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

Proposta de uma arquitetura de um *Split Cache* reconfigurável com decisão de reconfiguração em cada *quantum*

Luíza Maria Novais Coutinho

Belo Horizonte
2011

Luíza Maria Novais Coutinho

Proposta de uma arquitetura de um *Split Cache* reconfigurável com decisão de reconfiguração em cada *quantum*

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

Belo Horizonte

2011

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

Coutinho, Luiza Maria Novais
C871p Proposta de uma arquitetura de um *Split Cache* reconfigurável com decisão de reconfiguração em cada *quantum* / Luiza Maria Novais Coutinho. Belo Horizonte, 2011.
77 f. : il.

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

1. Memória cache. 2. Arquitetura de computador. 3. Computação de alto desempenho. 4. Gerenciamento de memória (Computação). I. Martins, Carlos Augusto Paiva da Silva. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 681.31

Luíza Maria Novais Coutinho

Proposta de uma arquitetura de um *Split Cache* reconfigurável com decisão de reconfiguração em cada *quantum*

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

Dr. Carlos Augusto Paiva da Silva Martins (Orientador) – PUC Minas

Dr. Ricardo dos Santos Ferreira - UFV

Dra. Flávia Magalhães Freitas Ferreira - PUC Minas

Belo Horizonte, 30 de maio de 2011.

RESUMO

As arquiteturas dos sistemas de memória têm sido um dos principais problemas para a melhoria do desempenho dos sistemas de computação atuais. Um dos principais motivos para esse baixo desempenho relativo das arquiteturas de memória é a sua rigidez. Assim, as memórias nem sempre conseguem se adaptar aos diferentes tipos de cargas de trabalho existentes.

As memórias caches separadas (*Split Cache*) foram desenvolvidas para melhorar o desempenho dos sistemas de computação, fazendo com que um bloco de dados não dispute um mesmo *slot* da memória cache com um bloco de instruções. Mas, como no *Split Cache* tradicional a memória cache de dados e a memória cache de instruções possuem um espaço de armazenamento fixo, e como as cargas de trabalho possuem diferentes padrões de acesso à memória cache de dados e à memória cache de instruções, portanto, essas duas caches podem não conseguir se adaptar a cada carga de trabalho, que em um momento, pode necessitar de mais espaço para dados e em outro, mais espaço para instruções.

A computação reconfigurável é baseada na idéia de que o sistema deve ser alterável de maneira fácil, para que, seja específico a cada aplicação executada, assim conseguindo um melhor desempenho.

Desta forma, o principal objetivo desta dissertação é propor uma arquitetura de memória *Split Cache* reconfigurável, que tenha a capacidade de reconfigurar o grau de associatividade de cada *slot* a cada *quantum*, caso seja decidida a necessidade da reconfiguração nesse *quantum*. Nesta proposta, será permitida a doação de espaços/entradas entre as caches, assim tentando resolver o problema da rigidez das memórias *Split Cache* tradicionais.

A arquitetura proposta foi simulada e o seu desempenho foi comparado ao de dois trabalhos relacionados. Analisando os resultados, foi possível perceber que o desempenho da arquitetura proposta foi superior ao dos trabalhos relacionados, para os traces simulados. Também foi possível perceber que a arquitetura proposta optou por não se reconfigurar em todas as vezes, a reconfiguração ocorreu entre 40 e 60% das vezes em que poderia haver a reconfiguração.

Palavras-chave: memória cache, *Split Cache*, arquitetura reconfigurável, grau de associatividade.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

AC – associativa por conjunto

BYU - Brigham Young University

CA – completamente associativa

CPU – unidade central de processamento

DRAM - memória dinâmica de acesso randômico

FIFO – *first in first out*

FPGA – *Field Programmable Gate Arrays*

J2SE - Java Standard Edition

LRU – *last recently used*

MD – mapeamento direto

MSCSim – *Multilevel and Split Cache Simulator*

RAM – memória de acesso randômico

SPEC – *Standard Performance Evaluation Corporation*

SRAM - memória estática de acesso randômico

Web-MHE – *Web Memory Hierarchy Simulator*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

Desde 1945 quando John Von Neumann e seus colaboradores propuseram o conceito de programa armazenado (uma memória onde é possível armazenar os programas no mesmo espaço que os dados) (VON NEUMANN, 1945; VON NEUMANN, 1993; EIGENMANN; LILJA, 1999) objetivando melhorar o desempenho do sistema de computação, várias arquiteturas de memória foram propostas com este mesmo objetivo.

