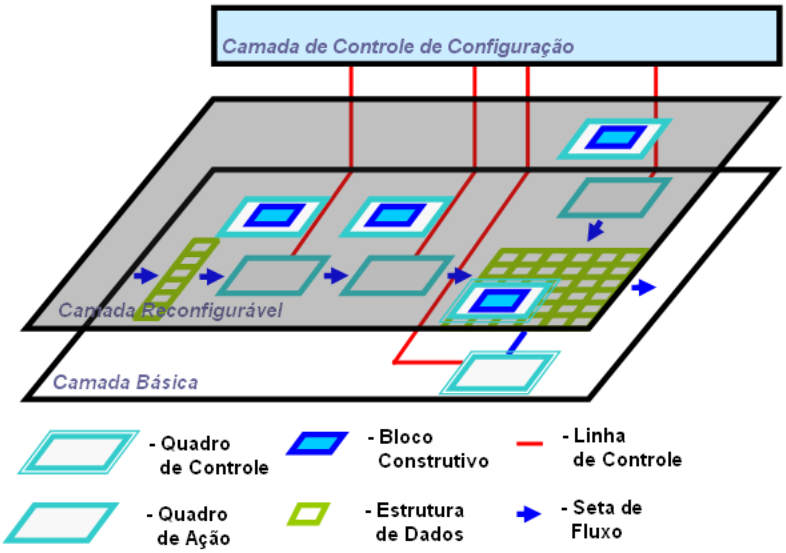

Figura 4 - A arquitetura geral de um software reconfigurável
Fonte: GÓES, 2004b

Na camada básica, encontra-se a estrutura fixa. No caso do software reconfigurável ela é composta pelos quadros de ação e estruturas de dados. A camada reconfigurável possui a lógica que determina o comportamento do sistema reconfigurável (blocos lógicos e interconexão). No software reconfigurável, os blocos construtivos de cada quadro determinam as ações do software. A camada de controle de configuração determina qual a configuração,

dentre as permissíveis, para o sistema reconfigurável deve ser usada em um determinado momento.

A camada de controle de configuração pode ser implementada de diversas maneiras diferentes. A única regra que existe é que ela deve ter, como saída, a configuração do elemento reconfigurável. Ela pode ou não conter entradas. Dentre as possíveis entradas, podem existir monitores do sistema, que permitiriam uma análise das necessidades e a escolha de uma configuração ótima, ou uma simples série de opções, da qual o usuário deve escolher uma. Existe uma infinidade de implementações para essa camada, no entanto, estamos interessados em um tipo especial de implementação: a que permite a adaptação do elemento reconfigurável ao seu ambiente de execução (ALBONESI, 2003).

2.3.1 Hardware Reconfigurável

Atualmente, a computação reconfigurável vem sendo aplicada, principalmente, sobre o hardware. Para isso são usados os dispositivos reconfiguráveis. Estes, incluindo os FPGAs (*Field-Programmable Gate Arrays*), contêm um arranjo de elementos computacionais, nos quais a funcionalidade é determinada pela programação de bits de configuração (*biststreams*). Esses elementos, conhecidos como blocos lógicos (ou construtivos), são conectados usando um conjunto de recursos de roteamento que também são programáveis. Dessa maneira, circuitos digitais podem ser mapeados para o hardware reconfigurável, computando funções lógicas do circuito dentro dos blocos lógicos e usando o roteamento configurável para conectar os blocos e formar o circuito necessário (COMPTON, 2002) (MARTINS, 2003).

2.4 Simulação

Existem três classes de técnicas de análise de desempenho: experimentação, modelagem analítica e simulação (JAIN, 1991). Sistemas complexos do mundo real podem ter alto custo de implementação para experimentação, por sempre necessitarem de um ambiente real. Além disso, como os sistemas computacionais são estocásticos, algumas medições e o isolamento das características tornam-se praticamente impossível.

Sistemas complexos do mundo real geralmente são de difícil descrição através de um modelo matemático. Para a construção do modelo, normalmente, é necessária uma grande experiência com o sistema que se deseja modelar. Além disso, um modelo matemático pode exigir simplificações que descaracterizem os sistemas.

Em uma simulação, utiliza-se um computador para resolver numericamente um modelo de simulação do sistema computacional e os resultados são coletados para se estimar as verdadeiras características desejadas do sistema (JAIN, 1991). A simulação permite comparar algumas alternativas de características do sistema computacional com diversas cargas de trabalho. Então a técnica de simulação aparece como uma alternativa com custo menor que a experimentação e mais precisa que um modelo analítico (considerando-se um mesmo esforço para o desenvolvimento dos modelos) (JAIN, 1991) (LAW, 1991).

2.4.1 Simulação dirigida por fluxo

Um dos tipos de simulação é a simulação dirigida por fluxo que tem como entrada um *trace* (UHLIG, 1997). *Trace* é um conjunto/fluxo de eventos de um sistema real ordenados temporalmente (JAIN, 1991). Esse é o tipo de simulação mais usado para a análise de caches. O *trace* é independente do sistema que está sendo avaliado pela simulação. No caso de memória cache, os *traces*, geralmente, são compostos pelo tipo de acesso (leitura ou escrita), posição do acesso (endereço de memória), tipo de busca (dado ou instrução) e, no caso de escrita, o valor que deve ser escrito. Isso define a carga de trabalho usada para analisar o desempenho de uma memória cache.

Como esse tipo de simulação é determinística, ela possui alta credibilidade. Além disso, como o *trace* é determinístico, o modelo total possui menos variância do que aqueles que usam uma carga de trabalho não determinística. Com isso, com poucas simulações é possível obter uma estatística confiável. Esta característica é vantajosa também em relação à comparação de diferentes configurações de sistema para uma mesma carga de trabalho. Em outros tipos de simulação, em que a carga de trabalho é gerada com algum tipo de aleatoriedade, este tipo de comparação torna-se difícil.

Além disso, a verificação de uma simulação é relativamente fácil, podendo ser comparada com o ambiente do qual o *trace* foi retirado ou até com uma execução algorítmica do comportamento do sistema. O *trace* é a carga de trabalho real, não sofrendo simplificações ou correndo riscos de erros como acontecem em outros tipos de simulação. A carga de

trabalho é bem detalhada e, por isso, é possível verificar o impacto de qualquer modificação no sistema simulado.

Apesar de todas as vantagens descritas anteriormente, a simulação dirigida por fluxos possui uma grande desvantagem: como o *trace* é uma representação da carga de trabalho muito detalhada, muitas vezes, para armazená-lo gasta-se muita memória. Além disso, não é possível fazer pequenas mudanças na carga de trabalho sem trocar o *trace* inteiro.

2.5 Trabalhos correlatos

Nesta seção apresentamos os principais trabalhos correlatos à nossa pesquisa. Podemos considerar três tipos de trabalhos correlatos: correlatos ao problema, correlatos à solução e correlatos aos dois ao mesmo tempo. Os trabalhos correlatos ao problema, isto é, à melhoria do desempenho de memória cache são inúmeros (HENNESSY, 2003), e tentam resolver um dos problemas apresentados na seção 2.2.2. Os trabalhos correlatos à solução são todos os trabalhos que usam a reconfiguração de algum objeto como meio de adaptação ao ambiente, nas diversas áreas do conhecimento. Não apresentaremos estes trabalhos para não nos estendermos em assuntos não relativos ao desempenho de memória cache. Os trabalhos correlatos ao nosso problema e à nossa solução são aqueles que usam de reconfiguração, adaptação ou variação da organização da memória cache como maneira de se obter uma melhora no desempenho da mesma. No restante desta seção, apresentamos os principais trabalhos correlatos ao nosso problema ou à nossa solução.

Em sistemas embarcados, como se conhece o hardware alvo no momento do desenvolvimento do software, este último pode ser otimizado para aproveitar todos os recursos do primeiro, garantindo o melhor desempenho possível para um determinado par hardware e software (WOLF, 2003). Em alguns casos, além da otimização realizada em software, ainda podem existir otimizações em hardware, que permitam um ganho em desempenho e em consumo de energia (ZHANG, 2003) (ZHANG, 2004).

Apesar da relativa facilidade de otimização de software de sistemas embarcados, softwares desenvolvidos para serem executados em máquinas de propósito geral não podem ter este mesmo “privilégio”. Como estes softwares são desenvolvidos para uma grande variedade de processadores de propósito geral, é muito difícil conseguir uma otimização do software que seja ideal para todas as arquiteturas em que o software pode ser executado. Por isso, muitos trabalhos procuram otimizar o desempenho do sistema computacional como um

todo melhorando o desempenho de alguns módulos de hardware, independente do software que é executado. Como estamos interessados no desempenho da memória cache, nesta seção, analisamos alguns trabalhos que procuram aumentar o desempenho das caches de sistemas computacionais de propósito geral.

Analisando o material bibliográfico sobre cache, concluímos que não existe um consenso entre a diferença de uma cache reconfigurável e uma adaptável. Portanto, nesta seção os trabalhos que utilizam qualquer uma dessas expressões no contexto de memórias cache de processadores de propósito geral são apresentados. 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 adaptável/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, como é feito, por exemplo, em (LEE, 2002).

Praticamente todos os trabalhos estudados usam monitores (JOHNSON, 1998) para obter informações e estatísticas da carga de trabalho durante sua execução. No entanto, os monitores precisam ser implementados, em software/aplicativo, no sistema operacional ou em hardware. Nos dois primeiros casos, existe uma sobrecarga (*overhead*) no processamento causado pelo monitor e no terceiro caso, existe o custo do desenvolvimento e implementação do monitor em hardware. Um algoritmo que escolhe a nova configuração da cache para uma carga de trabalho usa as informações do monitor. Uma 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 através de sua configuração.

Considerando a localidade espacial, existem trabalhos que apresentam mudanças no tamanho do bloco/linha (VEIDENBAUM, 1999), outros que modificam o tamanho de busca (*fetch*) (TANG, 2000) e ainda os que permitem unificar estas duas abordagens (LEE, 2002). Nestes 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. Esta abordagem é chamada por seus criadores de “adaptação”.

Na abordagem que realiza adaptação do tamanho do bloco (VEIDENBAUM, 1999), 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 (TANG, 2000), quando ocorre uma falta na cache, 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 utilizam a cache para realizar outras tarefas, como a de auxilio no processamento ou os trabalhos que modificam a associatividade da cache. No primeiro grupo existem trabalhos principalmente nas áreas de processamento multimídia (RANGANATHAN, 2000), processamento de sinais digitais (KIM, 2001) (KIM, 2002) e banco de dados (TANAKA, 2004).

O trabalho descrito em (RANGANATHAN, 2000) propõe o que os autores chamam de cache reconfigurável. Esta cache pode ser particionada para que uma parte seja usada como memória cache e outra como uma memória auxiliar na execução da carga de trabalho, sendo usada para diferentes aplicações. Dentre estas aplicações estão o auxílio ao hardware (com a utilização de *lookup tables*), a pré busca de dados (*prefetching*) e memória controlada pelo compilador ou pela aplicação. Neste trabalho, a partição da memória cache é feita em *ways*, isto é, cada banco de memória que representa um *way*, pode ser usado como parte da memória cache ou como uma memória específica. A grande restrição desta proposta é que a parte da cache reconfigurável que não tem a função de cache, somente pode ter função de memória.

A cache funcional reconfigurável, apresentada em (KIM, 2001) e (KIM, 2002) converte dinamicamente parte da memória cache em uma unidade computacional especializada. Quando isso acontece, a capacidade da memória cache é reduzida. Desta maneira, pode-se dizer que através da reconfiguração, a associatividade da memória cache é configurada para deixar espaço para a implementação das unidades especializadas. Alguns exemplos destas unidades implementadas são funções primitivas de processadores de sinais digitais (DSPs - *Digital Signal Processors*) como multiplicar-e-acumular (MAC - *Multiply-and-Accumulator*) e aritmética distribuída.

Em processamento de banco de dados assim como em aplicações multimídia, a utilização da memória cache pode ser contestada por causa da natureza destas aplicações que

apresentam, em alguns casos, baixa localidade temporal (TANAKA, 2004). Por isso, a cache reconfigurável descrita em (TANAKA, 2004) utiliza a memória cache como um *buffer* do tipo fila (FIFO – *First In First Out*) adequado para receber fluxos de dados seqüenciais (*streaming*), típicos das aplicações citadas anteriormente.

Alguns trabalhos, como (SANGIREDDY, 2004), além de usar técnicas como as citadas anteriormente, que usam a memória cache para executar outras funções, utilizam-se da não necessidade do uso total da área do dispositivo destinada à cache para reduzir o consumo de energia gasto pelo sistema computacional. Isso é possível porque ao manter toda a memória cache ligada para manter os dados nela armazenados (já que ela é uma memória volátil), consome-se muita energia.

Muitos trabalhos foram desenvolvidos com a preocupação de se diminuir o consumo de energia como o trabalho apresentado anteriormente (SANGIREDDY, 2004). Estes trabalhos apresentam a diminuição da associatividade da cache para minimizar a quantidade de energia consumida no uso da cache, enquanto mantém o alto desempenho (ALBONESI, 1999) (INOUE, 1999) (POWELL, 2001). Outros, no entanto, acreditam que o custo/benefício de uma diminuição no consumo de energia permite até uma queda no desempenho (KAXIRAS, 2001). 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) que não contêm palavras necessárias. 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.

Além disso, diversos esforços vêem sendo realizados para reduzir o consumo de energia através não só de características da carga de trabalho como também da tecnologia de implementação dos dispositivos de memória (KIM, 2005). Nestes casos, a tecnologia de implementação das memórias pode ser usada em conjunto com nossa proposta em um trabalho futuro, já que nosso objetivo principal neste trabalho é o desempenho da memória cache.

Em (ZHANG, 2004), além da diminuição da associatividade da memória cache através da desativação de *ways* para reduzir o consumo de energia, ainda é possível "modificar" o número de *slots* da cache. Isto é feito através da concatenação dos *ways*. Portanto, reduzindo-se a associatividade da cache é possível aumentar o número de *slots* da mesma.

Apresentamos, até o momento, caches com modificações em relação às tradicionais, no entanto, com a arquitetura baseada em uma das organizações tradicionais apresentadas na seção 2.2.1. Uma primeira organização de memória cache que se difere destas é a cache vítima (*victim cache*) (JOUPPI, 1990). Na realidade, esta cache é um pequeno *buffer* usado em conjunto com outra cache de organização tradicional. Este *buffer* guarda os blocos que recentemente foram retirados da cache principal. O *buffer* e cache principal são acessados ao mesmo tempo. Se o dado procurado é encontrado na cache vítima, o dado é retornado à UCP e gravado na cache principal. Desta maneira, a cache vítima funciona como um aumento da associatividade da mesma, com uma política de inserção e substituição diferente dos outros *ways* da cache principal.

O sistema de memória cache descrito em (KURPANEK, 1994) é composto de uma cache mapeamento direto e um *buffer* completamente associativo acessados em paralelo. Em caso de falta, o bloco é armazenado somente no *buffer* e é promovido para a cache mapeamento direto se ele exibir localidade temporal. A política de substituição utilizada no *buffer* é a FIFO.

A cache seletiva (GONZÁLEZ, 1995) consiste de duas caches e uma tabela de predição. Uma das cache é baseada na localidade espacial (com blocos grandes) e outra na localidade temporal (com blocos pequenos). Através da tabela de predição, determina-se o tipo de localidade característica de um acesso à cache e, através dela, determina-se se o bloco acessado deve ser armazenado na cache e, se for armazenado, em qual das duas caches ele deverá ser armazenado.

Em (LEE, 2002) um sistema de cache chamado DASAT (*Dynamically Aggressive Spatial and Adaptive Temporal*) é proposto. Este sistema é composto por duas caches mapeamento direto com um bloco pequeno e uma cache completamente associativa com bloco grande no mesmo nível de cache. O tamanho de *fetch* é variável dinamicamente, explorando a localidade espacial da carga de trabalho. Através de monitores, alguns blocos são mantidos nas cache mapeamento direto através da análise da localidade temporal. As caches mapeamento direto possuem blocos pequenos, reduzindo o consumo de energia. Cada bloco grande da cache completamente associativa pode ser configurado como múltiplos blocos pequenos, garantindo flexibilidade ao sistema. Através dos mecanismos descritos, a cache pode se adaptar à carga de trabalho melhorando o desempenho da mesma.

Dentre todos os trabalhos correlatos analisados, apenas um trabalho indica claramente a utilização de adaptação da associatividade como meio de se obter uma melhoria no desempenho da memória cache. Este trabalho, a *Reactive-Associative Cache* (r-a cache)

(BATSON, 2001) 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 de acerto (*hit*) maior 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 acerto 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). Além disso, ela possui as mesmas desvantagens das caches associativas por conjunto quando ocorre conflito.

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

3 ARQUITETURA DE CACHE RECONFIGURÁVEL

Neste capítulo, apresentamos a proposta e a definição da arquitetura de memória cache reconfigurável. Além disso, apresentamos a definição da política de adaptação. Posteriormente, apresentamos a proposta e a definição da memória cache reconfigurável com associatividade variável independente de *slot* e a política de adaptação por nós proposta para a adaptação desta memória cache à carga de trabalho.

3.1 Introdução

Idealmente, a memória principal de um computador deveria ser rápida o suficiente, para que seu acesso não exigisse um tempo diferente de zero (BURKS, 1962). No entanto, como isso não é possível, a memória cache é utilizada visando à diminuição do tempo médio de acesso à memória, criando para o processador a "ilusão" de uma memória principal mais rápida do que ela realmente é. Portanto, quanto melhor a utilização da memória cache, melhor será a "ilusão", isto é, o tempo médio de acesso à memória poderá ser menor e, conseqüentemente, o tempo de execução das aplicações também.

Qualquer pessoa que usa um sistema computacional espera que suas aplicações sejam executadas o mais rápido possível, e o tempo de resposta é uma das principais medidas de desempenho de um sistema computacional. Portanto, a qualidade de um sistema computacional depende do tempo que cada um de seus componentes gasta para executar uma determinada tarefa. Grande parte do tempo de execução de uma aplicação está associada aos acessos à memória. Se uma memória cache consegue reduzir o número de acessos à memória principal, então o tempo de resposta da aplicação pode diminuir consideravelmente. A otimização de uma memória cache está relacionada, principalmente, com a maximização da taxa de acerto (*hit*) e a minimização do tempo de acesso (SMITH, 1982).

Para que a memória cache reduza o número de acessos à memória principal (maximize a taxa de acerto), é necessário que a carga de trabalho possua características que permitam que a memória cache armazene o maior número de palavras que serão acessadas pelo processador. Nas cargas de trabalho atuais, tais características são a localidade espacial e temporal. Baseado nestas duas características, as memórias cache projetadas atualmente tentam minimizar o número de faltas de cache calculando-se qual a configuração seria mais

adequada para valores médios para estas características das cargas de trabalho. No entanto, algumas cargas de trabalho não se comportam como esta média, gerando um desempenho aquém do esperado.

3.2 Proposta da Arquitetura de Memória Cache Reconfigurável

Analisando nosso problema motivador e diversas situações do nosso cotidiano em que há a adaptação de objetos a fatores externos (por exemplo, a pupila que se dilata ou contrai de acordo com a luminosidade do ambiente), acreditamos que o processo de adaptação poderia ser aplicado para a resolução do nosso problema. Baseado nisso, concluímos que uma das maneiras de se 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. A variação dos parâmetros da cache, após seu projeto, pode ser obtida através da reconfiguração, permitindo a adaptação da mesma a cada carga de trabalho e gerando flexibilidade no comportamento da cache. Para a avaliação e otimização de uma memória cache, muitos parâmetros ou critérios podem ser considerados, como a taxa de acerto, o tempo de acesso, a penalidade de falta, etc. Podemos concluir então que a otimização de uma memória cache envolve diversos objetivos, restrições e parâmetros que podem ser variados.

Dentre os objetivos de uma otimização de memória cache, podemos citar a maximização da taxa de acerto (ou minimização da taxa de erro), a minimização do tempo de acesso, a minimização da penalidade de falta, etc. Os principais parâmetros variáveis são número de comparadores disponíveis, associatividade, tamanho do bloco e número de *slots*, considerando que mudanças nesses parâmetros podem modificar o tipo da cache. Os parâmetros não estão restritos a estes, mas a maioria dos trabalhos encontrados no estado da arte utiliza estes parâmetros. Algumas das restrições nos processos de otimização de uma memória cache são: quantidade de memória disponível (número de blocos que podem ser armazenados), largura de banda e outros recursos limitados, incluindo os parâmetros citados anteriormente.

Além dos objetivos, restrições e parâmetros próprios de qualquer processo de otimização, ainda é necessário avaliar as situações em que é possível e/ou necessário realizar uma modificação na memória cache para que ela se adapte a uma carga de trabalho. Além

disso, é necessário escolher quais parâmetros serão variáveis no processo de otimização e quais serão considerados apenas como restrições.