Em janeiro de 1968 foi proposto um *buffer* de alta velocidade que se situa entre o espaço de armazenamento principal e o processador. Esse *buffer* de alta velocidade ficou conhecido como cache. A cache é um bloco de memória para o armazenamento temporário de dados e instruções que possuem uma grande probabilidade de serem utilizados novamente, essa foi criada objetivando diminuir a latência de comunicação entre o processador e a memória principal (SMITH, 1982). A memória cache foi proposta e apresentada pela primeira vez em um artigo publicado no IBM *Systems Journal* em janeiro de 1968 (CONTI; GIBSON; PITKOWSKY, 1968; STIERHOFF; DAVIS, 1998).

Durante as últimas décadas, o desempenho do processador está melhorando pelo menos 60% ao ano, mas o tempo de acesso da DRAM tem melhorado menos de 10% ao ano (Figura 1) (HENNESSY; PATTERSON, 2006). Essa diferença é conhecida como o *gap* de desempenho entre o processador e a memória e este é um dos principais obstáculos para a melhoria no desempenho dos sistemas de computação (PATTERSON et al, 1997; HENNESSY; PATTERSON, 2006).

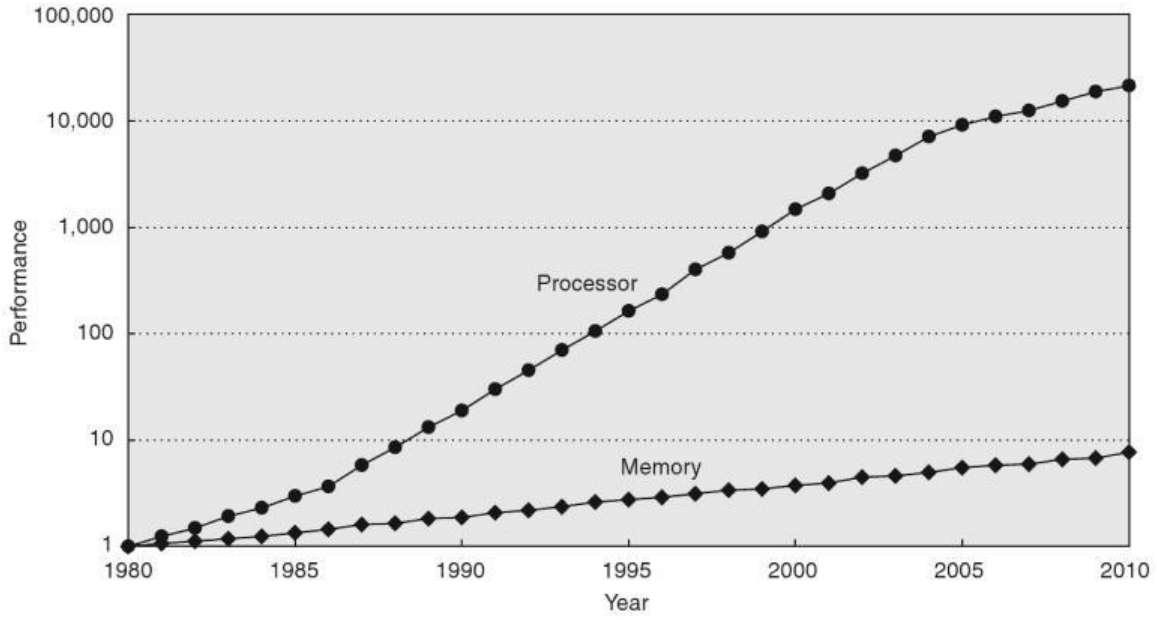


Figura 1- Gap de desempenho entre o processador e a Memória Principal (HENESSY; PATTERSON, 2006).