O Quadro 1 apresenta alguns exemplos de objetivos, parâmetros e restrições que podem ser utilizados no projeto de uma memória cache reconfigurável. No projeto, um ou mais objetivos podem ser levados em consideração e, muitas vezes, a otimização de um dos objetivos afetará outros. Por exemplo, ao se minimizar o número de faltas em uma memória cache, maximiza-se o número de acertos desta também. Além disso, diversas características da memória cache e da hierarquia de memória podem ser adaptadas durante a execução de uma carga de trabalho em uma cache reconfigurável. Quando uma característica pode ser adaptada, dizemos que ela é um parâmetro variável da organização de memória cache reconfigurável. No entanto, se uma característica é mantida fixa durante toda a execução da carga de trabalho na memória cache, dizemos que ela é uma restrição no projeto desta cache.

Objetivos
Minimização do tempo de resposta da execução da carga de trabalho
Minimização do número de faltas de cache
Maximização do número de acertos de cache
Minimização do consumo de energia da cache
Minimização da penalidade por falta de cache
Minimização do tempo de acesso à memória
Minimização do tempo gasto com acessos à memória
Parâmetros e restrições
Tamanho do bloco (número de palavras do bloco)
Tamanho da cache (quantidade de memória disponível)
Organização da cache
Associatividade da cache
Política de substituição
Política de escrita
Número de *slots*
Largura de banda de comunicação entre a cache e a memória inferior
Número de níveis na hierarquia de memória

Quadro 1 - Exemplos de objetivos, parâmetros e restrições que podem ser utilizados no projeto de uma memória cache reconfigurável
Fonte: Elaborado pela autora

3.2.1 Definição da Arquitetura de Memória Cache Reconfigurável

Independentemente dos objetivos, restrições e parâmetros envolvidos no processo de escolha de configuração ideal para a adaptação e a posterior reconfiguração da memória

cache, podemos definir uma arquitetura de memória cache reconfigurável. Uma memória cache reconfigurável é uma cache que permite que sua estrutura seja modificada, várias vezes, depois que a cache é fabricada.

Nossa arquitetura de cache reconfigurável é composta por dois módulos principais: a organização reconfigurável e a política de adaptação. A organização reconfigurável é composta pelo sub módulo de armazenamento de blocos da memória principal e pela lógica necessária para que estes blocos sejam acessados. A política de adaptação é o módulo responsável por determinar a configuração mais adequada dada uma carga de trabalho e por realizar a reconfiguração da estrutura do módulo organização reconfigurável, modificando o comportamento da memória cache.

Em nossa arquitetura de cache reconfigurável, representada pelo diagrama da Figura 5, a política de adaptação, baseada em características de acesso à memória da carga de trabalho, escolhe uma das possíveis configurações e (re)configura o módulo Organização[2] Reconfigurável. Estas configurações não precisam existir fisicamente *a priori*, a política de adaptação pode determinar uma configuração através da escolha de valores para os parâmetros adaptáveis do módulo Organização Reconfigurável. Na Figura 5, as Configurações representam todas as combinações válidas dos valores destes parâmetros.

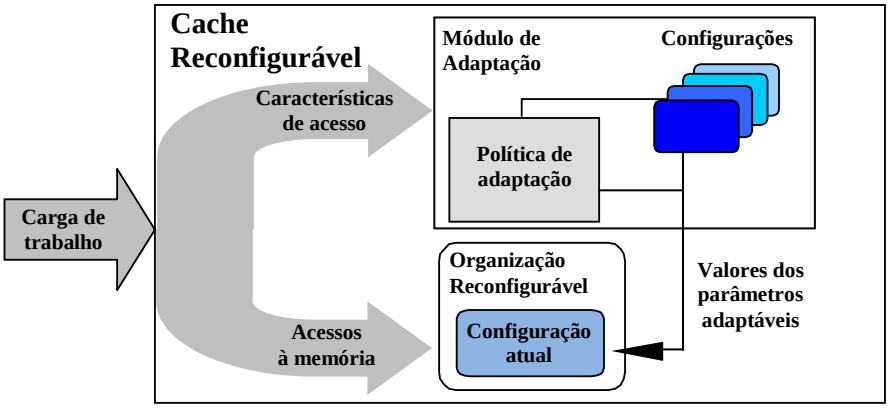

Figura 5 - Arquitetura da memória cache reconfigurável
Fonte: Elaborada pela autora

3.2.2 Definição da Política de Adaptação

A política de adaptação tem como parâmetros de entrada características da carga de trabalho do sistema e métricas de desempenho ou configurações da memória cache. Ela

[2] A palavra organização é usada aqui e no restante deste texto como uma descrição da estrutura de uma memória cache. Portanto, uma organização de memória cache reconfigurável possui parâmetros adaptáveis e restrições definidos.

escolhe a configuração da cache que determinará seu comportamento, como se estivesse escolhendo uma entre as diversas configurações possíveis considerando os parâmetros que podem ser adaptados na memória cache reconfigurável. Os parâmetros de entrada da política de adaptação podem ser indicados pelo usuário, estimados ou medidos com a execução da carga. Quanto mais confiáveis forem os parâmetros da política de adaptação, melhor será a configuração definida pela política.

Além dos parâmetros de entrada, a própria lógica da política de adaptação influencia na qualidade das configurações determinadas pela política. Uma lógica simples pode não ter um bom resultado, no entanto, uma lógica complexa pode inviabilizar a implementação da política ou a sua utilização durante o funcionamento da cache.

Como os parâmetros de entrada e a lógica da política de adaptação influenciam na qualidade de sua solução, uma das maneiras de se obter bons valores para os parâmetros sem sobrecarregar a política é realizar o monitoramento da execução de uma carga de trabalho ou parte dela. Este monitoramento é realizado durante um dado instante de tempo, chamado *quantum*. Este *quantum* deve ser pequeno o suficiente para não gerar muitas estatísticas, mas grande o suficiente para caracterizar o comportamento da carga de trabalho. Quando o monitoramento é finalizado, a política de adaptação pode ser executada e uma nova configuração é escolhida.

Se considerarmos que este monitoramento é realizado diversas vezes e conseqüentemente a reconfiguração da organização da cache também, então teremos uma cache reconfigurável dinâmica, que pode se reconfigurar durante a execução de uma carga de trabalho. No entanto, se isso não for possível, os parâmetros podem ser determinados uma única vez para uma determinada carga e esta terá apenas uma configuração de cache a ela associada. Neste caso, consideramos que temos uma cache reconfigurável estática, na qual sua configuração é escolhida, uma única vez, no início da execução da carga de trabalho.

Em sistemas computacionais multitarefa, diversos processos usam os recursos do sistema em espaços de tempo diferente. Como geralmente o escalonador do sistema operacional determina um tempo que o processo pode usar os recursos e depois ele é colocado em uma fila para voltar a usar, não é uma boa prática usar uma única configuração de cache para todos estes processos. Portanto, cada processo deve possuir uma configuração de cache associada, garantindo assim que a configuração seja adequada ao mesmo.

3.3 Proposta da Memória Cache Reconfigurável com Associatividade Variável Independente de *Slot*

Para analisarmos os possíveis benefícios da utilização de uma memória cache reconfigurável, foi necessário definir quais parâmetros, objetivos e restrições seriam utilizados em nossos testes. Estas escolhas refletem na complexidade de execução e no desempenho da memória cache reconfigurável. A seguir, são descritos os objetivos, parâmetros e restrições definidos no projeto da Memória Cache Reconfigurável com Associatividade Variável Independente de *Slot* e, no Quadro 2, estas características são sumarizadas. Uma característica mantida fixa pode ser uma restrição inicial de projeto ou uma opção de projeto. Neste caso, a opção de projeto torna-se uma restrição no processo de adaptação de uma determinada organização de cache reconfigurável. No entanto, uma restrição por opção de projeto pode ser modificada sem, no entanto, prejudicar a organização proposta. É necessário ressaltar ainda que esta é uma das possíveis propostas de memória cache reconfigurável baseada em nossa arquitetura de cache reconfigurável. Assim, alterando-se os objetivos levados em consideração e as características da memória cache que são consideradas parâmetros ou restrições, obtemos uma nova cache reconfigurável baseada em nossa proposta de arquitetura de cache reconfigurável.

Objetivos	**Considerado**
Minimização do tempo de resposta	Sim
Minimização do número de faltas de cache	Sim
Maximização do número de acertos de cache	Sim
Minimização do consumo de energia da cache	Não
Minimização da penalidade por falta de cache	Não
Minimização do tempo de acesso à memória	Não
Minimização do tempo gasto com acessos	Sim
Parâmetros e restrições	**Possibilidade de adaptação**
Tamanho do bloco (número de palavras no bloco)	Fixo (opção de projeto)
Tamanho da cache (quantidade de memória)	Fixo (restrição de projeto)
Organização da cache	Organização com associatividade variável
Associatividade da cache	Variável
Política de substituição	Fixa – LRU (opção de projeto)
Política de escrita	Fixa - *Write-back* (opção de projeto)
Número de *slots*	Fixo (opção de projeto)
Largura de banda de comunicação	Fixo (restrição de projeto)
Número de níveis na hierarquia de memória	Fixo (opção de projeto)

Quadro 2 - Objetivos, parâmetros e restrições considerados no projeto da Memória Cache Reconfigurável com Associatividade Variável Independente de *Slot*
Fonte: Elaborado pela autora

O tempo de resposta é hoje uma medida de desempenho muito utilizada. No entanto, a otimização do tempo de resposta pode ser considerada como a otimização de ao menos um dos outros possíveis objetivos a serem otimizados. Geralmente, estes objetivos são opostos, isto é, na maioria das vezes, uma melhora em um dos objetivos acarreta em uma piora de um ou vários outros. Portanto, como queríamos nos limitar a um objetivo, para facilitar a análise do impacto das modificações por nós sugeridas, escolhemos a minimização da taxa de falta na memória cache e a conseqüente maximização da taxa de acerto. Com esta otimização podemos conseguir a redução do tempo de execução dos programas através da redução do tempo total gasto com acessos às memórias inferiores.

Para que não houvesse modificações em outros objetivos, como o aumento do tempo de acesso, escolhemos parâmetros da memória cache reconfigurável que pudessem diminuir a taxa de faltas sem alterar tempos de acesso ou criar tempos de acessos diferenciados. Além disso, nossa proposta é baseada em conceitos já implementados atualmente em diversas áreas, como a adaptação e a reconfiguração. Como a adaptação é feita para cada carga de trabalho, alguns parâmetros não foram escolhidos para que o custo da memória cache não ficasse variável de acordo com cada solução, fazendo com que a otimização fosse feita antes da fabricação da memória cache e recaindo no problema dela não ser otimizada para todas as cargas de trabalho.

A solução para o problema de adaptação da memória cache mais utilizada atualmente é a modificação do tamanho do bloco armazenado. Diversos trabalhos, apresentados no capítulo 2, propõem que, de acordo com a localidade espacial da carga de trabalho, o tamanho do bloco lido da memória principal e gravado na cache seja variável para acompanhar as diferenças de localidade. Como já existem estes trabalhos que realizam a otimização da taxa de falta baseada na localidade espacial, neste trabalho, resolvemos descrever possíveis otimizações baseadas também na localidade temporal, para que as técnicas pudessem ser usadas em conjunto, melhorando o desempenho das aplicações ainda mais.

Dentre as características da memória cache que mantivemos constantes, estão o número de blocos possíveis de se armazenar na cache e o número de *slots*. Esta escolha é baseada no fato de que mudanças nesses dois parâmetros alterariam todo o projeto da memória cache, surgindo outros problemas. Como pretendemos realizar modificações na memória depois da sua fabricação ou até mesmo de maneira dinâmica, isto é, enquanto uma carga de trabalho é executada, tais parâmetros foram mantidos constantes.

Outra técnica muito utilizada para reduzir o tempo de acesso à memória é a utilização de diversos níveis de memória cache, seguindo a mesma teoria que levou a criação do que conhecemos como hierarquia de memória, definida na seção 2.1. No entanto, nosso objetivo é a otimização de uma memória cache e não de toda a hierarquia. Portanto, para avaliar as melhorias ocasionadas por nossa proposta, analisaremos a taxa de acerto. Em nosso contexto, podemos considerar que o aumento na taxa de acerto possivelmente acarreta em uma diminuição no tempo de execução da carga que estiver sendo executada. Então, em nossas análises, quanto maior for a taxa de acerto, melhor será o desempenho.

Já se sabe que um aumento na associatividade de uma cache, geralmente, melhora o desempenho da mesma, porque evita faltas por conflito (HENNESSY, 2003). Isso acontece porque, aumentando-se a associatividade, um bloco que poderia ser substituído e depois seria acessado novamente (causando um *cache miss*) poderia continuar na memória cache. No entanto, isso é totalmente válido somente se considerarmos um aumento no tamanho da cache (não redução do número de *slots*) para não criar outras situações conflitantes (estas situações são ilustradas na próxima seção).

Portanto, nossa proposta é modificar a associatividade da cache para diminuir o número de faltas por conflito sem, no entanto, aumentar o tamanho da cache (ou reduzir o número de *slots*) e sem gerar outras situações conflitantes.

Considerando caches associativas por conjunto com o mesmo número de blocos possíveis de se armazenar, temos as duas situações ilustradas na Figura 6. Na Figura 6(a) temos a representação de uma cache associativa por conjunto 2*way*. Já a Figura 6(b), representa uma cache associativa por conjunto 4*way*. É importante destacar que como o número de blocos é o mesmo nas duas organizações, o número de *slots* da segunda é a metade do número da primeira. Apesar do aumento da associatividade para reduzir as faltas por conflito, o compartilhamento de *slots* por blocos que antes ficavam em *slots* separados (Figura 6(b)) pode até gerar mais conflitos que a organização com menor associatividade (Figura 6(a)). Para melhor entendimento dessas situações e como elas poderiam ser evitadas, exemplificamos, na próxima seção, a execução de uma carga de trabalho em três organizações de memórias cache.

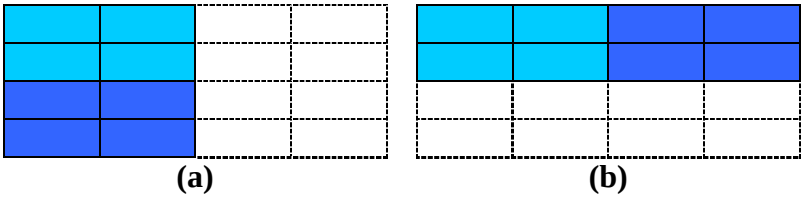

Figura 6 - Caches com o mesmo número de blocos e associatividades diferentes.
Fonte: Elaborada pela autora

3.3.1 Situação Problema Exemplo

Nesta seção apresentamos três situações que exemplificam a utilização de duas organizações de cache tradicionais e nossa proposta. Pretendemos por meio de exemplos mostrar as possíveis vantagens da adaptação da associatividade de cada *slot* da cache a cada carga de trabalho executada.

Nos exemplos a seguir, consideramos somente os acessos aos *slots* 0, 4, 5 e 6 (quando existirem) que são os que usamos em nossas análises. Nossas caches possuem dimensões reduzidas, para facilitar a exemplificação dos comportamentos do *trace* nas caches. O *trace* executado nas caches está representado na Figura 7. O *trace* apresenta os endereços dos blocos acessados, independente do tipo de acesso (leitura ou escrita). Não usamos endereços de palavras para facilitar a visualização dos blocos armazenados em cada entrada da cache.

Ordem	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**l**	**m**	**n**
Endereço do bloco	0	4	8	12	16	24	5	6	0	4	8	16	24

Figura 7 - Trace composto pelos endereços dos blocos acessados (leitura ou escrita).
Fonte: Elaborada pela autora

A primeira cache exemplo é uma cache 2*way* com 8 *slots*, apresentada na Figura 8. Cada entrada só pode armazenar um bloco, no entanto apresentamos todos os blocos já armazenados em cada entrada para facilitar a análise baseada no histórico de acessos. Cada tipo de acesso (falta ou acerto) pode ser visualizado no Quadro 3 (página 46). As faltas compulsórias só podem ser reduzidas se mudarmos o tamanho dos blocos. Como já existem diversos trabalhos já apresentados no Capítulo 2 (Revisão da Literatura), não nos preocuparemos em reduzir este tipo de falta. Nosso objetivo é reduzir as faltas por conflito.

Número do Slot	**Entrada 1**	**Entrada 2**	**Possíveis blocos**
0	0 16 0 16	8 24 8 24	0,8,16,24...
1			1,9,17,25...
2			2,10,18,26...
3			3,11,19,27...
4	4	12	4,12,20,28...
5	5		5,13,21,29...
6	6		6,14,22,30...
7			7,15,23,31...

Figura 8 - Cache 2*way* com 8 *slots* com os acessos definidos na Figura 7.
Fonte: Elaborada pela autora

Para reduzir o número de faltas por conflito, a técnica mais utilizada é aumentar a associatividade da cache. Esta técnica é baseada no conceito de que, se aumentarmos o

número de entradas de um *slot*, este poderá armazenar mais blocos conflitantes, diminuindo o número de faltas por conflito. No entanto, para aumentar a associatividade e não diminuir o número de *slots*, será necessário aumentar o tamanho da memória cache, isto é, aumentar o número de blocos armazenados na memória cache para manter o número de *slots*. Isto implica em um custo maior da cache.

Como nossa proposta visa não aumentar o custo da cache com o aumento da memória, vamos usar como exemplo uma cache 4*way* com 4 *slots*, dobrando a associatividade e reduzindo pela metade o número de *slots*, mantendo o mesmo número de blocos armazenados do exemplo anterior. Dessa maneira, a cache seria reduzida à linha tracejada da Figura 8, mas com 4 entradas para cada *slot*. Esta nova cache está apresentada na Figura 9.

Número do *Slot*	**Entrada 1**	**Entrada 2**	**Entrada 3**	**Entrada 4**	**Possíveis blocos**
0	0 16 8	4 24 16	8 0 24	12 4	0,4,8,12,16,20,24, 28...
1	5				1,5,9,13,17,21,25, 29...
2	6				2,6,10,14,18,22,26, 30...
3					3,7,11,15,19,23,27, 31...

Figura 9 - Cache 4*way* com 4 *slots* com os acessos definidos na Figura 7.
Fonte: Elaborada pela autora

Na cache 4*way* com 4 *slots*, apesar de termos dobrado a associatividade, outros blocos também passaram a disputar o mesmo *slot* (*slot* 0 e 4 da cache da Figura 8 concorrem pelo mesmo *slot* da Figura 9, por exemplo). Como os *slots* 0 e 4 da Figura 8 eram muito acessados, no *slot* 0 da Figura 9 ocorreram diversas faltas por conflito. Este é um caso extremo, pois a disputa ainda prejudicou o antigo *slot* 4, diminuindo a taxa de acerto. No entanto, não podemos contar com a "sorte" de que blocos de *slots* muito disputados em uma cache com o dobro de *slots* (Figura 8) não disputem *slots* em comum em uma cache com o dobro da associatividade (Figura 9).

Nossa proposta de solução funciona como uma opção intermediária à dos dois exemplos anteriores. Para reduzirmos o número de faltas por conflito, aumentaremos a associatividade de *slots* com conflitos, no entanto, não vamos reduzir o número de *slots*, para que blocos que antes não disputassem um *slot* passem a disputar. Nossa proposta possui a mesma restrição do segundo exemplo: o número de blocos que podem ser armazenados na cache é o mesmo nos três exemplos. Por causa dessas restrições, nossa proposta de cache reconfigurável possui uma associatividade diferente entre os *slots*, no entanto, possui o mesmo tempo de acesso em todas as entradas. A cache configurada para o problema descrito nos exemplos anteriores usando nossa proposta está representada na Figura 10.

Neste exemplo, *slots* pouco acessados, como os *slots* 5 e 6 “doam” entradas para o *slot* 0, onde estavam acontecendo muitos conflitos. Para os *slots* pouco acessados, a redução da associatividade não faz diferença ou o ganho reduzindo o número de faltas por conflito em outros *slots* é maior que uma possível perda com a “doação”. As faltas por conflitos são reduzidas a 0 neste exemplo simplificado. Não esperamos acabar com todas as faltas por conflito em caches reais com cargas de trabalho representativas, no entanto, pretendemos reduzir o número de faltas por conflito através da “doação” de entradas.

Número do *Slot*	**Entrada 1**	**Entrada 2**	**Entrada 3**	**Entrada 4**	**Possíveis blocos**
0	0	8	16	24	0,8,16,24...
1					1,9,17,25...
2					2,10,18,26...
3					3,11,19,27...
4	4	12			4,12,20,28...
5	5				5,13,21,29...
6	6				6,14,22,30...
7					7,15,23,31...

Figura 10 - Cache reconfigurável 8 *slots* com os acessos definidos na Figura 7.
Fonte: Elaborada pela autora

O Quadro 3 sumariza o tipo de acesso à memória cache (falta ou acerto) para cada acesso da execução do *trace* da Figura 7 nas memórias cache exemplificadas (Figuras 8, 9 e 10). Além disso, cada falta é classificada como compulsória ou por conflito. Nenhuma falta foi classificada como por capacidade, pois nestes exemplos não ocorreu este tipo de falta.

Ordem	**Endereço**	**cache 2way com 8 slots**	**cache 4way com 4 slots**	**cache reconfigurável com 4 slots**
A	0	falta compulsória	falta compulsória	falta compulsória
B	4	falta compulsória	falta compulsória	falta compulsória
C	8	falta compulsória	falta compulsória	falta compulsória
D	12	falta compulsória	falta compulsória	falta compulsória
E	16	falta compulsória	falta compulsória	falta compulsória
F	24	falta compulsória	falta compulsória	falta compulsória
G	5	falta compulsória	falta compulsória	falta compulsória
H	6	falta compulsória	falta compulsória	falta compulsória
I	0	falta por conflito	falta por conflito	acerto
J	4	acerto	falta por conflito	acerto
L	8	falta por conflito	falta por conflito	acerto
M	16	falta por conflito	falta por conflito	acerto
N	24	falta por conflito	falta por conflito	acerto

Quadro 3 - Tipo de acesso para o *trace* da Figura 7 e as caches das Figuras 8, 9 e 10.
Fonte: Elaborado pela autora

3.3.2 Definição da Organização de Memória Cache Reconfigurável com Associatividade Variável Independente de Slot

A organização de memória cache reconfigurável aqui proposta funciona como uma cache com organização 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 o funcionamento 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 máxima (número de comparadores disponíveis). Como a associatividade varia com um valor mínimo igual a um, nenhum *slot* é criado ou removido da memória cache. Portanto, apesar dessas modificações, nesta arquitetura reconfigurável, o tamanho da cache é sempre o tamanho determinado no momento de projeto da mesma.

Denominamos uma cache reconfigurável A_i-A_f*way* uma cache que possui uma arquitetura reconfigurável por nós proposta, com associatividade inicial igual a A_i e associatividade máxima igual a A_f. Uma cache reconfigurável 2-4*way* é representada na Figura 11. Esta representação pode ser comparada com as duas representações já apresentadas na Figura 6, tendo o mesmo número de blocos que elas. Na Figura 11, os blocos representados com traços verticais indicam os blocos fixos, isto é, os que não podem ser alocados para outros *slots* para que a cache continue com o mesmo número de *slots*. Os blocos com traços horizontais são os que são realocados pela política de adaptação. Nesta figura, o primeiro *slot* recebeu dois novos blocos, possuindo o tamanho máximo (4 blocos), enquanto os segundo e quarto *slots* doaram um bloco. O terceiro *slot* não sofreu alterações em sua associatividade.

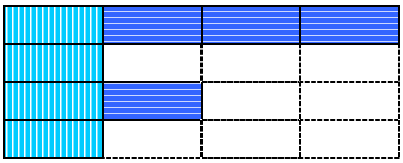

Figura 11 - Cache reconfigurável 2-4way
Fonte: Elaborada pela autora

3.3.3 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 memória cache associativa por conjunto e, conseqüentemente, pela nossa arquitetura também, estão representadas na Figura 12. Os bits usados no campo Deslocamento no Bloco indicam o deslocamento da palavra procurada no bloco em que ela se encontra. 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 um bloco é procurado na cache.

Tag do Bloco	Índice do *Slot*	Deslocamento no Bloco

Figura 12 - Porções do endereço visto por uma cache com organização associativa por conjunto ou reconfigurável com a organização proposta
Fonte: Elaborada pela autora

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

$$Número\ de\ Blocos = \frac{Tamanho\ da\ Cache}{Tamanho\ do\ Bloco} \quad \textbf{(3.1)}$$

O número de *slots* de uma cache associativa por conjunto e, conseqüentemente, de nossa arquitetura também, é dado pela equação (3.2). Em uma cache associativa por conjunto, as associatividades inicial e final são as mesmas, isto é, o número de comparadores da memória cache.

$$Número\ de\ Slots = \frac{Número\ de\ Blocos}{Associatividade\ Inicial} \quad \textbf{(3.2)}$$

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

$$Bits\ do\ índice\ de\ slots = \lceil \log_2 Número\ de\ Slots \rceil \quad \textbf{(3.3)}$$

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, mantendo-se o número de *slots* constante, como acontece nas organizações clássicas e nesta organização reconfigurável com associatividade variável independente de *slot*. Espera-se que o desempenho da cache com uma organização como a apresentada nesta dissertação

seja similar ou superior a uma cache com organização associativa por conjunto com o mesmo número de comparadores.

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. Em nossa memória cache com associatividade variável independente de *slot*, o número de comparadores indica a associatividade máxima da cache. 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 que na reconfigurável com associatividade variável. Isto acontece porque a cache associativa por conjunto possui um número menor de *slots* (possui mais blocos em um mesmo *slot*) que uma cache reconfigurável com associatividade variável com o mesmo número de comparadores, com isso, o número de blocos que podem ser armazenados em um mesmo *slot* é maior, então são necessários mais bits para diferenciá-los, considerando-se memórias cache com a mesma capacidade de armazenamento.

3.4 Política de Adaptação

Para realizar os experimentos com cargas de trabalho reais, projetamos uma política de adaptação simples, baseada no senso comum de que um *slot* com um número de faltas grande necessita de uma maior associatividade para reduzir este número. A qualidade dos resultados está diretamente relacionada com a implementação da política de adaptação.

Em nossa política implementada, consideramos dois parâmetros de entrada, obtidos com a execução da carga de trabalho em uma organização de cache: número de acessos por *slot* e número de faltas por *slot*. O número de acessos indica o nível de importância (utilização) do *slot*, isto é, uma mudança na associatividade dele altera o desempenho da cache ou se o desempenho não é muito influenciado por ele. O número de faltas por *slots* indica se a associatividade do mesmo é suficiente para suportar a demanda do mesmo.

Nesta implementação de política, uma tabela estatística é preenchida a cada *quantum* da política. Esta tabela contém as métricas de desempenho usadas como parâmetros da política de adaptação. Além destas, outras duas métricas são obtidas ao final do *quantum* em função das duas anteriores: a média de acessos por *slot* e a média de faltas por *slot*.

Baseado nas métricas citadas anteriormente, é possível caracterizar e classificar cada *slot* de acordo com as regras descritas a seguir. O número de acessos é considerado grande se

é maior ou igual à média do número de acesso de todos os *slots*. De maneira análoga, o número de faltas é considerado grande se é maior ou igual à média do número de faltas de todos os *slots*. Se o número é menor que a média, então ele é considerado pequeno. Um *slot* GG (grande-grande) tem um número grande de faltas e de acessos, então podemos considerá-lo um *slot* muito acessado e com um número insuficiente de blocos disponíveis, isto é, sua associatividade não é grande o suficiente para suportar a demanda. Apesar de ter um número pequeno de acessos, um slot GP (grande-pequeno) possui um número insuficiente de blocos e pode ser melhorado. Já um slot PG (pequeno-grande) tem uma condição ideal, isto é, ele é muito acessado, mas mantém uma pequena taxa de falta. Um *slot* PP (pequeno-pequeno) possui um pequeno número de faltas e acessos, portanto, ele pode ceder blocos para *slots* mais importantes (com maior número de acessos) ou *slots* com carência de blocos (com número grande de faltas). Estas combinações estão resumidas no Quadro 4.

Combinação (Classe)	Número de faltas	Número de acessos	Descrição
GG	Grande	Grande	Possui maior prioridade para receber um bloco.
GP	Grande	Pequeno	Possui menor prioridade para receber um bloco.
PG	Pequeno	Grande	Condição ideal: não recebe nem doa blocos.
PP	Pequeno	Pequeno	Deve doar blocos para *slots* GG ou GP.

Quadro 4 - Combinações possíveis para a classificação dos parâmetros usados pela política de adaptação
Fonte: Elaborado pela autora

Nossa política de adaptação retira blocos de *slots* PP (doadores) e adiciona blocos nos *slots* GG e GP (receptores). Apesar de tanto os *slots* GG quanto os GP poderem/precisarem receber blocos, os *slots* GG têm uma prioridade maior, porque uma taxa de faltas elevada em um *slot* muito acessado gera um impacto maior no desempenho que um *slot* menos acessado.

Como exemplo, suponha uma memória cache com 8 *slots*, 16 blocos disponíveis e associatividade máxima igual a 4 e inicial igual a 2 (memória cache reconfigurável *2-4way*). Inicialmente, a política de adaptação classifica os *slots* de acordo com as regras da política. A tabela estatística da política de adaptação para um determinado *quantum* é apresentada no Quadro 5. Depois que os *slots* são classificados, são criadas duas listas, uma de *slots* doadores e outra de receptores. Na lista de receptores os *slots* GG ficam no início e são seguidos pelos *slots* GP. Neste exemplo, a lista de doadores é formada pelos *slots* S1, S3 e S4 e a lista de receptores é formada pelos *slots* S0, S2, S7 e S5. A ordem dos *slots* de um mesmo grupo é determinada pela ordem em que os mesmos são classificados, portanto, neste nosso exemplo, vai do menor índice para o maior. Então as listas são cruzadas e o primeiro receptor recebe um bloco do primeiro doador. Depois disso, ambos são removidos de suas listas. Este

processo é repetido até que uma das listas esteja vazia. Desta maneira, cada vez que um *quantum* termina, os receptores que forem atendidos terão sua associatividade aumentada em um e os doadores usados terão a associatividade reduzida em um.

Slot	**Número de faltas**	**Número de acessos**	**Classificação**
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	

Quadro 5 - Tabela Estatística ao fim de um *quantum*
Fonte: Elaborado pela autora

Considerando uma configuração inicial de dois blocos por *slot* (cache reconfigurável *2-4way*) e a execução de um *trace*, a Figura 13(a) representa a utilização da cache no final de um *quantum*. Consideramos ainda que, através do monitoramento do *trace* no *quantum* anterior, foi gerada a tabela estatística apresentada no Quadro 5 com a classificação e listas de doadores e receptores descritas anteriormente. Este é um exemplo simplificado, com uma cache de tamanho reduzido e uma carga de trabalho simples, usado somente para exemplificar o mecanismo de alocação de blocos.

Slot	***Tag***	***Tag***
0	8	24
1	1	-
2	10	51
3	3	-
4	12	53
5	5	-
6	13	40
7	7	-

(a)

Slot	***Tag***	***Tag***	***Tag***
0	0	8	24
1	1		
2	2	10	51
3	3		
4	4	12	53
5	5		
6	6	13	40
7	7		

(b)

Figura 13 - Representação de uma cache antes (a) e depois (b) da reconfiguração
Fonte: Elaborada pela autora

Os blocos não usados pertencem a *slots* PP, com um pequeno número de acessos e faltas. Os outros *slots* são GG ou GP e podem reduzir a taxa de falta recebendo outros blocos. Baseado nesta classificação, a cache é reconfigurada e terá uma configuração como a representada na Figura 13(b). *Slots* PP tiveram a associatividade reduzida e os *slots* GG e GP tiveram a associatividade aumentada, sendo capazes de armazenar mais blocos e,

possivelmente, reduzindo a taxa de faltas. Neste exemplo, um acesso ao bloco de *tag* 0, antes dos acessos aos blocos de *tag* 8 e 24, em uma cache com comportamento fixo (*2way*) causaria uma falta. No entanto, na cache reconfigurável, como o *slot* que armazena estes blocos foi classificado como GG e, portanto, recebeu um bloco, não haveria conflito e o bloco seria encontrado na cache gerando um acerto.

4 RESULTADOS DA VERIFICAÇÃO DA ARQUITETURA PROPOSTA

Neste capítulo, apresentamos os métodos e materiais usados para analisar a proposta da Arquitetura de Cache reconfigurável. Além disso, apresentamos as ferramentas desenvolvidas para a verificação de nossa proposta, as simulações realizadas, as configurações utilizadas, os resultados das simulações e a análise dos mesmos. Ao final do capítulo, apresentamos uma síntese dos resultados da verificação da arquitetura proposta.

4.1 Métodos e Materiais

A implementação física em hardware e a verificação de uma memória cache envolvem custos elevados. Além disso, por se tratar de uma cache que possui seu comportamento adaptado/reconfigurado no decorrer da execução das cargas de trabalho, uma cache real (fixa) não poderia ser usada para verificar e analisar as idéias propostas nesse trabalho. Por tudo isso, escolhemos a simulação, devido ao baixo custo e a possibilidade de verificação e análise de qualquer configuração de memória cache. O tipo de simulação utilizada é a dirigida por rastro (*trace-driven simulation*), onde são identificados os acessos à memória ocorridos durante a execução de uma carga de trabalho. Estes acessos são conhecidos como rastro ou *trace*.

Nosso modelo de cache para simulação utiliza acessos realizados à memória descritos através de um *trace*. A memória cache que executa esta carga de trabalho é modelada através de estruturas de dados que permitam que sejam realizadas análises dos blocos armazenados na memória cache, da taxa de falta de cache durante a execução da carga de trabalho e do tempo de execução total da mesma.

Para a verificação da simulação da cache, baseado em (LAW, 1991) (UHLIG, 1997), é possível fazer um experimento com *trace* e configuração escalados (reduzidos de maneira que sejam compatíveis com um experimento com *trace* e configurações reais), isto é, execução de casos simplificados (reduzidos). A verificação é realizada através da comparação do experimento realizado usando o simulador com a execução manual do algoritmo que descreve o comportamento de uma memória cache com as mesmas cargas de trabalho e configurações de memória cache.

Os resultados obtidos através de *traces* sintéticos e configurações de caches escaladas nem sempre são a opção ideal para comprovar a qualidade de uma solução proposta. Portanto, para analisar o desempenho de nossa proposta, realizamos experimentos com cargas de trabalho reais.

Os materiais utilizados nessa pesquisa foram divididos em estruturas de dados, aplicativos e ferramentas desenvolvidas e de desenvolvimento e equipamentos:

a) Estruturas de dados:
   i) *Traces* obtidos do "*BYU Trace Distribution Center*" (BYU, 2001): *traces* recolhidos de execuções em máquinas reais através de execução de *benchmarks*.
b) Aplicativos e ferramentas desenvolvidas:
   i) Simulador de memória cache tradicional e reconfigurável (CacheSim): desenvolvido utilizando-se a linguagem Java;
   ii) Codificador/decodificador de *trace*: decodifica os *traces* obtidos do "*Brigham Young University* (*BYU) Trace Distribution Center*" e os converte para serem lidos pelo simulador de memória cache desenvolvido.
c) Aplicativos e ferramentas de desenvolvimento:
   i) Compilador *Borland C++ Builder* versão 4.0;
   ii) Compilador JDK (*Java Development Kit*) 1.4.1.
d) Equipamentos:
   i) Computadores pessoais AMD Athlon e/ou Intel Pentium 4.

4.2 Simulador de Memória Cache (CacheSim)

O CacheSim é um simulador desenvolvido a partir do módulo de hierarquia de memória de um simulador de sistemas computacionais paralelos, o ClusterSim (GÓES, 2004c), desenvolvido pelo nosso grupo de pesquisa. O ClusterSim, por sua vez, é uma evolução do RJSSim (GÓES, 2003), uma ferramenta de auxílio ao aprendizado de processamento paralelo. O CacheSim foi desenvolvido em Java e permite a utilização do módulo de hierarquia de memória do ClusterSim, por nós desenvolvido, sem a necessidade de configuração e simulação das partes do sistema computacional paralelo que não são necessárias para as análises realizadas neste trabalho.

O CacheSim usa simulação dirigida por fluxo (*trace-driven simulation*) (UHLIG, 1997). Através desta simulação, é possível determinar o número de faltas, número de acessos à memória e o tempo gasto em uma execução real em uma hierarquia de memória especificada pelo usuário. Além disso, o simulador ainda calcula a taxa de faltas da cache, uma métrica de desempenho. No CacheSim é possível simular uma máquina monoprocessada com um nível de memória cache.

O CacheSim simula somente instruções de acesso à memória e, para isso, recebe como entrada um arquivo de *trace* contendo o tipo de acesso realizado (leitura ou escrita) e o endereço onde o mesmo ocorre. O usuário, além de fornecer o arquivo com o *trace* a ser simulado, deve configurar a hierarquia de memória. Para isso, ele deve especificar a latência, os tempos de leitura e escrita e o tamanho da cache e da memória principal, além do tamanho da palavra e do tamanho do bloco.

O CacheSim permite ao usuário escolher a organização da cache simulada: mapeamento direto, completamente associativa, associativa por conjunto e adaptativa. A organização adaptativa simula nossa proposta de memória cache reconfigurável com associatividade variável independente de *slot* e usa a política de adaptação apresentada na seção 3.4. Quando a organização associativa por conjunto ou adaptativa é escolhida, é necessário especificar também a associatividade da cache. Na organização adaptativa, esse valor indica a associatividade inicial da cache, então, o usuário deve especificar também um valor limite de associatividade, isto é, a associatividade máxima de um *slot* (número de comparadores da cache).

Na versão atual do simulador, foram implementadas a política de substituição LRU (*Least Recently Used* - Usado Menos Recentemente) e a estratégia de escrita conhecida como *write-back*. No entanto, como sua implementação foi realizada de maneira modularizada, novas políticas podem ser facilmente inseridas.

4.2.1 Verificação do Simulador CacheSim

Para verificar o CacheSim, montamos vários *traces* e várias configurações de caches simples. Como exemplo, é apresentado a seguir a execução de um *trace* em uma das caches. O *trace* é composto por 10 instruções e a memória cache possui os seguintes parâmetros: organização associativa por conjunto *2way*, 4 *slots* de cache e blocos de 16 palavras. Na Figura 14 podemos verificar a execução do *trace* que teve uma taxa de acerto de 40%.

Endereço	Bloco	*Slot*	Acesso à cache
1000	62	2	*miss*
199	12	0	*miss*
15	0	0	*miss*
1012	63	3	*miss*
203	12	0	*hit*
3240	202	2	*miss*
1290	80	0	*miss*
1007	62	2	*hit*
207	12	0	*hit*
1295	80	0	*hit*

Figura 14 - Execução do *trace* para verificação do simulador CacheSim.
Fonte: Elaborada pela autora

No simulador, obtivemos os mesmos resultados representados na Figura 14. Com isso, verificamos o módulo de hierarquia de memória para esta configuração. O simulador ainda exibe o tempo de resposta da execução desse *trace* em um computador real (com um nível de cache). A configuração do computador é exibida no Quadro 6. Esses valores foram obtidos através de alguns *benchmarks* (*Sandra* 2003, *Cachemem* 2.6 e *Cache Burst* 32). Estes valores são usados em todos os experimentos apresentados neste capítulo.

O tempo de execução deste *trace*, nesta configuração de computador, foi de 2,17032ms. Baseado nestes resultados, verificamos o funcionamento do módulo de hierarquia de memória comparando-se com a execução algorítmica do mesmo.

Processador	Athlon XP 2000	Tamanho da palavra	32 bits
Velocidade do processador	6211 MIPS	Tamanho do bloco	4 palavras
Latência da memória principal	139ns	Tempo de acesso à cache	2ns
Tempo de leitura da memória principal	3.42ns	Tempo de leitura da memória cache	0.28ns
Tempo de escrita da memória principal	5.81ns	Tempo de escrita da memória cache	0.32ns
Tamanho da memória cache (instruções e dados)	64 KB		

Quadro 6 - Especificação do computador simulado.
Fonte: Elaborado pela autora

4.3 Conversor de *Trace*

Cada registro dos *traces* do *BYU Trace Distribution Center* contém seis campos: endereço, tipo de requisição, bytes transferidos e atributo (não utilizados em *traces* com

execução sem cache) e o tempo entre o acesso anterior e o atual (BYU, 2001). Em nossos experimentos no CacheSim, usamos apenas o endereço de acesso e o tipo de requisição. Não usamos o tempo entre os acessos porque o mesmo é dependente da máquina utilizada na coleta das referências.

O conversor de *trace* lê um arquivo com o formato de *trace* do *BYU Trace Distribution Center* e para cada registro lido, o conversor escreve em um outro arquivo o tipo de requisição e o endereço da mesma. No entanto, os registros do *trace* não são requisições de leitura ou escrita à memória somente. Como não estamos interessados em outras requisições, somente são escritos no arquivo os registros que representam leitura ou escrita na memória principal. Este arquivo pode ser lido pelo nosso simulador, o CacheSim, e a execução do *trace* pode ser simulada.

4.4 Experimentos

Para verificar e avaliar nossa proposta, tentamos aproximar ao máximo as especificações das caches e *traces* às usadas comercialmente. Por isso, utilizamos uma configuração de um computador comercial que possuímos. Esta configuração está apresentada no Quadro 6. Os valores foram obtidos através da medição com a execução de *benchmarks*. Portanto, os tamanhos, latências e tempo de acesso das memórias principal e cache usados em nossos experimentos são baseados em um computador real. Além disso, o tamanho da palavra e do bloco da cache também são iguais aos deste computador real. A política de substituição de blocos da memória cache usada é a LRU (*Least Recently Used* – Usado Menos Recentemente).