Uma das formas de tentar melhorar esse *gap* de desempenho entre a memória e o processador é o uso do *Split Cache* (cache separada) que utiliza conceitos da arquitetura Harvard (AIKEN; HOPPER, 1946; SMITH, 1982). O *Split Cache* melhora o desempenho da *cache*, fazendo com que um bloco de dados não dispute um mesmo *slot* com um bloco de instruções, reduzindo o perigo estrutural e diminuindo o tempo médio de acesso (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

Além desse problema relacionado ao desempenho do sistema de memória, a quantidade de dados existente está dobrando a cada 24 meses, algo que pode ser explicado pela Lei de Moore (SCHALLER, 1997; DEBENEDICTIS, 2004). Com esta inundação de dados aliados à grande variabilidade das cargas de trabalho, crescente número de dispositivos móveis se conectando a todo o momento e novas tendências, como computação de alto desempenho e o uso da Internet como meio e globalização, tornam-se necessárias transformações nas arquiteturas atuais, com o intuito de atender a esses novos contextos (AGERWALA; CHATTERJEE, 2005; BORKAR, 2005; DUBEY, 2005; RAMANATHAN; BRUENING, 2006; SARIKAYA; BUYUKTOSUNOGLU, 2010).

Os computadores atuais que possuem uma arquitetura convencional demorarão muito tempo para conseguirem processar esses dados. Para que esses

sejam processados em um tempo hábil, será necessária a criação de sistemas de computação com um melhor desempenho (IRWIN; SHEN, 2005). Estas arquiteturas convencionais, por terem uma configuração fixa predeterminada (a configuração é realizada quando este é fabricado), nem sempre conseguem se adaptar a diferentes tipos de carga de trabalho, podendo ter um desempenho bom ao processar algumas, mas um desempenho não satisfatório ao processar outras (BOSE, 2006; XU, SOHONI, MIN, HU, 2004).

Com o objetivo de melhorar o desempenho dos sistemas de computação foi proposto um novo modelo, que é a computação reconfigurável (BONDALAPATI; PRASANNA, 2002; HARTENSTEIN, 2001; BISHOP; SULLIVAN, 2003). Essa é baseada na idéia de que o sistema deve ser alterável de maneira fácil, para que seja específico a cada aplicação executada, assim conseguindo um melhor desempenho. Dispositivos reconfiguráveis são dispositivos que unem desempenho e flexibilidade. Eles realizam computação usando conexão espacial “pós-fabricação” de elementos de computação. Isso quer dizer que esses dispositivos podem ser configurados várias vezes depois de fabricados, diferentemente dos dispositivos configuráveis que podem receber uma única configuração definitiva após a sua fabricação (BONDALAPATI; PRASANNA, 2000). Os dispositivos reconfiguráveis podem então ser específicos para a aplicação executada e, quando essa mudar, o dispositivo é reconfigurado e torna-se específico para a outra aplicação (COMPTON;HAUCK, 2000; DEHON, 2000).

Um dos dispositivos que podem ser reconfigurados, visando a melhoria no desempenho, é a memória cache. O seu desempenho está diretamente relacionado com a sua organização e com a carga de trabalho (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008). A arquitetura das memórias cache presentes nos processadores atuais é fixa, isto é, quando a memória é projetada, sua configuração é definida e não pode ser modificada durante a execução de uma carga de trabalho, e, como a carga de trabalho de um processador de propósito geral é muito variável, algumas cargas de trabalho não são executadas com um desempenho satisfatório (DUBEY, 2005).

1.2 Problema Motivador

As arquiteturas dos sistemas de memória têm sido um dos principais problemas para a melhoria do desempenho dos sistemas de computação atuais. Um dos principais motivos para esse baixo desempenho relativo das arquiteturas de memória é a sua rigidez. Assim, a memória cache não consegue ter um desempenho tão eficiente durante a execução das cargas de trabalho devido à sua incapacidade de adaptação a cada carga. Isso pode resultar em um desempenho mediano para a execução das cargas de trabalho.

As memórias caches separadas (*Split Cache*) foram desenvolvidas para melhorar o desempenho da cache, fazendo com que um bloco de dados não dispute um mesmo *slot* com um bloco de instruções. Mas, como no *Split Cache* a memória cache de dados e a memória cache de instruções possuem um determinado tamanho fixo, e como as cargas de trabalho possuem diferentes padrões de acesso à memória, essas duas caches podem não conseguir se adaptar a cada carga de trabalho, que em um momento pode necessitar de mais espaço para dados e em outro momento pode necessitar de mais espaço para instruções. Portanto, um problema motivador dessa pesquisa é a incapacidade de adaptação das memórias *Split Cache* a cada carga de trabalho.