As memórias cache simuladas são memórias normalmente encontradas em sistemas computacionais tradicionais. Para determinar o tamanho e a associatividade das caches simuladas, consideramos o tamanho da memória cache na qual fizemos a medição, isto é, o número de blocos possíveis de se armazenar na cache. Desta maneira, segundo a Equação 4.1, a cache simulada pode armazenar 4096 blocos.

$$N^o de\ Blocos = \frac{Tamanho\ da\ Memória}{N^o\ de\ palavras\ no\ bloco\ \times Tamanho\ da\ Palavra} = \frac{64\ KB}{4 \times 32b} = 4096 \qquad \textbf{(4.1)}$$

As memórias cache associativas por conjunto simuladas têm o número de *slots* definido pela Equação 4.2. Através dela podemos concluir que a cache associativa por

conjunto 2way possui 2048 *slots*, enquanto a cache associativa por conjunto *4way* possui 1024 *slots*.

$$N^o de\ Slots = \frac{N^o\ de\ Blocos}{Associatividade} \quad \textbf{(4.2)}$$

Além destas duas configurações tradicionais de cache, ainda simulamos mais uma organização, não muito usual com este número de blocos, mas que serve para nós como referência. Simulamos uma cache completamente associativa (CA) com 4096 *slots*. Esta cache foi utilizada para medirmos o melhor desempenho de uma memória cache possível com a capacidade por nós determinada (LENNERSTAD, 2000). Desta maneira, foi possível saber, para cada *trace* simulado, qual seria a taxa de acerto máxima que teríamos, apenas modificando a associatividade da memória cache.

Além dessas três configurações de cache já citadas, ainda simulamos uma quarta configuração que chamamos de *Double4way*. Esta configuração é uma cache associativa por conjunto *4way* com o mesmo número de *slots* (2048) da cache associativa por conjunto *2way* anteriormente descrita e, conseqüentemente, o dobro de *slots* da cache associativa por conjunto *4way* anteriormente descrita. Portanto, a configuração *Double4way* pode armazenar 8192 blocos, o dobro das outras caches simuladas. Esta configuração foi simulada para analisarmos o impacto do aumento da associatividade na cache *2way* sem, no entanto, a ocorrência de conflitos em *slots* gerados pelo agrupamento na cache *4way* de dois *slots* da *2way*.

Para a avaliação de nossa arquitetura de cache reconfigurável, simulamos duas memórias cache reconfiguráveis com associatividade variável independente de *slot*: *2-4way* e *2-8way*. Como as duas configurações possuem associatividade inicial igual a 2, o número de *slots* das duas caches é igual ao número de *slots* da cache 2*way* (2048). Além disso, o número de blocos possíveis de se armazenar é igual para as caches 2*way*, 4*way*, 2-4*way* e 2-8*way* (4096). O que diferencia as caches 2-4*way* e a 2-8*way* é a associatividade máxima ou número de comparadores disponíveis. Como a cache 2-8*way* possui mais comparadores, o custo da mesma será maior que o da 2-4way. No entanto, usando esta configuração em nossos experimentos, desejamos analisar o impacto do aumento da associatividade em *slots* com alta taxa de falta, e verificar se existem *slots* que são pouco usados e podem “doar” blocos. Desejamos verificar se esse aumento na associatividade máxima poderia gerar um desempenho equivalente ao da cache associativa por conjunto *Double4way*.

Para uma melhor visualização das memórias cache simuladas, fizemos um quadro comparativo apresentado no Quadro 7. Nela, *N* é o menor número de *slots* usados em nossas

simulações (1024). Podemos observar que todas as caches possuem o mesmo tamanho, com exceção da configuração *Double4way* por motivos já explicados. O primeiro valor da equação que define o tamanho das memórias cache indica a associatividade inicial delas, enquanto o segundo indica o número de *slots*. Essa notação usa a idéia descrita no Capítulo 2 (Revisão da Literatura) e representada na Figura 3(a), onde uma cache completamente associativa pode ser representada como uma cache associativa por conjunto com um único *slot*.

Configurações de cache	**Tamanho**
2*way*	2 x 2*N*
2-4*way*	2 x 2*N*
2-8*way*	2 x 2*N*
4*way*	4 x *N*
Completamente associativa	4*N* x 1
Double4way	4 x 2*N*

Quadro 7 - Configurações de caches simuladas
Fonte: Elaborado pela autora

Usamos *traces* reais obtidos do BYU *Trace Distribution Center* (BYU, 2001). Estes *traces* correspondem à execução de alguns *benchmarks* SPEC (*Standard Performance Evaluation Corporation*) (HENNING, 2000) executados utilizando-se o Windows 2000. Os *traces* disponíveis no BYU *Trace Distribution Center* possuem, além dos acessos à memória gerados pelo próprio *benchmark*, os acessos também do sistema operacional (SO). Nossa opção por usar este tipo de *trace* está baseada no fato de que nenhum *benchmark* é executado sem a interferência de um sistema operacional, portanto, a existência dos acessos do SO faria com que o experimento ficasse mais real.

Para os experimentos, usamos *traces* de duas classes diferentes do SPEC CPU2000 (HENNING, 2000), conhecidos como SPEC CINT (*SPEC CPU Integer*) e SPEC CFP (*SPEC CPU Float Point*). Os resultados destes experimentos são apresentados a seguir.

O SPEC CPU2000 é um *benchmark* usado para avaliar, principalmente, os módulos que processam valores inteiros (SPEC INT) e de ponto flutuante (SPEC FP). No entanto, eles também são usados para a avaliação de módulos de memória, já que, para a realização de cálculos com estes valores, muitos deles são lidos e/ou escritos na memória (HENNING, 2000).

Como simulamos *traces* de um sistema operacional multitarefa (*Windows* 2000), e este tipo de sistema operacional é comumente utilizados nas máquinas, tivemos que considerar o escalonamento de processos realizado pelo sistema operacional. Por isso, a cada 10 ms do tempo de simulação, interrompíamos a simulação e invalidávamos o conteúdo da memória

cache. Para reduzir o possível *overhead* imposto pela reconfiguração da memória cache, consideramos que o *quantum* da política de adaptação é igual a 10 ms também.

4.4.2 SPEC CINT

Os *traces* do SPEC CINT usados em nossos experimentos são apresentados no Quadro 8. Este quadro contém a categoria da carga de trabalho e o número de acessos à memória coletados para a simulação. O número de acessos é variável porque no BYU *Trace Distribution Center* são disponibilizados *traces* com cem milhões de referências, no entanto, para nossa simulação, filtramos o *trace* através do conversor de *trace*, para que somente os acessos à memória de instruções e dados fossem simulados. Como este número de acesso é variável de carga de trabalho para carga de trabalho, o número de acessos simulados também é variável.

Nos gráficos das Figuras 15, 17 e 18 são apresentadas as taxas de acerto dos *traces* do SPEC CINT usados em nossos experimentos. Os valores das métricas obtidos com a simulação destes *traces* são apresentados no Apêndice I. Ao analisarem-se os gráficos é importante lembrar que quanto maior é a taxa de acerto, melhor será o desempenho da memória cache.

Nome	Categoria	Número de Acessos
crafty	Jogo de xadrez	10.481.991
Gap	Interpretador de teoria de grupos	10.481.868
Gcc	Compilador	10.481.937
gzip_graphic	Compressão	10.481.993
Mcf	Otimização combinatória	10.481.962
parser	Processamento de texto	10.482.049
perl_diffmail	Linguagem de programação PERL	10.481.727
twolf	Pacote de *placement and routing*	10.481.022
vpr_place	*Placement and routing* de FPGA	10.481.927

Quadro 8 - *Traces* do SPEC CINT obtidos no *BYU Trace Distribution Center* usados nos experimentos
Fonte: Elaborado pela autora

Para os *traces crafty*, *gcc* e *twolf*, com taxas de acerto mostradas na Figura 15, a mudança da cache de associativa por conjunto 2*way* para associativa por conjunto 4*way* traz ganhos. Isto porque, mesmo se existirem *slots* conflitantes (como ilustrados na seção 3.3), as perdas de desempenho provocadas por eles são menores que o ganho de se aumentar a associatividade. A cache reconfigurável 2-4*way* é bem melhor que a associativa por conjunto 2*way* e apresentou um desempenho pouco melhor que a cache 4*way*. Diferença igual a

0.052% no *crafty*, 0.276% no *gcc* e 0.090% no *twolf*. Esta diferença mostra que a 2-4*way* consegue o ganho ao se aumentar a associatividade, como acontece com a 4*way*.

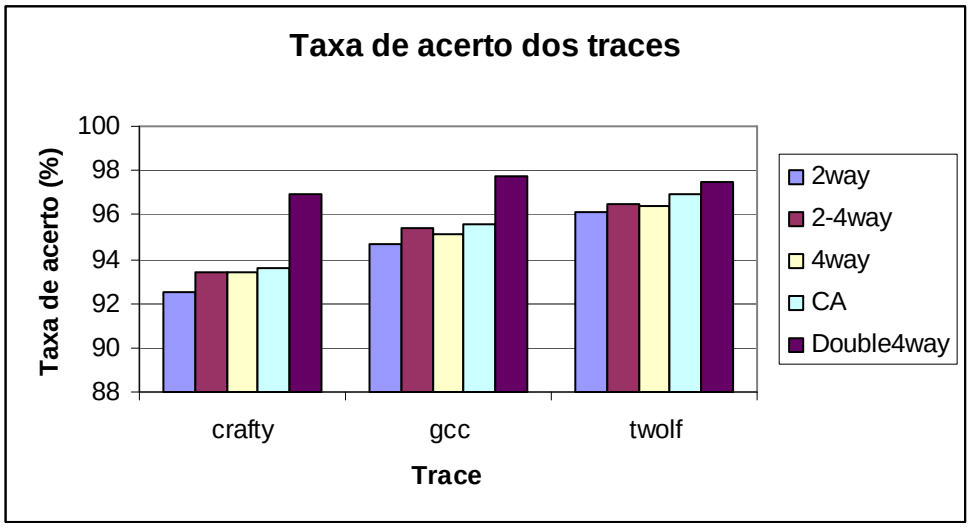

Figura 15 - Taxa de acerto dos *traces crafty*, *gcc* e *twolf*
Fonte: Elaborada pela autora

A diferença entre o desempenho da cache reconfigurável *2-4way* e a completamente associativa também é pequena 0.129% para o *crafty*, 0.205267% para o *gcc* e 0.511897% para o *twolf*. Esta diferença pequena mostra que a configuração da cache reconfigurável 2-4*way* já está próxima da configuração ideal, para este tamanho de cache, representada pela cache completamente associativa. Analisando-se esta diferença, pode-se concluir que as faltas por conflito, na cache *2-4way*, não são tão significativas quanto às faltas por capacidade para estas cargas de trabalho. Além disso, pode-se concluir que, como a diferença do ideal para a configuração atual é pequena, não há a necessidade de se aumentar a associatividade da cache para um valor maior que 4 para estes *traces*.

A Figura 16 é uma ilustração do comportamento da política de adaptação. Nela, o comportamento dos *traces crafty*, *gcc* e *twolf* pode ser ilustrado pela classificação dos *slots* (Figura 16(a)). A Figura 16(a) representada uma cache associativa por conjunto 2way, onde o *slot* GG precisa de mais blocos enquanto o *slot* PP não precisa do número de *slots* que dispõe para aumentar o desempenho. Quando os *traces* são analisados em uma cache associativa por conjunto 4way (Figura 16(b)), os conflitos que aconteciam no *slot* GG não acontecem mais, pois ele passa a compartilhar espaço com um *slot* PP, não acontecendo o conflito que ocorreu no exemplo da seção 3.3. Os *slots* PG da cache 2way compartilham um mesmo *slot* na cache 4way, não ocorrendo conflitos também. A cache reconfigurável 2-4way consegue se adaptar retirando um bloco do *slot* PP e aumentando a associatividade do *slot* GG. Os *slots* PG não são alterados, porque já possuem uma boa associatividade.

Analisando o ganho de desempenho obtido com a utilização da cache associativa por conjunto *Double4way*, em relação às demais, é possível concluir que, para se aumentar a taxa de acerto para estes *traces*, é necessário aumentar o número de blocos disponíveis, nas caches inicialmente com 4096 blocos (2*way*, 4*way*, 2-4*way* e completamente associativa).

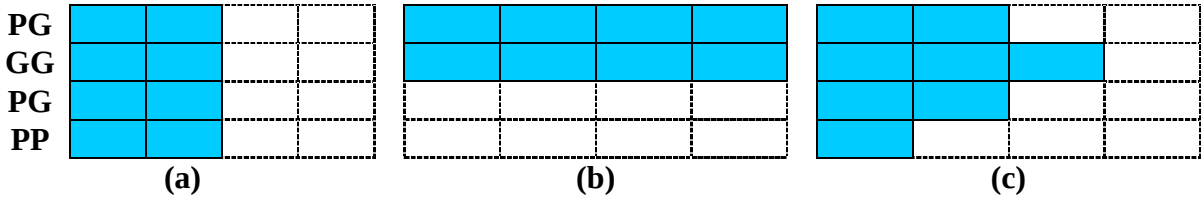

Figura 16 - Exemplo de utilização de cache com os *traces crafty*, *gcc* e *twolf*
Fonte: Elaborada pela autora

Para os *traces gap* e *perl_diffmail*, com taxa de acerto mostrada na Figura 17, como nos *traces* anteriormente apresentados, a mudança de uma cache associativa por conjunto 2*way* para uma 4*way* traz ganho de desempenho. O desempenho da cache reconfigurável 2-4*way* é melhor que o da cache associativa por conjunto 2*way*. No entanto, o desempenho da cache 2-4*way* é pior do que o da cache associativa por conjunto 4*way*.

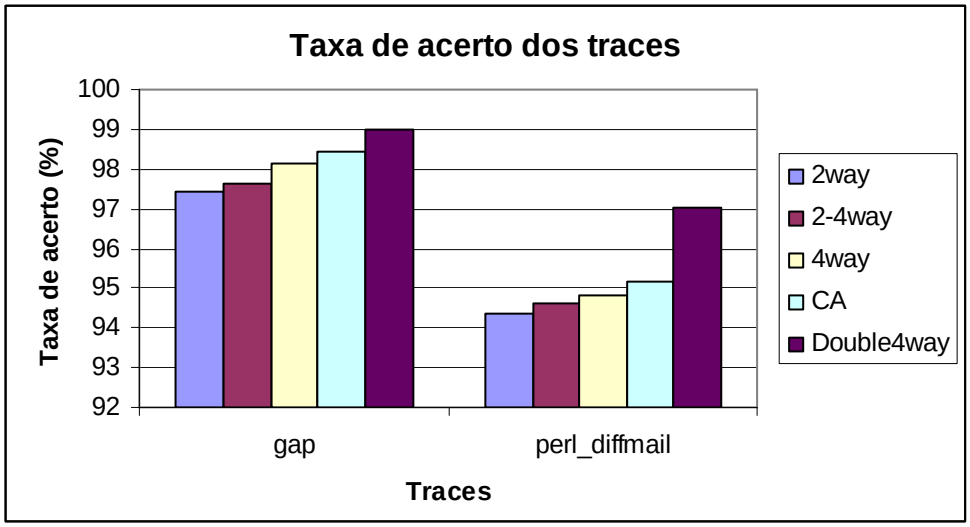

Figura 17 - Taxa de acerto dos *traces gap* e *perl_diffmail*
Fonte: Elaborada pela autora

A situação descrita anteriormente pode ter ocorrido por dois motivos: o tempo gasto para a adaptação ou por causa da carga de trabalho. No primeiro motivo, a cache vai se adaptando à carga de trabalho, mas como cada *slot* só recebe ou perde um bloco por *quantum*, o tempo gasto até a adaptação, pode acarretar numa perda de desempenho. O outro motivo é a característica da carga de trabalho. Uma das possíveis razões para isso ocorre quando a carga não possui um comportamento homogêneo, dessa maneira, a política pode ficar trocando os blocos entre os *slots* e nunca atingir uma configuração que fique constante o suficiente para

garantir um ganho de desempenho. Isto acontece porque um *slot* considerado doador em um *quantum* fica com poucos blocos disponíveis após ter sua associatividade reduzida e torna-se um *slot* carente em blocos. À medida que isso vai acontecendo, o ganho de desempenho normal quando há a adaptação da cache à carga de trabalho não acontece, pois um bloco que antes não tinha deficiência em sua associatividade passa a ter.

Nos *traces gzip_graphic*, *mcf*, *parser* e *vpr_place* a taxa de acerto das caches associativa por conjunto 2*way* e 4*way* e da completamente associativa estão bem próximas (Figura 18). Esta semelhança mostra que o número de faltas por conflito é pequeno. Além disso, a proximidade da taxa de acerto destas organizações e da cache associativa por conjunto *Double4way*, mostra que o número de faltas por capacidade também é pequeno. Através do gráfico da Figura 18 ainda podemos notar que a taxa de acerto dos *traces gzip_graphic*, *parser* e *vpr_place* é muito alta, mais de 97% nas organizações tradicionais. No entanto, a cache reconfigurável 2-4*way* apresentou desempenho pior que as organizações tradicionais. Isto é justificado pela política de adaptação da memória cache reconfigurável que tenta otimizar a cache diminuindo o número de faltas por conflito. Como este número é pequeno para estes *traces*, a política fica "confusa" e não consegue alocar os blocos adequadamente. Esta "confusão" está ilustrada na Figura 19.

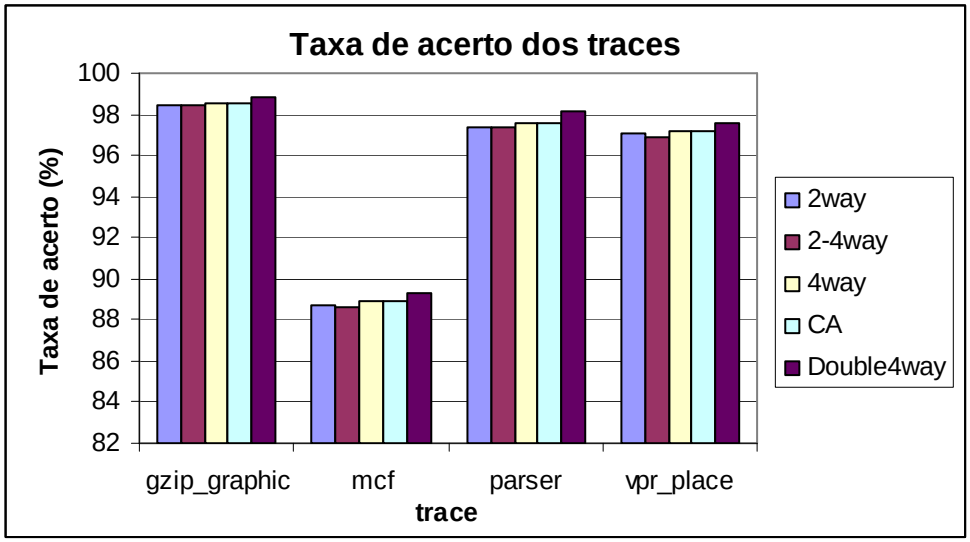

Figura 18 - Taxa de acerto dos *traces gzip_graphic*, *mcf*, *parser* e *vpr_place*
Fonte: Elaborada pela autora

No exemplo ilustrado na Figura 19, a cache reconfigurável 2-4*way* no primeiro *quantum* (semelhante a uma cache associativa por conjunto 2*way*) tem os *slots* classificados como na Figura 19(a). A partir desta classificação, a reconfiguração é realizada, retirando-se um bloco do *slot* PP e aumentando-se a associatividade do *slot* GG. No entanto, no segundo

quantum (Figura 19(b)), o *slot* com apenas um bloco, é muito acessado e ocorrem diversas faltas nele, conseqüentemente ele é (re)classificado como GG. Por outro lado, o *slot* com maior associatividade, é (re)classificado como PP. Há uma nova reconfiguração e os blocos são trocados de *slot*. E isso se repete até o fim do *trace* ou até o mesmo mudar de comportamento. Este é um problema detectado em nossa política de adaptação usada nestes experimentos. Foi possível detectar uma "confusão" na lógica de escolha da doação de blocos.

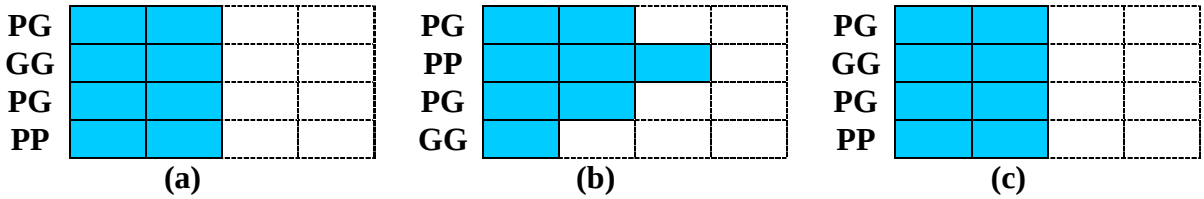

Figura 19 - Exemplo de utilização de cache com os *traces gzip_graphic*, *mcf*, *parser* e *vpr_place*
Fonte: Elaborada pela autora

O SPEC CINT não se mostrou um bom *benchmark* para nossas avaliações por causa de sua alta taxa de acerto, não sendo possível avaliar a capacidade da política de adaptação de alocar os blocos corretamente para que, através da reconfiguração da cache, houvesse uma melhoria do desempenho da mesma. Portanto, para a avaliação da nossa proposta, modificamos o tamanho dos blocos das caches simuladas para avaliarmos somente a localidade temporal da mesma, isolando os benefícios da localidade temporal (várias palavras em um bloco) obtidas nas simulações analisadas nesta seção.

4.4.3 SPEC CINT com blocos de uma palavra

Nesta subseção apresentamos e analisamos os resultados dos experimentos realizados com as organizações descritas no início da seção 4.4 (Experimentos) com uma pequena modificação no tamanho do bloco das memórias cache. Com estes experimentos, queremos verificar somente o impacto da localidade temporal da carga de trabalho nas organizações de memória cache. Esta simplificação não prejudica a avaliação da memória cache reconfigurável, pois ela está baseada em nossa hipótese de que é possível atingir um melhor aproveitamento da localidade temporal da carga de trabalho através de mudanças em sua associatividade e os prejuízos causados por esta simplificação são os mesmos para todas as organizações analisadas. Nos experimentos apresentados a seguir, cada bloco da cache contém apenas uma palavra, mas o número de blocos que podem ser armazenados é 4096

blocos nas caches 2way, 2-4way, 4way, 2-8way e completamente associativa e 8192 blocos na cache associativa por conjunto *Double4way*.

Nos gráficos das Figuras 20 e 22 são apresentadas as taxas de acerto dos *traces* do SPEC CINT utilizando-se blocos com tamanho igual a uma palavra usados em nossos experimentos. Os valores das métricas obtidos com a simulação destes *traces* são apresentados no Apêndice I. Ao analisarem-se os gráficos é importante lembrar que quanto maior é a taxa de acerto, melhor será o desempenho da memória cache.

Para os *traces gzip_graphic*, *mcf*, *parser* e *vpr_place* a taxa de acerto das caches associativa por conjunto 2*way* e 4*way* e da completamente associativa continuam relativamente bem próximas (Figura 20) considerando um mesmo *trace*, mostrando que a mudança que fizemos no tamanho do bloco não surtiu muitos efeitos no comportamento da taxa de acerto. No entanto, pode-se verificar que houve uma queda na mesma. Como eliminamos uma das técnicas para diminuir o número de faltas (aproveitamento da localidade espacial da carga de trabalho), reduzindo o tamanho do bloco para uma palavra, as taxas de acerto também diminuíram. Como todas as organizações estão com uma taxa de acerto próxima à completamente associativa, podemos afirmar que o número de faltas por conflito é pequeno.

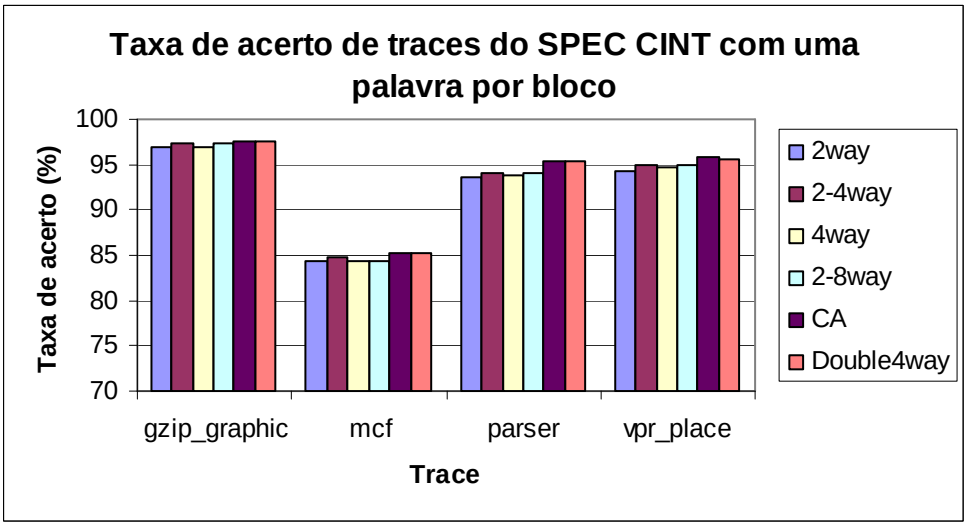

Figura 20 - Exemplo de utilização de cache com os *traces gzip_graphic*, *mcf*, *parser* e *vpr_place* com uma palavra por bloco
Fonte: Elaborada pela autora

Ainda analisando a Figura 20, é possível perceber que os valores da taxa de acerto das organizações também estão próximos da cache associativa por conjunto *Double4way*. Portanto, mesmo aumentando o tamanho da memória cache, é possível observar que não há um aumento significativo da taxa de acerto. Com isso, podemos concluir que o número de

faltas por capacidade também não é muito representativo considerando a execução destes *traces* com estas configurações de cache, pois o aumento da capacidade não alterou muito a taxa de acerto.

Como os números de faltas por conflito e por capacidade são relativamente reduzidos, pode-se concluir que as faltas compulsórias são significativas. Uma outra característica do gráfico da Figura 20 que reforça esta conclusão é a alta taxa de acerto. Como na execução do *mcf* ocorrem muitas trocas de contexto (41 na associativa por conjunto 2*way*) em relação ao *gzip_graphic* (9 na associativa por conjunto 2*way*), *parser* (17 na associativa por conjunto 2way) e *vpr_place* (14 na associativa por conjunto 2*way*), a taxa de acerto dele é menor que a dos demais.

Na Figura 20 ainda é possível observar que a cache reconfigurável 2-4*way* conseguiu um desempenho melhor que as caches associativas por conjunto 2*way* e 4*way*, diferentemente do aconteceu com os mesmos *traces* com mais palavras por bloco (Figura 18). Isto mostra que a política de adaptação conseguiu alocar corretamente os blocos para a cache reconfigurável 2-4*way*. No entanto, em relação à cache reconfigurável 2-4*way*, a cache reconfigurável 2-8*way* teve um desempenho pior. Isso provavelmente acontece porque quanto mais liberdade a política de adaptação tem (maior associatividade máxima), mais difícil de reagir a uma mudança grande no comportamento da carga de trabalho.

Como exemplo da situação citada anteriormente, a Figura 21 ilustra o comportamento da política de adaptação para uma cache 2-8*way*. Suponha que até certo ponto da execução da carga de trabalho, a cache reconfigurável possua os blocos alocados como os blocos coloridos da Figura 21(a). Neste momento, ocorre um fim de *quantum* e os *slots* são classificados como ilustrado na Figura 21(a). A classificação do *slot* com maior número de blocos como PP mostra que houve uma modificação no comportamento da carga de trabalho, pois um *slot* com muitos blocos alocados já foi um *slot* GG ou GP. Por outro lado, os *slots* GG precisam de mais blocos. Portanto, a política de adaptação tirará um bloco do *slot* PP e o doará a um *slot* GG. Como nossa política de adaptação só retira um bloco por *quantum*, a alocação de blocos ficará como a Figura 21(b). Se a carga de trabalho, no final do próximo *quantum*, se mantiver, os *slots* serão (re)classificados como na Figura 21(b), mostrando que deve haver nova (re)alocação de blocos. Portanto, ao menos mais um *quantum* será necessário para que a memória cache tenha a estrutura recomendada pela classificação de *slots* do *quantum* da Figura 21(a).

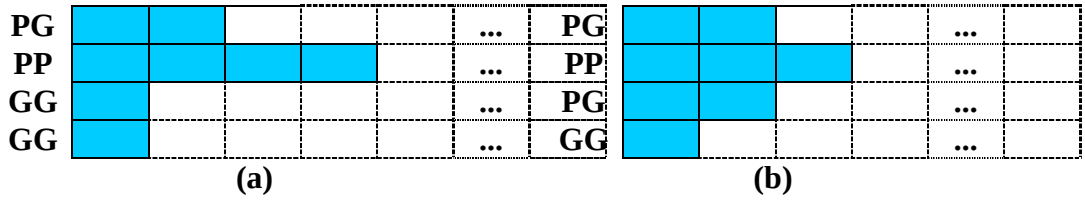

Figura 21 - Exemplo de comportamento da política de adaptação em uma cache reconfigurável 2-8*way*
Fonte: Elaborada pela autora

Na Figura 22 são apresentadas as taxas de acerto dos traces *crafty*, *gap*, *gcc*, *perl_diffmail* e *twolf*. È possível observar que a diferença na taxa de acerto das execuções dos *traces* nas cache associativa por conjunto 2*way* e 4*way*, com exceção do *gap*, é pequena, mostrando que os conflitos gerados pelo compartilhamento de *slots* na cache 4*way* que antes não eram na 2*way* não degradou o desempenho. No *trace gap*, considerando a mesma diferença, pode-se observar que houve um ganho um pouco maior (1,58%) da 4*way* em relação à 2*way*. Isto acontece porque *slots* com características opostas compartilham *slots* na cache 4*way*. *Slots* com características opostas podem ser o par *slot* carente em entrada e *slot* com bloco sobrando ou o par formado por 2 *slots* com número de entradas adequados à carga de trabalho. Estas características permitem que os *slots* que na cache 2way precisavam de blocos, se aproveitem dos blocos dos *slots* da 2way que compartilham o mesmo *slot* na 4way.

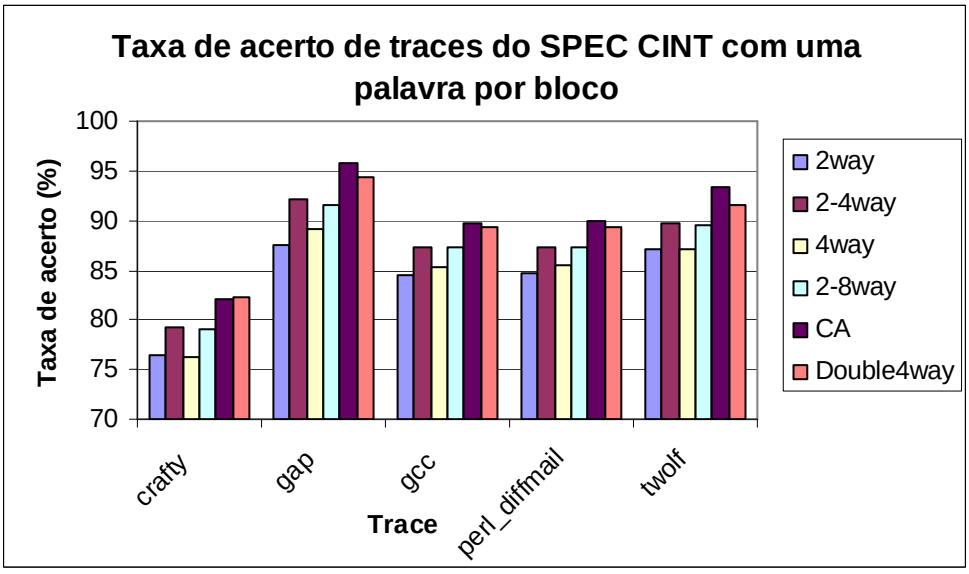

Figura 22 - Exemplo de utilização de cache com os *traces crafty*, *gap*, *gcc*, *perl_diffmail* e *twolf* com uma palavra por bloco
Fonte: Elaborada pela autora

Para todos os *traces*, a taxa de acerto da cache completamente associativa é relativamente maior que a das demais caches que armazenam a mesma quantidade de blocos. Portanto, existe uma quantidade significativa de faltas por conflito nestas outras caches. As caches reconfiguráveis não conseguiram obter um desempenho melhor porque a política de

adaptação é baseada nas características do último *quantum* de execução da carga de trabalho. Por causa disso, as mudanças na memória cache ficam um pouco defasadas no tempo em relação às mudanças na carga de trabalho, pois é necessário acabar um *quantum* para que a mudança seja realizada. Portanto, mesmo com os ganhos conseguidos em relação às caches associativa por conjunto com associatividade fixa, o desempenho das caches reconfiguráveis não é tão grande quanto o da cache completamente associativa. É necessário observar que estes ganhos, mesmo não sendo tão representativos, são maiores que outros já apresentados.

A pequena diferença entre os valores da taxa de acerto das cache reconfigurável 2-4*way* (melhor) e 2-8*way* (pior) pode ser justificada através de duas hipóteses. A primeira é que uma associatividade igual a quatro já é suficiente para evitar as faltas por conflito por *slot* na cache reconfigurável, salvas as considerações citadas anteriormente. Outra hipótese é relacionada ao exemplo ilustrado na Figura 21. Os ganhos obtidos com a cache reconfigurável 2-8*way* podem ser “camuflados” por algumas perdas como as ilustradas.

O baixo desempenho da memória cache associativa por conjunto *Double4way* em relação à cache completamente associativa é explicada pela grande quantidade de *slots* conflitantes. Mesmo com a maior quantidade de blocos possíveis de se armazenar, o número de conflitos é grande. A soma das faltas por conflito e compulsórias supera os ganhos obtidos com o aumento da capacidade.

Apesar dos bons resultados obtidos nos experimentos com os *traces* do SPEC CINT com uma palavra por bloco, melhores que os obtidos com os mesmos *traces* considerando mais palavras por bloco, não se pode considerar que eles representam totalmente a realidade, pois realizamos uma modificação na arquitetura da cache. Portanto, realizamos outros testes com o SPEC CFP.

4.4.4 SPEC CFP

Os *traces* do SPEC CFP usados em nossos experimentos são apresentados no Quadro 9. Este quadro contém a categoria da carga de trabalho e o número de acessos à memória coletados para a simulação. Assim como no SPEC CINT, o número de acessos é variável porque no BYU *Trace Distribution Center* são disponibilizados *traces* com cem milhões de referências, no entanto, para nossa simulação, filtramos o *trace*, para que somente os acessos à memória de instruções e dados fossem simulados. Como este número de acesso é variável de uma carga de trabalho para outra, o número de acessos simulados também é variável.

Nome	Categoria	Número de Acessos
applu	Equações diferenciais parciais	10.352.349
art	Redes neurais/Reconhecimento de imagem	10.367.655
galgel	Dinâmica de fluidos	10.325.242
mesa	Biblioteca de gráficos 3-D	10.366.592
mgrid	Campos potenciais 3D	10.363.192
swim	Modelagem de *Shallow Water*	10.377.743
wupwise	Cromodinâmica	10.319.347

Quadro 9 - *Traces* do SPEC CFP obtidos no *BYU Trace Distribution Center* usados nos experimentos
Fonte: Elaborado pela autora

Nos gráficos das Figuras 23 a 27 são apresentadas as taxas de acerto dos *traces* do SPEC CFP usados em nossos experimentos. Os valores das métricas obtidos com a simulação destes *traces* são apresentados no Anexo I. Ao analisar-se os gráficos é importante lembrar que quanto maior é a taxa de acerto, melhor será o desempenho da memória cache.

Analisando-se o gráfico da Figura 23, que mostra os resultados da execução do *trace galgel*, pode-se afirmar que, a cache associativa por conjunto 4*way* possui uma taxa de acerto superior à da cache 2*way*. Isto mostra que com o aumento da associatividade os conflitos foram reduzidos não existindo conflitos entre *slots* da 2*way* que passaram a compartilhar *slots* na 4*way* a ponto de degradar o desempenho. No entanto, a distância entre a taxa de acerto da cache completamente associativa e as associativas por conjunto, mostra que ainda existem muitas faltas por conflito nestas organizações. Todavia, a cache reconfigurável mostra-se como uma solução para reduzir o número de faltas por conflito. Por isso, a taxa de acerto das caches reconfiguráveis 2-4*way* e 2-8*way* apresentam um desempenho melhor que as caches associativas por conjunto.

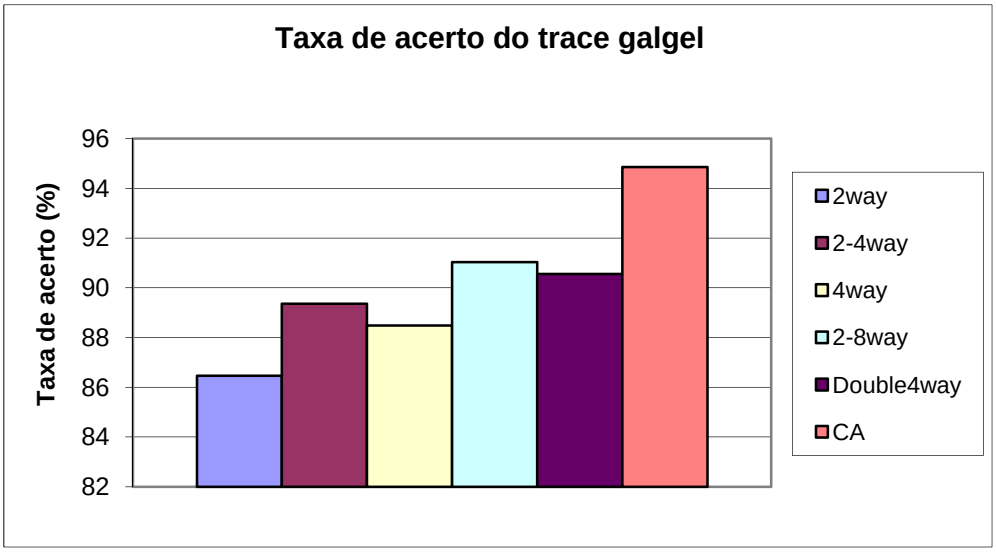

Figura 23 - Taxa de acerto da execução do *trace galgel* do SEPC CFP
Fonte: Elaborada pela autora

A cache 2-4*way* consegue se adaptar à carga de trabalho obtendo melhor desempenho que a 4*way*. Esta melhora é obtida porque a cache reconfigurável 2-4*way* usa o número de slots da cache 2*way* (evitando o compartilhamento de *slots* por *slots* conflitantes, como ocorre na 4*way*) e oferecendo blocos aos *slots* da 2*way* carentes em blocos (chegando à associatividade 4). A cache reconfigurável 2-8*way* possui as mesmas características que a 2-4*way* e ainda a possibilidade de *slots* com associatividade até 8. Como o desempenho da 2-8*way* foi melhor que o da 2-4*way*, podemos afirmar que existem *slots* com necessidade maior que 4 blocos. Como a cache reconfigurável 2-8*way* oferece a possibilidade de mais de 4 comparadores em um *slot*, esta cache teve o melhor desempenho entre as caches com associatividade restrita.

A cache 2-8*way* possui melhor desempenho até mesmo em relação à cache associativa por conjunto *Double4way*. O baixo desempenho da cache *Double4way* mostra que, mesmo aumentando-se a capacidade da memória cache, não foi possível aumentar significativamente a taxa de acerto, mostrando que existe um número significativo de faltas por conflito. Como a cache 2-8way tem a capacidade de reduzir o número de faltas por conflito, ela conseguiu um desempenho melhor que o da *Double4way*, mesmo possuindo a metade da capacidade da *Double4way*. No entanto, a diferença entre a taxa de acerto da cache completamente associativa em relação às demais, mostra que ainda possuem muitas faltas por conflito. Uma otimização na política de adaptação da memória cache reconfigurável provavelmente melhoraria o desempenho das caches reconfiguráveis.

Analisando-se o gráfico da Figura 24, que mostra os resultados da execução do *trace applu*, assim como no *trace galgel*, pode-se afirmar o aumento da associatividade de 2*way* para 4*way* melhorou o desempenho não existindo conflitos entre *slots* da 2*way* que passaram a compartilhar *slots* na 4*way* o suficiente para degradar o desempenho. Além disso, a distância entre a taxa de acerto da cache completamente associativa e as associativas por conjunto, mostra que ainda existem muitas faltas por conflito nestas organizações.

Diferentemente da execução do *trace galgel*, a taxa de acerto das caches reconfigurável 2-4*way* e 2-8*way* são muito semelhantes e inferiores a da cache 4*way*. Isto pode ter acontecidos por dois motivos ou pela união destes dois. O primeiro motivo é a não necessidade de uma associatividade máxima superior a quatro na cache reconfigurável, usando esta política. O outro motivo pode estar associado à política de adaptação. Como ilustrado pela Figura 21, a política de adaptação demora mais para se adaptar a uma mudança na carga de trabalho quando a associatividade é maior. Com isso, os possíveis ganhos de se ter uma associatividade maior podem ser mascarados por uma perda gerada pelo tempo

necessário para que a política de adaptação consiga adequar a cache reconfigurável 2-8*way* à carga de trabalho. A demora da política de adaptação para adequar a cache à carga de trabalho pode explicar a proximidade dos valores da taxa de acerto das caches reconfiguráveis e da associativa por conjunto 4*way*.