Recentemente, alguns trabalhos vêm explorando o uso da computação reconfigurável no projeto de arquiteturas de memórias cache reconfiguráveis, visando à adaptação da memória cache à carga de trabalho executada em um dado momento (CARVALHO, 2004; KERR JUNIOR, 2008; COUTINHO; MENDES; MARTINS, 2008; Veindenbaum, 1999; Ranganathan, 2000; Dasarathan; Kulandaiyan, 2002; Gil et AL, 2010; ALBONESI, 1999; SANGIREDDY; KIM; SOMANI, 2004; CHEN et al, 2007). Mas, o uso da computação reconfigurável requer: a escolha da configuração ideal para a memória cache para cada carga de trabalho executada, a determinação do melhor momento para a reconfiguração e a escolha de como será feita a reconfiguração. Dessa forma, outro problema motivador esta relacionado a esses problemas no uso da computação reconfigurável.

1.3 Objetivos e Metas

A seguir, serão apresentados e discutidos, os objetivos e as metas desta pesquisa, de acordo com os problemas motivadores apresentados.

1.3.1 Objetivos

O principal objetivo desta pesquisa é o projeto e o desenvolvimento de uma arquitetura de memória de um Split Cache reconfigurável, que tenha a capacidade de se reconfigurar de modo a melhorar o desempenho computacional, adaptando-se ao executar as diferentes cargas de trabalho.

Um objetivo derivado do problema principal é propor e desenvolver uma arquitetura de memória cache que tenha a capacidade de decidir a necessidade da reconfiguração no momento.

Outro objetivo é a determinação de uma melhor configuração para a memória cache, a ser escolhida no momento da reconfiguração, de forma a melhor se adaptar à carga de trabalho executada.

Para verificarmos a eficiência da arquitetura proposta, também temos como objetivo a comparação de desempenho entre a arquitetura proposta, a arquitetura convencional e as arquiteturas dos principais trabalhos relacionados (CARVALHO, 2004; KERR JUNIOR, 2008).

1.3.2 Metas

- Arquitetura de um *Split Cache* Reconfigurável;
- Arquitetura de um *Split Cache* Reconfigurável com uma funcionalidade que possibilita decidir a necessidade da reconfiguração em um dado momento;
- Arquitetura de um *Split Cache* Reconfigurável com uma funcionalidade que determina a melhor configuração a ser utilizada em um dado momento;

- Simulação da arquitetura proposta em software utilizando cargas de trabalho do SPEC (SPEC, 2009);
- Verificação e comparação dessa arquitetura com uma arquitetura tradicional e com os principais trabalhos relacionados;
- Análise dos resultados obtidos com a comparação das arquiteturas.

1.4 Escopo da pesquisa

De acordo com os objetivos e as metas desta pesquisa serão realizadas simulações de memórias *Split Cache* de máquinas monoprocessadas de propósito geral, para avaliar o seu desempenho em relação às memórias *Split Cache* tradicionais e às memórias propostas pelos principais trabalhos relacionados.

Esta pesquisa não tem como escopo fins industriais, mas apenas fins acadêmicos. Portanto, a implementação física da arquitetura de memória proposta está fora do escopo desta pesquisa. Dessa forma, o escopo desta pesquisa está limitado aos objetivos e metas descritos anteriormente.

1.5 Organização da Dissertação

Essa dissertação possui 5 capítulos. No capítulo 2 serão apresentados os principais conceitos relacionados à memória cache e à computação reconfigurável. Também serão apresentados nesse capítulo os principais trabalhos relacionados. No capítulo 3 será apresentada a proposta da arquitetura do *Split Cache* Reconfigurável, definindo e detalhando os seus principais módulos. Em seguida, no capítulo 4, serão apresentados e analisados os resultados da simulação da arquitetura proposta e será feita uma comparação de desempenho com os principais trabalhos relacionados. E por último serão apresentados: a discussão dos resultados, as principais contribuições e os trabalhos futuros relacionados com essa proposta.

2 REVISÃO DA LITERATURA

Neste capítulo será apresentada, primeiramente, uma visão geral de hierarquia de memória. No segundo tópico, serão apresentados alguns conceitos e características sobre memória *cache*, em seguida uma explicação e os principais benefícios do *Split Cache*. No próximo tópico, serão apresentados conceitos iniciais de arquitetura reconfigurável. E por fim, serão apresentados os principais trabalhos relacionados.