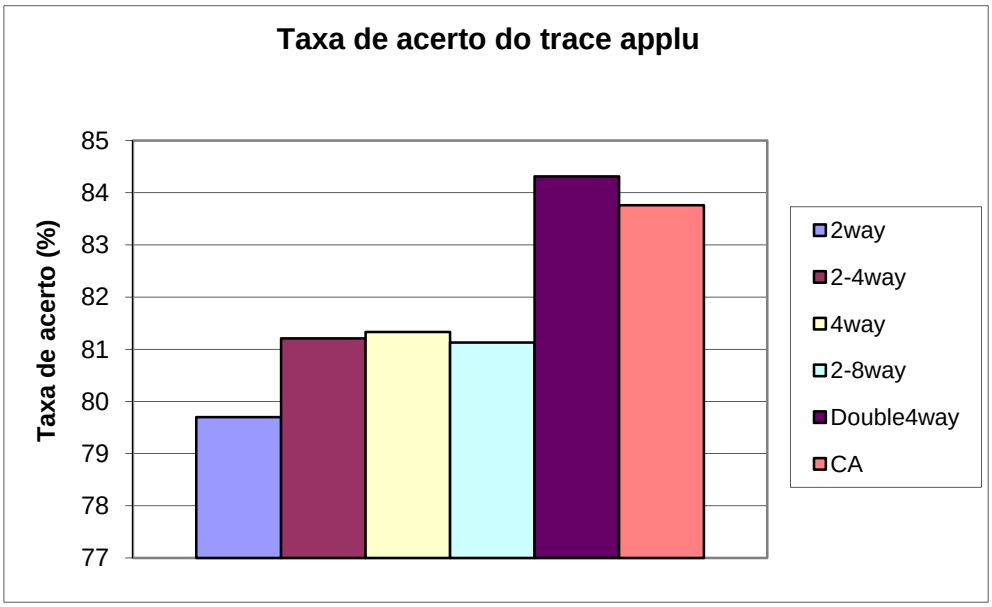

Figura 24 - Taxa de acerto da execução do *trace applu* SEPC CFP
Fonte: Elaborada pela autora

Comparando-se a taxa de acerto das caches completamente associativa e *Double4way* e a taxa de acerto das demais caches, pode-se afirmar que uma melhora de desempenho destas últimas pode ser atingida através da redução das faltas por conflito e/ou por capacidade. Com a redução das faltas por capacidade, é possível se aproximar do desempenho da cache *Double4way* ou até superá-lo. Através da redução das faltas por conflito, é possível aproximar do desempenho da cache completamente associativa. Reduzindo-se os dois tipos de falta, é possível obter uma taxa de acerto superior a todas as taxas apresentadas na Figura 24.

Analisando-se o gráfico da Figura 25, que mostra os resultados da execução do *trace mesa*, assim como nos *traces applu* e *galgel*, pode-se afirmar que o aumento da associatividade de 2*way* para 4*way* melhorou o desempenho e que ainda existem muitas faltas por conflito nas caches associativas por conjunto. A taxa de acerto da cache reconfigurável 2-4*way* é inferior à da cache associativa por conjunto 4*way*, mostrando que os problemas da política de adaptação já citados anteriormente não permitiram que fosse possível melhorar o desempenho através das vantagens oferecidas pela associatividade variável independente de *slot*.

A cache reconfigurável 2-8*way*, apesar de sofrer com os mesmos problemas da política de adaptação que a cache 2-4*way* sofre, apresentou um desempenho superior ao dela e

ao da 4*way*. Isso mostra que o *trace* mesa possui um desempenho melhor quando executado em uma cache com alguns *slots* com associatividade máxima igual a oito. A diferença entre as taxas de acerto da cache reconfigurável 2-8*way* e da completamente associativa indica que ainda existem diversas faltas por conflito, no entanto, algumas adequações na política de adaptação poderiam diminuir esta diferença significativamente. A diferença entre a taxa de acerto das caches associativa por conjunto *Double4way* e completamente associativa, mostra que existem muitas faltas por conflito na *Double4way*, porque, mesmo tendo o dobro da capacidade das demais caches, seu desempenho foi pior que o da cache completamente associativa. Isso mostra que o número de faltas por conflito foi maior que o número de acertos que antes eram faltas por capacidade.

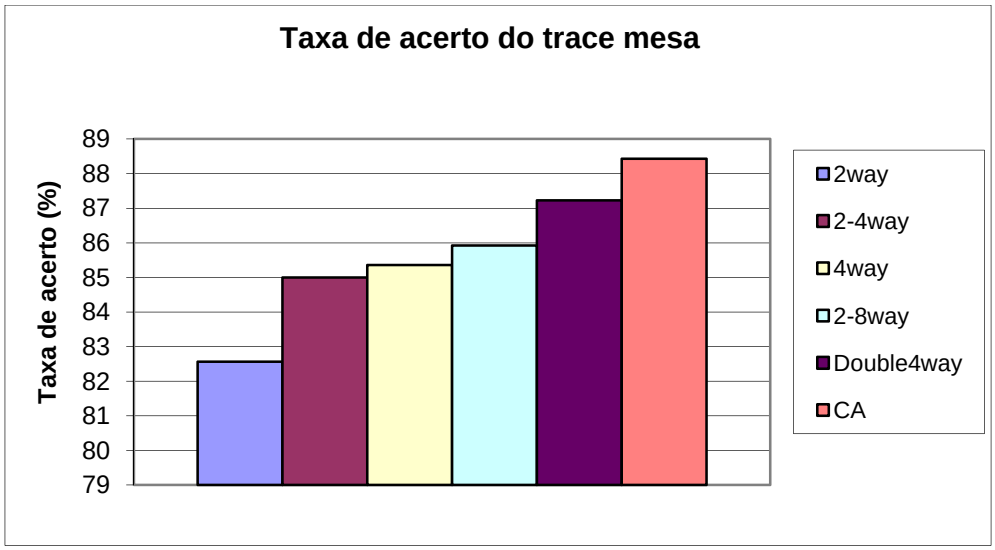

Figura 25 - Taxa de acerto da execução do *trace* mesa SEPC CFP
Fonte: Elaborada pela autora

Analisando-se o gráfico da Figura 26, que mostra os resultados da execução do *trace mgrid*, assim como nos *traces applu*, *galgel* e *mesa*, pode-se afirmar que o aumento da associatividade de 2*way* para 4*way* melhorou o desempenho e que ainda existem muitas faltas por conflito nas caches associativas por conjunto. Além disso, o desempenho das caches associativas por conjunto e reconfiguráveis com associatividade variável independente de *slot* poderia ser melhorado tanto pelo aumento da capacidade quanto pela redução das faltas por conflito.

Analisando-se a taxa de acerto das caches reconfiguráveis 2-4*way* e 2-8*way* e a da cache associativa por conjunto 4*way*, pode-se dizer que, através da utilização da reconfigurabilidade de associatividade, é possível obter um desempenho melhor que uma cache com associatividade fixa considerando o mesmo número de comparadores (cache reconfigurável 2-4*way* e cache associativa por conjunto 4*way*). O desempenho da cache 2-

8*way* foi pior que o da 4way, possivelmente, por causa dos problemas da política de adaptação explicados anteriormente: tempo gasto até o final do *quantum* (quando ocorre a reconfiguração e conseqüente adaptação à carga) e dificuldade de adequação a mudanças ocorridas na carga quando a associatividade de alguns *slots* é alta. A redução do impacto do primeiro problema da política de adaptação citado sobre o desempenho da cache pode melhorar o desempenho da cache 2-4*way* também.

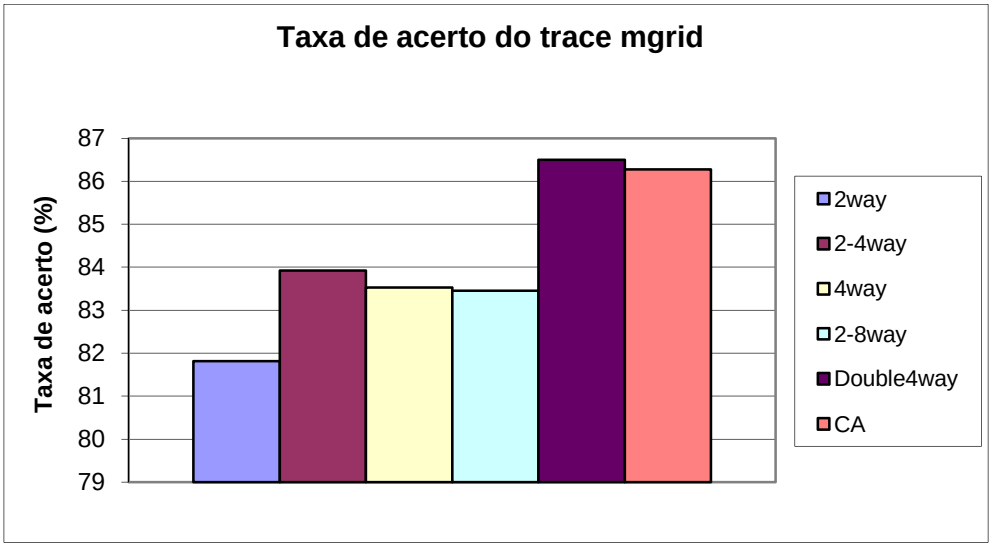

Figura 26 - Taxa de acerto da execução do *trace* mgrid SEPC CFP
Fonte: Elaborada pela autora

Analisando-se o gráfico da Figura 27, pode-se dizer que os *traces art*, *swim* e *wupwise* possuem taxas de acerto com comportamento parecido com o do *trace galgel*, apresentado na Figura 23. Portanto, as análises feitas para o *trace galgel* são válidas para estes *traces*, com exceção dos comentários feitos em relação à cache *Double4way*. Analisando a taxa de acerto obtida com a execução dos *traces* na cache *Double4way*, pode-se verificar, principalmente para o *trace* wupwise que é possível melhorar o desempenho da memória cache não só diminuindo o número de faltas por conflito, mas também aumentado a capacidade (diminuindo as faltas por capacidade).

Com os experimentos realizados com os *traces* do SPEC CFP, pudemos verificar que os comportamentos que inicialmente vimos que seriam beneficiados pela utilização de uma cache com associatividade reconfigurável independente de *slot* ocorrem em cargas de trabalho reais com memórias cache com parâmetros reais (capacidade, número de *slots*, número de comparadores, tamanho de bloco, etc.). Com isto, verificamos que a utilização de uma memória cache reconfigurável, com a associatividade independente de *slot*, traz ganhos de desempenho em relação às arquiteturas fixas, isto é, com parâmetros não variáveis após a fabricação e/ou execução.

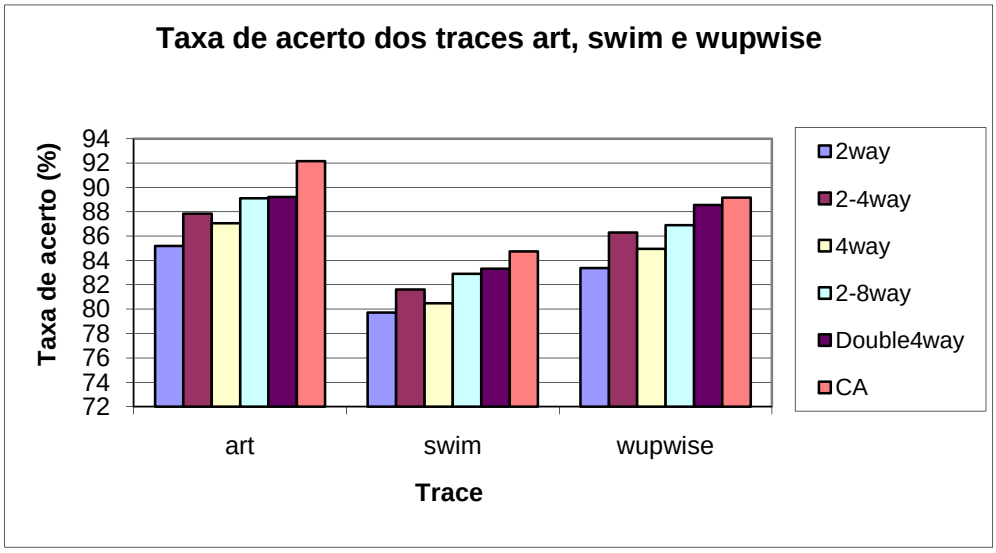

Figura 27 - Taxa de acerto da execução dos *traces* art, swim e wupwise do SPEC CFP
Fonte: Elaborada pela autora

4.4.5 Síntese dos resultados dos experimentos

Para as cargas de trabalho do SPEC CINT, a memória cache reconfigurável com associatividade variável independente de *slots*, com 4 palavras por bloco, não apresentou ganhos significativos em relação à memória cache com organização fixa, com mesma capacidade e número de comparadores, obtendo uma melhoria na taxa de acerto em apenas 33% dos *traces* simulados, com um ganho de no máximo 0,3% dos acessos. No entanto, ao modificarmos o número de palavras por bloco, para uma palavra por bloco, houve melhoria na taxa de acerto na execução de todos os *traces* do SPEC CINT simulados quando utilizamos a memória cache reconfigurável com associatividade variável independente de *slots* em relação à cache com organização fixa e mesmo número de comparadores. Nestas situações, a taxa de acerto aumentou em até 3% dos acessos realizados em cada *trace*.

Nos testes realizados com as cargas de trabalho do SPEC CFP, em 57% dos *traces* simulados, a memória cache reconfigurável com associatividade variável independente de *slots*, com 4 palavras por bloco, apresentou ganhos em relação à memória cache com organização fixa, como mesma capacidade e número de comparadores. Os aumentos na taxa de acerto chegaram a 1,3% dos acessos à memória.

Os ganhos aqui descritos são limitados pelas características da carga de trabalho, as restrições impostas à memória cache reconfigurável e ao comportamento da política de adaptação. Outros valores para a taxa de acerto podem ser encontrados na execução das

cargas de trabalho com outras memórias cache baseadas em nossa arquitetura de memória cache reconfigurável ou mesmo variando-se os parâmetros adaptáveis de uma organização e a política de adaptação.

5 CONCLUSÕES

Neste capítulo, discutimos os principais resultados obtidos, apresentamos as principais contribuições da pesquisa e alguns possíveis trabalhos futuros.

5.1 Discussão dos Resultados

Pelas limitações apresentadas no escopo de nossa pesquisa os resultados apresentados e analisados nesta dissertação se restringiram a cache reconfiguráveis com associatividade variável independente de *slot*. Para avaliar os resultados, foram implementadas duas ferramentas: um simulador (CacheSim) de memória cache com organizações tradicionais (fixas) e de nossa cache reconfigurável com associatividade variável independente de *slot*; e um conversor de *traces* do formato do BYU *Trace Distribution Center* para o formato do CacheSim.

Foram utilizados *traces* do *benchmark* SPEC 2000 CPU (CINT e CFP) obtidos no BYU *Trace Distribution Center* para analisar o desempenho de nossa proposta em relação às organizações tradicionais de memória cache. Usamos cargas de trabalho reais (*benchmarks*) e parâmetros de caches reais em nossas simulações. Na análise de desempenho apresentada nesta dissertação utilizamos uma política de adaptação simples que ainda possui algumas limitações na análise do comportamento da carga de trabalho e no processo de adaptação.

A análise dos resultados apresentados no capítulo 4 desta dissertação nos leva a dois tipos de conclusão: uma baseada na proposta de uma memória cache reconfigurável e outra baseada na política de adaptação de memória cache reconfigurável utilizada em nossos experimentos. Portanto, é necessário avaliar quais fatores, sejam eles positivos ou negativos, são provocados pela arquitetura de memória cache reconfigurável e quais são provocados pela política de adaptação.

Os experimentos realizados com os *traces* do SPEC CINT mostraram a dificuldade que a política de adaptação implementada tem para identificar e reagir a mudanças na carga de trabalho. Como a política implementada caracteriza a carga de trabalho dividindo o comportamento da mesma em relação aos *slots* da memória cache levando em consideração apenas duas métricas (número de acessos e número de faltas), algumas vezes, alguns *slots* são

mal caracterizados. Com isso, a associatividade de *slot* que é reconfigurada pode causar uma perda de desempenho na execução da carga de trabalho durante o próximo *quantum*.

Além disso, a política de adaptação implementada permite que somente uma entrada seja retirada ou adicionada por *slot* em cada *quantum* de reconfiguração. Esta restrição faz com que, mesmo identificando modificações na carga de trabalho que necessitem que mais do que uma entrada seja retirada ou adicionada em um *slot*, a cache se reconfigura, para adaptar-se à carga de trabalho, de maneira mais lenta que a desejável. Esta característica da política de adaptação prejudica, principalmente, as memórias cache reconfiguráveis com maior associatividade máxima. Isto acontece porque quanto maior a associatividade de um *slot*, um maior número de *quanta* (mais de um *quantum*) será gasto para reduzir ou aumentar o número de entradas de modo significativo.

Outro problema identificado na política de adaptação está relacionado ao momento em que ocorre a reconfiguração. Nesta política, existem *quanta* que determinam o tempo de monitoramento da carga de trabalho antes de uma reconfiguração. Portanto, a reconfiguração só ocorre ao fim de um *quantum*. Isso significa, que a cache pode passar um *quantum* quase que inteiro sem estar com uma configuração considerada melhor que a atual por causa do problema descrito anteriormente.

Apesar das limitações da política de adaptação implementada citadas anteriormente, conseguimos resultados muito bons com nossos experimentos usando as cargas de trabalho do SPEC CINT usando apenas uma palavra por bloco e as cargas do SPEC CFP. Nos testes com o SPEC CINT usando apenas uma palavra por bloco, na execução de todos os *traces* simulados, a memória cache reconfigurável com associatividade variável independente de *slot* 2-4way teve uma taxa de acerto superior a da memória cache associativa por conjunto com o mesmo número de comparadores. Além disso, nos testes com o SPEC CFP, em 57% dos traces simulados, a memória cache reconfigurável com associatividade variável independente de *slot* 2-4way com 4 palavras por bloco teve uma taxa de acerto superior a da memória cache associativa por conjunto com o mesmo número de comparadores. Portanto, concluímos que a flexibilidade e a liberdade de se ter *slots* com associatividade diferenciada e variável em uma mesma memória cache permite que a organização da memória seja adaptada às características de cada carga de trabalho produzindo um ganho no desempenho.

Através da adaptação da cache à carga de trabalho, foi possível, com uma memória cache reconfigurável com os mesmos números de comparadores e entradas de *slot* de uma cache com organização fixa, obter uma taxa de acerto maior que a da cache fixa em até 3% do total de acessos. Isso mostra que, sem aumentar o número de comparadores e/ou o tamanho da

memória cache, é possível diminuir o número de faltas por conflito através da adaptação da cache à carga de trabalho.

Com a redução do número de faltas por conflito obtida com a utilização da memória cache reconfigurável, foi possível, em alguns casos, aproximar muito da taxa de acerto da cache completamente associativa com o mesmo número de entrada (taxa máxima de acerto para uma carga de trabalho em uma cache com estes parâmetros). Nestes casos, pode-se dizer que os erros por conflito foram quase eliminados. Em alguns casos, essa redução do número de conflitos, permitiu que o desempenho da memória cache reconfigurável fosse superior ao desempenho de uma memória cache associativa por conjunto 4*way* com o dobro do número de entradas da cache reconfigurável (*Double4way*) que normalmente apresenta menos erros de capacidade.

Na maioria das simulações realizadas, o desempenho da memória cache reconfigurável foi satisfatório, sendo melhor ou igual ao desempenho de uma memória cache associativa por conjunto com associatividade igual à associatividade máxima da memória cache reconfigurável. As situações em que o desempenho não foi considerado satisfatório (menor que o desempenho de uma cache com o mesmo número de comparadores) estão associadas à política de adaptação. As cargas de trabalho em que o desempenho não foi satisfatório provocaram algumas das situações descritas anteriormente como fatores negativos da política de adaptação.

Baseado nos resultados apresentados e analisados no capítulo 4 desta dissertação e discutidos anteriormente nesta seção, podemos concluir que os objetivos principais e intermediários de nossa pesquisa foram alcançados. No entanto, alguns deles foram alcançados parcialmente. Não foi possível realizar a análise da reconfiguração de outros parâmetros da memória cache além da associatividade independente de *slot*. Além disso, não foi possível propor e implementar uma política de adaptação capaz de determinar os valores dos parâmetros reconfiguráveis de maneira ideal. Estes objetivos não foram totalmente alcançados por limitações de tempo. No entanto, os resultados apresentados nesta dissertação foram capazes de confirmar nossa hipótese de que uma memória cache reconfigurável capaz de se adaptar à carga de trabalho executada em cada instante de tempo consegue atingir um desempenho melhor que as caches com organizações fixas.

Comparando nossa proposta com os trabalhos correlatos, podemos afirmar que em nenhum deles uma arquitetura semelhante à nossa é descrita. O trabalho mais próximo do nosso, possui tempos de acesso diferenciados para os blocos na memória cache. Em nossa arquitetura não há esta característica, no entanto, talvez isto ocorra quando tentarmos

implementar fisicamente nossa arquitetura. Os demais trabalhos correlatos apresentam propostas que podem ser adicionadas à nossa arquitetura de memória cache reconfigurável, permitindo uma melhora ainda maior do desempenho da mesma.

5.2 Principais Contribuições

Nesta pesquisa propusemos uma arquitetura de memória cache reconfigurável. Como na nossa proposta de cache é possível reconfigurar diversos parâmetros da mesma e como existem limitações, principalmente de tempo, para a finalização desta dissertação, limitamos o escopo de nossa pesquisa a memórias cache reconfiguráveis com associatividade variável independente de *slot*. Portanto, nossas principais contribuições são:

a) Proposta da arquitetura de memória cache reconfigurável;
b) Proposta, simulação e verificação da arquitetura de memória cache reconfigurável com associatividade independente de *slot*;
c) Proposta e implementação de uma política de adaptação de memória cache reconfigurável com associatividade independente de *slot*;
d) Implementação da ferramenta de simulação CacheSim de caches com organizações tradicionais e de caches reconfiguráveis com associatividade independente de *slot*;
e) Implementação do conversor de *trace* do formato do BYU *Trace Distribution Center* para o CacheSim;
f) Análise de desempenho da arquitetura de memória cache reconfigurável com associatividade independente de *slot*.

5.3 Trabalhos Futuros

Dentro do escopo de nossa pesquisa, alguns objetivos não foram totalmente atingidos. Portanto como trabalhos futuros para complementar os objetivos desta dissertação, citamos a descrição e desenvolvimento de uma arquitetura de memória cache reconfigurável com outros parâmetros além da associatividade reconfigurável e a análise de desempenho da mesma.

Como possíveis parâmetros podemos citar o tamanho do bloco e a quantidade de palavras buscadas da memória inferior (*fetching*).

Outro trabalho futuro é a implementação de uma política de adaptação ou alocação de blocos em que os problemas da política implementada neste trabalho, citados anteriormente, sejam reduzidos ou eliminados. Além disso, desenvolvendo-se uma memória cache reconfigurável com outros parâmetros reconfiguráveis, será necessário implementar outras políticas de adaptação que levem em conta estes novos parâmetros.

Além dos trabalhos que faziam parte do escopo inicial de nossa pesquisa, ainda podemos citar trabalhos futuros que não estavam dentro de nossos objetivos iniciais. Um destes trabalhos é inserir em nossa memória cache reconfigurável outros parâmetros para a avaliação da configuração ideal, como consumo de energia e tolerância a falhas.

No entanto, o trabalho futuro que consideramos mais importante é a implementação física de nossa memória cache reconfigurável. Através desta implementação poderemos medir o *overhead* gerado pela reconfiguração e verificar todos os impactos do uso de uma memória cache reconfigurável em sistemas computacionais reais.

REFERÊNCIAS

ALBONESI, D. H. Selective cache ways: on-demand cache resource allocation. In: Annual International Conference on Microarchitecture, 32, 1999, Haifa, Israel. **Proceedings of the…** Washington: IEEE Computer Society, 1999. p. 248-259.

ALBONESI, D. H, et al. Dynamically tuning processor resources with adaptive processing. **IEEE Computer**, Los Alamitos, v. 36, n.12, p. 49-58, dez. 2003.

BATSON, B., VIJAYKUMAR, T. N. Reactive-associative caches. In: IEEE International Conference on Parallel Architectures and Compilation Techniques, 2001, Barcelona, Espanha. **Proceedings of the…** Washington: IEEE Computer Society, 2001. p. 49-60.

BURKS, A. W., GOLDSTINE, H. H., VON NEUMANN, J. **Preliminary discussion of the Logical Design of an Electronic Computing Instrument**. Princeton: Institute for Advanced Study, 1962. 53 p. Disponível em: http://www.cs.unc.edu/~adyilie/comp265/vonNeumann.html. Acesso em: 10 dez 2005.

BYU PERFORMANCE EVALUATION LABORATORY. **BYU Trace Distribution Center**. Disponível em: http://tds.cs.byu.edu/tds/, 2001. Acesso em: 10 dez, 2005.

COMPTON, K., HAUCK, S. Reconfigurable computing: a survey of systems and software. **ACM Computing Survey**, New York, v. 34, n. 2, p. 171-210, jun. 2002.

DEHON, A., WAWRZYNEK, J. Reconfigurable computing: what, why, and implications for design automation. In: Design Automation Conference, 36, 1999, New Orleans, EUA. **Proceedings of the…** Washington: IEEE Computer Society, 1999. p. 610-615.

DEHON, A., The density advantage of configurable computing. **IEEE Computer**, Los Alamitos, v. 33, n. 4, p. 41-49, abr. 2000.

FREITAS, H. C. **Proposta e desenvolvimento de um processador de rede com chave crossbar reconfigurável**. 2003. Dissertação (Mestrado), Pontifícia Universidade Católica de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica, Belo Horizonte, 2003.

GÓES, L. F. W., MARTINS, C. A. P. S. RJSSim: A reconfigurable job scheduling simulator for parallel processing learning. In: ASEE/IEEE Frontiers in Education Conference, 33, 2003, Westminster, EUA. **Proceedings of the…** Los Alamitos: IEEE Computer Society, 2003. p. F3C3-8.

GÓES, L. F. W., MARTINS, C. A. P. S. Reconfigurable gang scheduling algorithm. In: Workshop on job scheduling strategies for parallel processing, 10, 2004, New York, EUA. **Proceedings of the…** New York: Springer, 2004a. p. 81-101.

GÓES, L. F. W. **Proposta e desenvolvimento de um algoritmo reconfigurável de escalonamento paralelo de tarefas.** Dissertação (Mestrado), Pontifícia Universidade Católica de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica, Belo Horizonte, 2004b.

GÓES, L. F. W., MARTINS, C. A. P. S. ClusterSim: a java parallel discrete event simulation tool. In: IEEE International Conference on Cluster Computing, 6, 2004, San Diego, EUA. **Proceedings of the…** Washington: IEEE Computer Society, 2004c. p. 401-401.

GONZÁLEZ, A., ALIAGAS, C., VALERO, M. A data cache with multiple caching strategies tuned to different types of locality. In: International Conference on Supercomputing, 9, 1995, Barcelona, Spain. **Proceedings of the…** New York: ACM, 1995. p. 338-347.

HENNESSY, J. L., PATTERSON, D. A. **Arquitetura de computadores:** uma abordagem quantitativa. 3. ed. Rio de Janeiro: Editora Campus, 2003.

HENNING, L. SPEC CPU2000: Measuring CPU performance in the new millennium. **IEEE Computer**, Los Alamitos, v. 33, n. 7, p. 28-35, jul. 2000.

HILL, M. D., SMITH, A. J. Evaluating associativity in CPU caches. **IEEE Transactions on Computers**, Los Alamitos, v. 38, n. 12, p. 1612-1630, dez. 1989.

INOUE, K., ISHIHARA, T. , MURAKAMI, K. Way-predicting set-associative cache for high performance and low energy consumption. In: ACM International Symposium on Low Power Electronics and Design, 1999, San Diego, EUA. **Proceedings of the…** New York: ACM, 1999. p. 273-275.

JAIN, R. K. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling**. 1. ed. New York: John Wiley & Sons, 1991.

JOHNSON, T. L., CONNORS, D. A., HWU, W. W. Run-time adaptive cache management. In: Hawaii International Conference on System Sciences. 31, 1998, Kohala Coast, EUA. **Proceedings of the…** Washington: IEEE Computer Society, 1998. v. 7, p. 774 -775.

JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers. In: International Symposium on Computer Architecture, 17, 1990, Seattle, EUA. **Proceedings of the…** New York: ACM, 1990. p. 364-373.

KAXIRAS, S., HU, Z., MARTONOSI, M. Cache decay: exploiting generational behavior to reduce cache leakage power. In: Annual International Symposium on Computer Architecture, 28, 2011, Göteborg, Suécia. **Proceedings of the…** New York: ACM, 2001. p. 240-251.

KIM, H., SOMANI, A.K., TYAGI, A. A reconfigurable multifunction computing cache architecture. **IEEE Transactions on Very Large Scale Integration Systems**, Piscataway, v. 9, n. 4, p. 509-523, aug. 2001.

KIM, H., SOMANI, A.K., TYAGI, A. Adaptive balanced computing (ABC) microprocessor using reconfigurable functional caches (RFCs). In: IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2002, Freiburg, Alemanha. **Proceedings of the…** Washington: IEEE Computer Society, 2002. p. 138-144.

KIM, C. H., KIM, J., MUKHOPADHYAY, S., ROY, K. A forward body-biased low-leakage SRAM cache: device, circuit and architecture considerations. **IEEE Transactions on Very Large Scale Integration Systems**, Piscataway, v. 13, n. 3, p. 349-357, mar. 2005.

KURPANEK, G., et al. PA7200: A PA-RISC processor with integrated high performance MP bus interface. In: IEEE Compcon Spring '94, 1994, San Francisco, EUA. **Digest of papers…** Washington: IEEE Computer Society, 1994. p. 375-382.

LAW, A.M., KELTON, W.D. **Simulation modeling and analysis**. 2. ed. New York: McGraw-Hill, 1991.

LEE, J., KIM, S., WEEMS, C. Application-adaptive intelligent cache memory system. **ACM Transactions on Embedded Computing Systems**, New York, v. 1, n. 1, p. 56–78, nov. 2002.

LENNERSTAD, H., LUNDBERG, L. Optimal worst case formulas comparing cache memory associativity. **SIAM Journal on Computing**, Philadelphia, v. 30, v. 3, p. 872–905, mai. 2000.

MARTINS, C. A. P. S., ORDONEZ, E. D., CORRÊA, J. B. T., CARVALHO, M. B. Computação reconfigurável: conceitos, tendências e aplicações. In: Sociedade Brasileira de Computação. (Org.). **XXII Jornada de Atualização em Informática (JAI), SBC2003**, Porto Alegre: SBC, v. 2, p.339-388, 2003.

PATTERSON, D. A., HENNESSY, J. L. **Organização e projeto de computadores: a interface hardware/software**, 3. ed., Rio de Janeiro: Editora Campus, 2005.

POUSA, C.V. **Proposta e desenvolvimento de um algoritmo reconfigurável de consistência de objetos**. Dissertação (Mestrado), Pontifícia Universidade Católica de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica, Belo Horizonte, 2005.

POWELL, M. D., et al. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In: ACM/IEEE International Symposium on Microarchitecture, 34, 2001, Austin, EUA. **Proceedings of the…** Washington: IEEE Computer Society, 2001. p. 54-65.

RAMOS, L. E. S. **Proposta e desenvolvimento de funções mpi de comunicação coletiva reconfiguráveis.** Dissertação (Mestrado), Pontifícia Universidade Católica de Minas Gerais, Programa de Pós-Graduação em Engenharia Elétrica, Belo Horizonte, 2004.

RANGANATHAN, P., ADVE, S., JOUPPI, N.P. Reconfigurable caches and their application to media processing. In: International Symposium on Computer Architecture, 2000, Vancouver, Canadá. **Proceedings of the…** New York: ACM, 2000. p. 214–224.

REINMAN, G., JOUPPI, N. P. **CACTI 2.0 beta**. Disponível em: http://www.research.digital.com/wrl/people/jouppi/CACTI.html, 1999. Acesso em: 10 dez 2005.

SANGIREDDY, R., KIM, H., SOMANI, A.K. Low-power high-performance reconfigurable computing cache architectures. **IEEE Transactions on Computers**, Los Alamitos, v. 53, n. 10, p. 1274-1290, out. 2004.

SEN, S., CHATTERJEE, S., DUMIR, N. Towards a theory of cache-efficient algorithms. **Journal of the ACM**, New York, v. 49, n. 6, p. 828-858, nov. 2002.

SMITH, A. J. Cache memories. **ACM Computing Surveys**, New York, v. 14, n. 3, p. 473-530, set. 1982.

TANAKA, K., FUKAWA, T. Highly functional memory architecture for large-scale data applications. IEEE Innovative Architecture for Future Generation High-Performance Processors and Systems, 2004, Maui, EUA. **Proceedings of the…** Washington: IEEE Computer Society, 2004. p. 109-118.

TANENBAUM, A. S. **Structured computer organization**. 4. ed., Englewood Cliffs: Prentice Hall, 1998.

TANG, W., et al. Cache with adaptive fetch size, **Technical Report ICS-00-16 of University of California**. Irvine: University of California, 2000.

UHLIG, R. A., MUDGE, T. N. Trace-driven memory simulation: a survey. **ACM Computing Surveys**, New York, v. 29, n. 2, p. 128-170, jun. 1997.

VEIDENBAUM, A., et al. Adapting cache line size to application behavior. In: ACM International Conference on Supercomputing, 13, 1999, Rhodes, Grécia. **Proceedings of the…** New York: ACM, 1999. p. 145-154.

VILLASENOR, J., MANGIONE-SMITH, W. H. Configurable computing. **Scientific American Magazine**, p. 54-59, jun. 1997.

WOLF, W. A decade of hardware/software codesign. **IEEE Computer**, Los Alamitos, v. 36, n. 4, p. 38- 43, abr. 2003.

ZHANG, C., VAHID, F., NAJJAR, W. A highly configurable cache architecture for embedded systems. In: International Symposium on Computer Architecture, 30, 2003, San Diego, EUA. **Proceedings of the…** New York: ACM, 2003. p. 136-146.

ZHANG, C., VAHID, F., LYSECKY, R. A self-tuning cache architecture for embedded systems. **ACM Transactions on Embedded Computing Systems**, New York, v. 3, n. 2, p. 1-19, 2004.

APÊNDICES

Nesta seção são apresentados os dados obtidos nas simulações realizadas.

1 Dados das simulações realizadas com os traces do SPEC CINT

1.1. Cache associativa por conjunto 2way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	788.008	20	92,48
gap	10.481.868	269.814	9	97,43
gcc	10.481.937	562.203	15	94,64
gzip_graphic	10.481.993	158.586	7	98,49
mcf	10.481.962	1.179.777	40	88,74
parser	10.482.049	272.277	10	97,40
perl_diffmail	10.481.727	591.766	15	94,35
twolf	10.481.022	409.270	12	96,10
vpr_place	10.481.927	309.253	11	97,05

1.2. Cache associativa por conjunto 4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	693.154	18	93,39
gap	10.481.868	194.973	7	98,14
gcc	10.481.937	509.955	13	95,13
gzip_graphic	10.481.993	153.910	6	98,53
mcf	10.481.962	1.165.610	39	88,88
parser	10.482.049	257.325	9	97,55
perl_diffmail	10.481.727	543.873	14	94,81
twolf	10.481.022	381.989	11	96,36
vpr_place	10.481.927	297.377	10	97,16

1.3. Cache completamente associativa

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	674.170	17	93,57
gap	10.481.868	165.191	7	98,42
gcc	10.481.937	459.543	12	95,62
gzip_graphic	10.481.993	148.349	6	98,58
mcf	10.481.962	1.160.449	39	88,93
parser	10.482.049	250.509	9	97,61
perl_diffmail	10.481.727	508.420	14	95,15
twolf	10.481.022	318.931	10	96,96
vpr_place	10.481.927	296.483	10	97,17

1.4. Cache associativa por conjunto Double4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	317.098	10	96,97
Gap	10.481.868	105.283	5	99,00
Gcc	10.481.937	232.454	7	97,78
gzip_graphic	10.481.993	127.206	6	98,79
mcf	10.481.962	1.124.937	38	89,27
parser	10.482.049	191.516	8	98,17
perl_diffmail	10.481.727	311.825	9	97,03
twolf	10.481.022	267.823	9	97,44
vpr_place	10.481.927	259.226	9	97,53

1.5. Cache reconfigurável 2-4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	687.684	17	93,44
gap	10.481.868	249.657	9	97,62
gcc	10.481.937	481.059	13	95,41
gzip_graphic	10.481.993	167.611	7	98,40
mcf	10.481.962	1.195.028	40	88,60
parser	10.482.049	276.609	10	97,36
perl_diffmail	10.481.727	565.998	15	94,60
twolf	10.481.022	372.583	11	96,45
vpr_place	10.481.927	326.287	11	96,89

1.6. Taxa de acerto (%) de todas as organizações de cache

Trace	2way	4way	CA	Double 4way	2-4way
crafty	92,48	93,39	93,57	96,97	93,44
gap	97,43	98,14	98,42	99,00	97,62
gcc	94,64	95,13	95,62	97,78	95,41
gzip_graphic	98,49	98,53	98,58	98,79	98,40
mcf	88,74	88,88	88,93	89,27	88,60
parser	97,40	97,55	97,61	98,17	97,36
perl_diffmail	94,35	94,81	95,15	97,03	94,60
twolf	96,10	96,36	96,96	97,44	96,45
vpr_place	97,05	97,16	97,17	97,53	96,89

2 Dados das simulações realizadas com os traces do SPEC CINT com uma palavra por bloco

2.1. Cache associativa por conjunto 2way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	2.477.917	44	76,36
gap	10.481.868	1.315.983	26	87,45
gcc	10.481.937	1.627.766	31	84,47
gzip_graphic	10.481.993	334.272	9	96,81
mcf	10.481.962	1.634.842	41	84,40
parser	10.482.049	668.954	17	93,62
perl_diffmail	10.481.727	1.597.778	30	84,76
twolf	10.481.022	1.351.109	26	87,11
vpr_place	10.481.927	589.957	14	94,37

2.2. Cache associativa por conjunto 4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	2.488.485	44	76,26
gap	10.481.868	1.150.020	23	89,03
gcc	10.481.937	1.535.056	29	85,36
gzip_graphic	10.481.993	312.507	9	97,02
mcf	10.481.962	1.638.971	41	84,36
parser	10.482.049	636.105	16	93,93
perl_diffmail	10.481.727	1.512.077	28	85,57
twolf	10.481.022	1.356.060	26	87,06
vpr_place	10.481.927	564.311	14	94,62

2.3. Cache completamente associativa

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	1.873.094	33	82,13
gap	10.481.868	434.098	11	95,86
gcc	10.481.937	1.083.061	21	89,67
gzip_graphic	10.481.993	252.309	8	97,59
mcf	10.481.962	1.551.847	39	85,20
parser	10.482.049	490.235	13	95,32
perl_diffmail	10.481.727	1.044.813	20	90,03
twolf	10.481.022	705.275	15	93,27
vpr_place	10.481.927	444.003	11	95,76

2.4. Cache associativa por conjunto Double4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	1.864.676	33	82,21
gap	10.481.868	600.552	14	94,27
gcc	10.481.937	1.108.997	21	89,42
gzip_graphic	10.481.993	248.575	7	97,63
mcf	10.481.962	1.540.626	39	85,30
parser	10.482.049	493.182	13	95,29
perl_diffmail	10.481.727	1.118.148	22	89,33
twolf	10.481.022	891.334	18	91,50
vpr_place	10.481.927	457.702	12	95,63

2.5. Cache reconfigurável 2-4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	2.184.238	39	79,16
gap	10.481.868	828.356	18	92,10
gcc	10.481.937	1.324.371	25	87,37
gzip_graphic	10.481.993	288.877	8	97,24
mcf	10.481.962	1.592.299	40	84,81
parser	10.482.049	616.386	15	94,12
perl_diffmail	10.481.727	1.324.369	25	87,36
twolf	10.481.022	1.082.731	21	89,67
vpr_place	10.481.927	537.563	13	94,87

2.6. Cache reconfigurável 2-8way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
crafty	10.481.991	2.192.832	39	79,08
gap	10.481.868	878.517	19	91,62
gcc	10.481.937	1.329.393	25	87,32
gzip_graphic	10.481.993	288.843	8	97,24
mcf	10.481.962	1.645.800	41	84,30
parser	10.482.049	618.468	16	94,10
perl_diffmail	10.481.727	1.336.761	25	87,25
twolf	10.481.022	1.090.225	21	89,60
vpr_place	10.481.927	537.886	13	94,87

2.7. Taxa de acerto (%) de todas as organizações de cache

Trace	2way	4way	CA	Double 4way	2-4way
crafty	76,36	76,26	82,13	82,21	79,16
gap	87,45	89,03	95,86	94,27	92,10
gcc	84,47	85,36	89,67	89,42	87,37
gzip_graphic	96,81	97,02	97,59	97,63	97,24
mcf	84,40	84,36	85,20	85,30	84,81
parser	93,62	93,93	95,32	95,29	94,12
perl_diffmail	84,76	85,57	90,03	89,33	87,36
twolf	87,11	87,06	93,27	91,50	89,67
vpr_place	94,37	94,62	95,76	95,63	94,87

3 Dados das simulações realizadas com os traces do SPEC CFP

3.1. Cache associativa por conjunto 2way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	2.101.806	61	79,70
art	10.367.655	1.534.107	38	85,20
galgel	10.325.242	1.397.906	35	86,46
mesa	10.366.592	1.807.909	47	82,56
mgrid	10.363.192	1.884.432	54	81,82
swim	10.377.743	2.103.939	51	79,73
wupwise	10.319.347	1.716.029	41	83,37

3.2. Cache associativa por conjunto 4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	1.932.750	57	81,33
art	10.367.655	1.340.188	34	87,07
galgel	10.325.242	1.189.130	30	88,48
mesa	10.366.592	1.517.727	41	85,36
mgrid	10.363.192	1.706.945	50	83,53
swim	10.377.743	2.023.943	49	80,50
wupwise	10.319.347	1.551.500	37	84,97

3.3. Cache completamente associativa

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	1.681.275	51	83,76
art	10.367.655	812.425	22	92,16
galgel	10.325.242	531.410	15	94,85
mesa	10.366.592	1.199.455	34	88,43
mgrid	10.363.192	1.422.400	44	86,27
swim	10.377.743	1.583.409	40	84,74
wupwise	10.319.347	1.117.340	28	89,17

3.4. Cache associativa por conjunto Double4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	1.624.406	49	84,31
art	10.367.655	1.116.857	28	89,23
galgel	10.325.242	974.663	25	90,56
mesa	10.366.592	1.324.545	36	87,22
mgrid	10.363.192	1.399.106	42	86,50
swim	10.377.743	1.729.661	43	83,33
wupwise	10.319.347	1.180.658	29	88,56

3.5. Cache reconfigurável 2-4way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	1.945.462	57	81,21
art	10.367.655	1.260.169	32	87,85
galgel	10.325.242	1.098.422	28	89,36
mesa	10.366.592	1.555.584	42	84,99
mgrid	10.363.192	1.666.090	49	83,92
swim	10.377.743	1.908.953	47	81,61
wupwise	10.319.347	1.414.472	34	86,29

3.6. Cache reconfigurável 2-8way

Trace	Número de acessos	Número de faltas	Quanta	Taxa de acerto (%)
applu	10.352.349	1.953.900	58	81,13
art	10.367.655	1.127.457	29	89,13
galgel	10.325.242	925.855	24	91,03
mesa	10.366.592	1.459.331	40	85,92
mgrid	10.363.192	1.714.832	50	83,45
swim	10.377.743	1.774.677	44	82,90
wupwise	10.319.347	1.351.176	32	86,91

3.7. Taxa de acerto (%) de todas as organizações de cache

Trace	2way	4way	CA	Double 4way	2-4way
applu	79,70	81,33	83,76	84,31	81,21
art	86,46	88,48	94,85	90,56	89,36
galgel	82,56	85,36	88,43	87,22	84,99
mesa	81,82	83,53	86,27	86,50	83,92
mgrid	79,73	80,50	84,74	83,33	81,61
swim	83,37	84,97	89,17	88,56	86,29
applu	79,70	81,33	83,76	84,31	81,21

ANEXOS

Nesta seção apresentamos os artigos publicados e artigos submetidos aguardando aprovação com resultados parciais da pesquisa.