2.1 Hierarquia de memória

A hierarquia de memória prevê a existência de vários níveis de memória, cada um deles com tamanhos e tempos de acesso diferentes. As memórias mais rápidas são também as mais caras, considerando o custo do bit armazenado, ao passo que as mais lentas são mais baratas. Em função disso, as memórias mais rápidas são menores que as mais lentas (SMITH, 1982).

Atualmente existem três tecnologias principais usadas na construção das hierarquias de memória. A chamada memória principal é implementada a partir da tecnologia *DRAM* (*Dynamic Random Access Memory*), enquanto que os níveis da hierarquia mais próximos do processador (as chamadas memórias *cache*) usam a tecnologia *SRAM* (*Static Random Access Memory*). O custo por bit de armazenamento é bem menor na *DRAM* do que na *SRAM*, além de as memórias implementadas utilizando a tecnologia *DRAM* serem muito mais lentas do que as que utilizam *SRAM*. A diferença de preço é resultante do fato de a *DRAM* ocupar uma área de *chip* significativamente menor por bit de memória, assim apresentando maior capacidade para a mesma quantidade de silício. A terceira tecnologia, a do disco magnético, é usada na implementação do nível mais lento e de maior capacidade da hierarquia. O tempo de acesso e o custo por bit variam muito dependendo da tecnologia empregada. Essas diferenças no custo e no tempo de acesso justificam a implementação da hierarquia de memória, com a memória mais rápida ficando mais próxima do processador e a mais lenta mais distante

(HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

O objetivo de um sistema de hierarquia de memória é apresentar ao usuário uma capacidade de memória próxima a da tecnologia mais barata (que possui a maior capacidade), e um tempo de acesso médio próximo ao da tecnologia mais cara (que possui o menor tempo de acesso).

Os sistemas de memória organizados de maneira hierárquica tiram proveito da localidade temporal (tendência de se reutilizar informações acessadas recentemente) mantendo as informações acessadas recentemente em componentes da hierarquia situados mais próximos do processador. Estes sistemas também se beneficiam da localidade espacial (tendência a referenciar informações que estão próximas àquelas acessadas recentemente) movendo para seus componentes superiores blocos com palavras de memória contíguas (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

2.2 Memória Cache

Em janeiro de 1968 foi proposto um *buffer* de alta velocidade que se situa entre o espaço de armazenamento principal e o processador. Este *buffer* de alta velocidade ficou conhecido como cache. A cache é um bloco de memória para o armazenamento temporário de dados e instruções que possuem uma grande probabilidade de serem utilizados novamente e foi criada objetivando diminuir a latência média de comunicação entre o processador e a memória principal (SMITH, 1982). A memória cache foi proposto e apresentado pela primeira vez em um artigo publicado no IBM Systems Journal em janeiro de 1968 (CONTI; GIBSON; PITKOWSKY, 1968; STIERHOFF; DAVIS, 1998).

Quando a *CPU* (Unidade Central de Processamento) encontra na *cache* um item solicitado, isso é conhecido como acerto de *cache* (*cache hit*). Quando a *CPU* não encontra na *cache* o item que precisa, ocorre uma falta de *cache* (*cache miss*), neste caso a busca será feita na memória principal. Uma coleção de tamanho fixo de dados contendo a palavra solicitada, chamado bloco, é recuperada da memória principal e inserida na *cache*.

O local onde o bloco é inserido na cache é conhecido como *slot e esse local*

depende de qual organização a cache possui. Existem três tipos de organização, se cada bloco só tem um lugar em que pode aparecer na *cache* (um *slot* específico), a *cache* é do tipo mapeamento direto, sendo esse mapeamento feito utilizando-se o resultado do resto inteiro da divisão do endereço do bloco pelo número de blocos que a *cache* possui. Se um bloco pode ser inserido em qualquer lugar na *cache* (qualquer *slot*), ela é do tipo completamente associativa. E se um bloco pode ser inserido em um conjunto restrito de lugares na *cache* (um único *slot*, mas em várias posições/entradas/espaços nele), a *cache* é do tipo associativa por conjunto. Na *cache* associativa por conjunto o bloco é mapeado primeiramente em um *slot* e depois o bloco pode ser inserido em qualquer lugar/entrada/espaço dentro desse *slot*. Se há n entradas/espaços em um *slot,* a *cache* é chamada associativa por conjunto de n *ways*. A *cache* mapeamento direto é uma *cache* associativa por conjunto com um *way*, enquanto que uma *cache* completamente associativa com m blocos poderia ser chamada de associativa por conjunto com m *ways* em um único *slot* (HENNESSY; PATTERSON, 2006; BURGER; GOODMAN; KAGI, 1996; HANDY, 1998).