CARVALHO, M. B., MARTINS, C. A. P. S., **Arquitetura de Cache com Associatividade Reconfigurável**, 5º *Workshop* em Sistemas Computacionais de Alto Desempenho, Foz do Iguaçu, p. 50-57, 2004.

CARVALHO, M. B. ; GÓES, L. F. W.; MARTINS, C. A. P. S. ***Dynamically Reconfigurable Cache Architecture Using Adaptive Block Allocation Policy***. In: *Reconfigurable Architectures Workshop 2006, 2006, Rhodes Island. 20th Annual International Parallel & Distributed Processing Symposium* (IPDPS 2006), 2006. (submetido)

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.

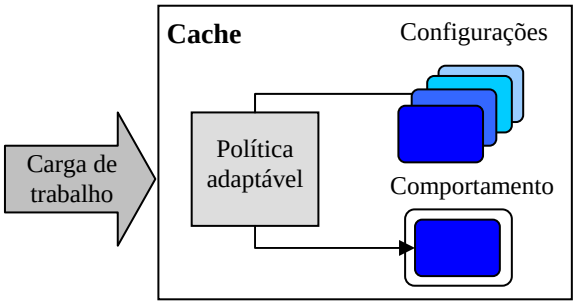

Figura 1. Arquitetura da cache com associatividade reconfigurável

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.

Tag do bloco	Índice do *slot*	Deslocamento no bloco

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.

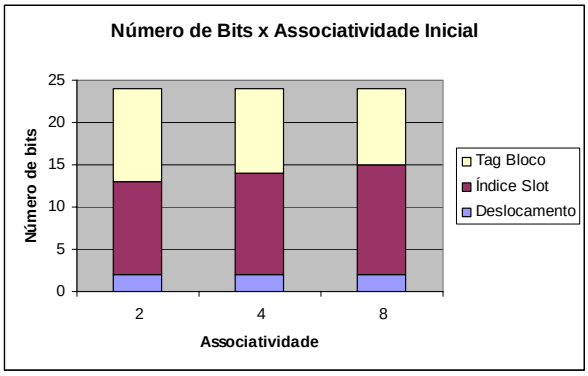

Figura 4. Tamanho das porções do endereço de uma palavra em relação à associatividade inicial da cache

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).

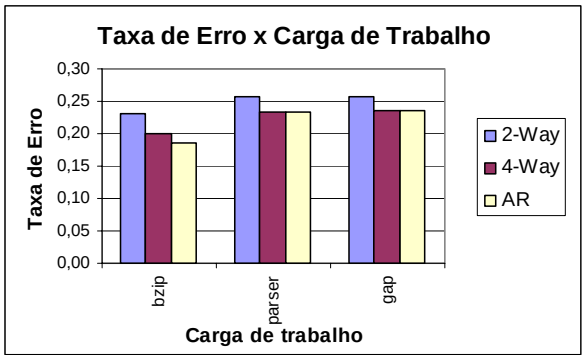

Figura 5. Taxa de erro de algumas cargas executadas em diferentes organizações de cache

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.

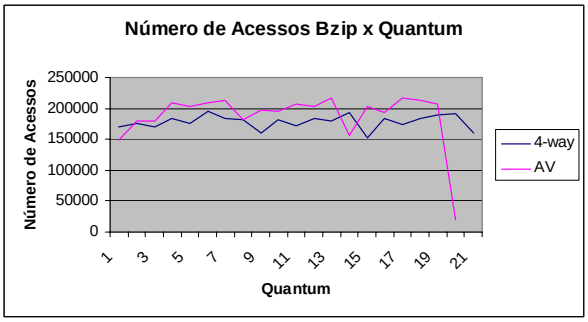

Figura 6. Número de acessos do trace bzip por quantum

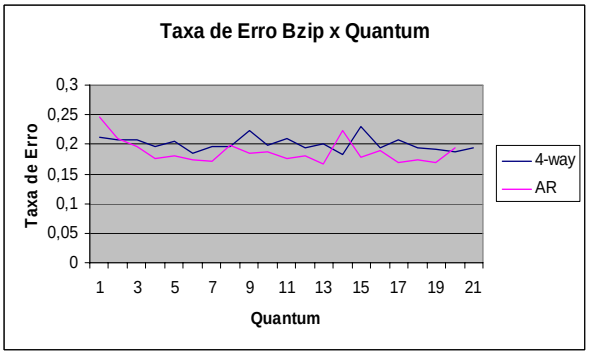

Figura 7. Taxa de erro do trace bzip por quantum

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/

Dynamically Reconfigurable Cache Architecture Using Adaptive Block Allocation Policy

Milene B. Carvalho, Luís F. W. Góes, Carlos A. P. S. Martins
Computational and Digital Systems Laboratory (LSDC)
Pontifical Catholic University of Minas Gerais - Belo Horizonte, Brazil
milene@ieee.org, lfwgoes@yahoo.com.br, capsm@pucminas.br

Abstract

In this paper, we present a dynamically reconfigurable cache architecture using adaptive block allocation policy analyzed by means of simulation. Our main objectives are: to propose a reconfigurable cache architecture and to propose, implement and analyze the performance of an adaptive cache block allocation policy. First, we present a proposal of the reconfigurable cache architecture that can adapt according to the workload. Then we present our adaptive policy and do some performance tests comparing our cache architecture with some set associative configurations. In these tests, we use some traces from BYU Trace Distribution Center of SPEC 2000 Benchmark. Finally, we analyze the results based on some metrics like cache miss ratio, response time, etc.

1. Introduction

An ideal memory for a computer system would have an infinite size and be extremely fast, that is, access time equal to zero and infinite bandwidth, but generally resources are limited. Thus, caches are designed to create the illusion to the processor of a big and fast memory [5].

The design of a cache is an optimization problem, like any computer design. This optimization is mainly related with the maximization of the hit ratio and the minimization of the access time [10]. Considering the constraints involved in the problem and these desired aspects, cache designers proposed three well-known cache organizations: direct mapped cache, fully associative cache and set associative cache.

Each organization can be better for a specific workload, that is, a specific memory trace behavior. However, it is difficult to design a cache that has a high performance for all different workloads of a general purpose processor. Thus, the designers choose cache organization/configuration that has a good performance for the most part of workloads or for the most used ones. Nevertheless, the ideal design must be optimized for all workloads. As it is not possible, an alternative is to adapt the cache organization to the workload, reconfiguring the cache or a part of it dynamically according to the executed workload characteristics [1].

In this work, we present a cache optimization based in the associativity. So, our reconfigurable cache architecture allows changing the set associativity dynamically reconfiguring itself. To perform cache reconfiguration, we developed an adaptive cache block allocation. The reconfigurable cache architecture allows other policies to be implemented, but in this paper we will describe only an adaptive cache block allocation policy based on set miss/hit and number of accesses. To verify it, we used simulation, because it appears as a less expensive (cost) alternative than cache implementation in hardware. Moreover, simulation allows detailed measurements and flexible configurations. It is also possible to determine workloads with desired characteristics [12].

Our main objectives in this article are: to propose a dynamically reconfigurable cache architecture and to propose, implement and analyze the performance of an adaptive cache block allocation policy. Our main goals are: the proposal of a dynamically reconfigurable cache architecture; proposal, development and implementation of an adaptive cache block allocation policy.

2. Related Works

Some works deal with changes in cache memory after design, dynamically adapting the cache structure or organization according to the workload. In this paper, we present a reduced number of these works. Almost all of them use monitors [7] to get information from workloads execution. Thus, an algorithm that predicts the new cache configuration for a given workload uses this information.

Considering spatial locality, there are works that present changes in the line/block [13] and in fetch [11] size. These approaches use the inherent spatial locality of applications and the memory traffic decreases. Lots of papers describe an cache associativity decreasing to minimize the cache energy dissipation while maintaining

high performance [9]. The Reactive-Associative Cache (r-a cache) [2] provides flexible associativity. It has two types of positions: direct-mapped and set-associativity, the latter has a higher hit latency than the former.

Our approach present some new ideas not found in these presented works, like different associativity between sets and reconfigurable associativity with the same access time for all sets and entries.

3. Reconfigurable Cache Architecture

Reconfigurable computing was being applied, especially in hardware, with reconfigurable devices, such as FPGAs (Field Programmable Gate Arrays), contain an array of computing elements whose behavior are determined by configuration bits [4]. Our group has been working on a reconfigurable cache that dynamically changes its behavior according to the executed workload [3] based on concepts of reconfigurable computing [4][11]. The goal of reconfigurable computing is to allow that a reconfigurable object has its structure changed to a nonpredicted state of its design time. It allows an object to adjust its behavior to a specific situation. So, this object becomes flexible, leading to a high performance compared to an object with a fixed behavior.

In our reconfigurable cache architecture (represented by Figure 1), the configuration of cache's behavior is determined by our adaptive cache block allocation policy. This policy has the parameters of system's workload and/or cache performance metrics as an input and chooses, from possible solutions (configurations), one that will configure the cache behavior.

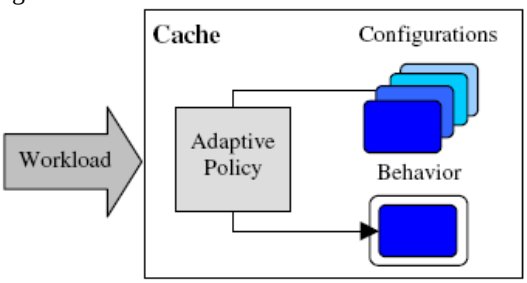

Figure 1. Reconfigurable cache architecture

In a *n*-way set associative cache, there are several sets, each one with exactly *n* entries. When the processor decodes a memory instruction and a requisition is sent to the cache, part of the address is used to locate the cache set that must be accessed. Then *n* set entries tags are verified. Our reconfigurable cache works like a set associative organization but it does not have the constraint of all sets with *n* entries. It has an initial and a maximum associativity. The latter indicates the maximum set entries that can be simultaneously verified. During the cache execution, it can dynamically adapt to the workload, changing the number of entries of each set. This number can change from one to the maximum associativity (number of comparators). In spite of these changes, the cache size is always the same of cache design. To determine the new number of entries to each set, an adaptive cache block allocation policy was designed.

4. Adaptive Block Allocation Policy

A block allocation policy assigns blocks to cache sets according to some parameters and restrictions. In our policy, we consider two parameters: the number of accesses and number of cache misses per set. The number of accesses indicates the importance level of a set and number of cache misses indicates if the number of blocks is sufficient to support the demand. These parameters are collected for a slice of time, defined as quantum. In a quantum the parameters are collected only for a processor task, so we can consider that there is one quantum for each task. It is done because each task has a behavior different of other tasks. During each quantum of a process, a statistical table is filled. This table contains performance metrics: the number of accesses and cache misses per set during the quantum and their means.

A number of accesses or cache misses is considered large if it is higher than or equal its respective mean of all the cache sets, otherwise it is considered small. An LL (large-large) set has a large number of misses and accesses, so we can consider that it is a highly accessed set with an insufficient number of block entries. Spite of having a small number of accesses, a LS (large-small) set has an insufficient number of block entries and can be improved. An SL (small-large) set has an ideal condition, because it is highly accessed and keeps a low cache miss ratio. A SS (small-small) set has a small number of accesses and misses, so it can give a block to a more important (large number of accesses) set or with a higher cache miss rate.

The three restrictions to assign blocks to sets considered in our policy are: maximum associativity value (number of comparators), number of cache sets and number of cache block entries. The increase of associativity in cache implementation really improves performance, doing a parallel search in a set to find a specific block. However, as the cost to add comparators is very expensive, designers must evaluate the cost and benefits of a higher number of comparators. As a consequence of a limited number of comparators, the number of blocks in a set cannot exceed the associativity number. On the other hand, this number cannot be less than one block, to maintain the number of sets in cache. We consider that the number of cache sets is fixed, because its variation can invalidate past configurations

(number of blocks per set), parameters (number of access and cache misses) generating re-allocation of blocks and to simplify our block allocation policy.

Based on statistics obtained during the last quantum of a specific task, our adaptive policy allocates the blocks among the cache sets to improve performance (reduce the cache miss rate). Our adaptive policy takes blocks of SS sets (donors) and gives to the LL and LS sets (receptors). Although both LL and LS sets can receive block entries, the LL sets have higher priority, because high miss rate in high accessed sets can have a higher performance cost than a less accessed set.

In the end of a quantum, our adaptive policy classifies all sets in one of the possible combinations. Then, it creates a queue of donors and receptors. In the receptors list, the LL sets come first followed by the LS sets. Then we match donors and receptors in a FCFS (first-come-first-served) way, until one of the queues becomes empty. We decrease in one the associativity of donor set and increase in one the associativity of receptor set and then both are removed from lists. Then, the cache is reconfigured and the workload is executed. This operation can be done many times (quanta) as needed to reach the end of a task. In each end of quantum, the reconfiguration is performed. Beyond of reconfiguration overhead, we must consider the overhead imposed by the adaptive policy. To determine the total overhead of reconfiguring a cache we have to consider some implementation choices as the architecture level on each the adaptive policy is implemented and the quantum size.

5. Experimental Results

To analyze the performance of our cache architecture using our adaptive block allocation policy, we developed a simulator implemented in Java and verified using known traces from Brigham Young University Trace Distribution Center [14] available on Internet. They have all memory references occurred in an execution, including operating system references.

In our simulations, we used only data access (read/write) from the available traces. The memory traces used were six traces collected from SPEC benchmarks [15] running on Pentium III and Windows 2000 from BYU Trace Distribution Center [6]. The SPEC benchmarks used in the simulations and the number of data memory accesses used were: 256.bzip (3746058), 186.crafty (3861895), 164.gzip (3647919), 254.gap (4058463), 197.parser (4099236) and 181.mcf (3874037).

In these simulations, the computer configuration is based on an Athlon XP 2000 cache and primary memory times. These times are: primary memory latency equal to 139ns; primary memory read time equals to 3.42ns; primary memory write time equals to 5.81ns; cache memory access time equals to 2ns; cache memory write time equals to 0.28ns and cache memory read time equals to 0.32ns. We used a 512MB primary memory, 64KB data cache memory, 32 bits word size and blocks with 4 words. Word and block size is always the same. The cache uses LRU (Less Recently Used) replacement policy and write-back strategy. 2-way and 4-way set associative caches were simulated for performance comparison. The reconfigurable cache architecture simulated has as initial associativity equal to 2 and 4 comparators, indicating the maximum associativity of 4. The cache size (quantity of blocks that can be stored) used in all simulated configurations is the same one. As the number of blocks that can be stored in a cache is the same in all simulations, the number of sets is variable. The sets number of the reconfigurable cache is equal to a 2-way set associative cache (initial condition).

For all simulations we considered a quantum of size equals to operational system quantum. Using this value, the overhead can be amortized, considering that the reconfiguration process is realized during the context switch, since the task processing has to be interrupted and some “new” task context must be loaded.

Our reconfigurable cache with adaptive policy has a miss ratio smaller than a 2-way set associative cache organization. As a reconfigurable cache has 4 comparators, it was expected. In 256.bzip and 164.gzip we found a difference between 2-way and 4-way set associative caches similar to the difference between 4-way set associative and reconfigurable caches (Figure 2).

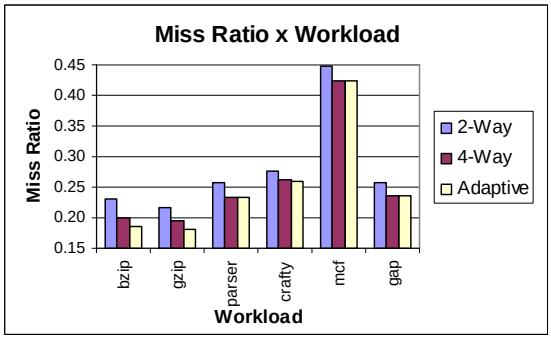

Figure 2. Miss ratio of executed workloads

In traces 197.parser and 186.crafty the difference between set associative 4-way and reconfigurable caches miss ratio is small (Figure 2). Our cache has a miss ratio 0.075% for 197.parser and 0.29% for 186.crafty smaller. In traces 181.mcf and 254.gap the difference between set associative 4-way and reconfigurable caches miss ratio is small. But our cache has a miss ratio higher than 4-way. As explained before, it is necessary to analyze this difference considering the cache memory area. This

difference could be allowed in a design that cares about used memory area as an example.

The adaptive policy works better on applications (workload) that access sets in a heterogeneous. So, the policy tries to balance the distribution of blocks, giving more blocks to sets more accessed and with more cache misses. Thus, the access conflicts in these sets are reduced. Workloads with different types of memory traces will enforce the policy to adapt the cache many times. The adaptive policy does not present good improvement with applications that access addresses in a homogeneous way. Because more accessed sets will steal blocks from less accessed but still high accessed sets. This will unbalance the distribution of blocks among the sets and can decrease performance.

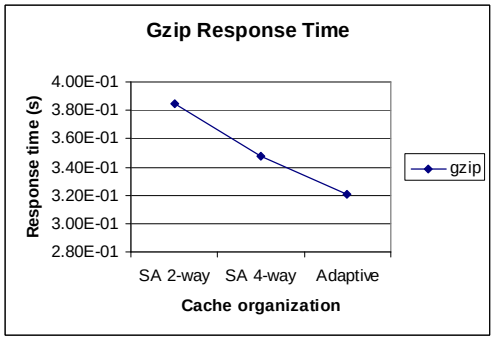

Figure 3. Response time of gzip trace

A miss ratio decreasing improves the workload response time (a lower response time). As an example, Figure 3 represents the 164.gzip response time for the simulated architectures.

6. Conclusions

In this work, we presented a reconfigurable cache architecture and an adaptive cache block allocation policy analyzed by means of simulation. So, we used a real computer (memories size and times) configuration to measure the response time and miss ratio from execution of real memory traces. We proposed, implemented and tested the adaptive policy using traces from BYU Trace Distribution Center (workload). Finally, we compared the execution of classic cache organizations (set associative) against our reconfigurable cache with the adaptive block allocation policy through some metrics. So, we concluded that the simple adaptive policy can find best cache configurations, decreasing significantly miss ratio and response time. In spite of this, the policy could work improperly on some applications that access memory sets in a homogeneous way. In this case, the policy could not improve the performance, but with other policies, the reconfigurable cache could reach better performance than a fixed cache.

Some improvements on our reconfigurable cache architecture and adaptive policy are: adaptation based on other cache parameters like block size etc; avoidance of adaptation on application with homogenous memory accesses.

Our main contributions described in this article are: the proposal of a dynamically reconfigurable cache architecture; proposal, development and implementation of an adaptive cache block allocation policy.

As future works, we remark: test and verification of multilevel caches; improvement of the adaptive policy; implementation of new adaptive policies.

7. References

[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, pp. 49-60, 2001.

[3] M. B. Carvalho, C. A. P. S. Martins. Cache Architecture with Reconfigurable Associativity, 5th WSCAD, pp. 50-57, 2004. (in Portuguese)

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

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

[6] J.L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium, IEEE Computer, 7(33): 28-35,2000.

[7] T. L. Johnson, D. A. Connors and W. W. Hwu. Runtime adaptive cache management, Proceedings of the 31th HICSS,7(6-9): 774-775, 1998.

[8] C. Martins, E. Ordonez, J. Corrêa and M. Carvalho. Reconfigurable Computing: concepts, tendencies and application. In: XXII JAI, SBC2003, Vol. 2, pp. 339 – 388 2003. (In Portuguese)

[9] 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, 2001.

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

[11] 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.

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

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

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

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