As *cache*s têm uma *tag* de endereço em cada *frame* de bloco, que fornece o endereço do bloco, e possuem também um bit de validade junto à *tag*, a fim de informar se essa entrada contém ou não um endereço válido. Um endereço é dividido primeiramente em endereço do bloco e deslocamento do bloco. O endereço do bloco é dividido ainda em campo de *tag* e campo de índice (*slot*). O campo de deslocamento do bloco seleciona os dados desejados do bloco, o campo de índice seleciona o *slot* e o campo de *tag* é comparado com as *tag*s que já estão neste *slot* em busca de um acerto.

Caso ocorra uma falta, o controlador da *cache* deve selecionar um bloco para ser substituído pelos dados ou instruções desejados. Existem três estratégias principais que são utilizadas para selecionar o bloco a ser substituído: aleatória, LRU (*Least Recently Used*) e FIFO (*First In, First Out)*. Na estratégia aleatória, os blocos são selecionados ao acaso. Na LRU, os acessos aos blocos são registrados, assim o bloco a ser substituído será aquele que não é utilizado por um tempo mais longo, esta técnica é baseada no princípio da localidade, se os blocos recentemente usados têm probabilidade de serem usados outra vez em um futuro próximo, então o melhor candidato a ser descartado é o que foi utilizado menos recentemente. A FIFO, em que o primeiro bloco a entrar é o primeiro a sair, é mais simples que a *LRU*

em questão de cálculos (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

Existem três tipos de erros/faltas que ocorrem em uma cache, sendo esses compulsório, conflito e capacidade. O erro/falta do tipo compusório ocorre quando é o primeiro acesso a um bloco, assim o bloco deve ser trazido para a cache, também é conhecido como erro de primeira referência. Já o erro/falta de capacidade ocorre se a cache não puder conter todos os blocos necessários durante a execução de um programa, porque os blocos serão descartados e mais tarde recuperados, para resolver um erro/falta por capacidade é necessário aumentar o tamanho da cache. Se a organização da cache for associativa por conjunto ou mapeamento direto, poderão ocorrer erros de conflito porque um bloco pode ser descartado e mais tarde recuperado se um número excessivo de blocos for mapeado para o seu conjunto, assim, se um erro/falta em uma cache associativa por conjuntos for um acerto em uma cache completamente associativa o erro/falta será de conflito. (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

2.3 *Split Cache*

A *cache* de dados não pode suprir todas as necessidades de memória do processador, pois este também precisa de instruções. Embora uma única *cache* pudesse tentar fornecer ambos, é possível que ela seja um gargalo. Por exemplo, quando uma instrução de carga ou armazenamento for executada, o processador com *pipeline* solicitará ao mesmo tempo uma palavra de dados e uma palavra de instrução. Conseqüentemente, uma única *cache* representaria um conflito estrutural para cargas e armazenamentos, levando a paradas. Outro problema, que poderia acontecer tendo dados e instruções em uma única *cache,* seria quando dois endereços, um de dado e outro de instrução, disputam o mesmo *slot* da *cache*. Nessa situação, um tiraria o lugar do outro, podendo gerar mais *cache miss*. Um modo de resolver esses problemas e melhorar o desempenho da *cache* é dividi-la: uma *cache* é dedicada para as instruções e a outra cache é dedicada para os dados. Essa *cache* é chamada *cache* separada ou *Split Cache* (HAIKALA; KUTVONEN, 1985; AIKEN; HOPPER, 1946; SMITH, 1982).

A *CPU* sabe se está emitindo um endereço de instrução ou um endereço de dado, e assim pode haver portas separadas para ambos, duplicando a largura de banda entre a hierarquia de memória e a *CPU*. *Cache*s separadas também oferecem a possibilidade de otimizar cada *cache* isoladamente: diferentes capacidades, tamanhos de blocos e associatividades podem levar a um melhor desempenho. A separação de instruções e dados remove erros/faltas causados por conflito entre blocos de instruções e blocos de dados. (HENNESSY; PATTERSON, 2006; HENNESSY; PATTERSON, 2008).

2.4 Computação Reconfigurável

A computação reconfigurável é uma solução intermediária entre as soluções em *Hardware* e *Software*, com objetivos relacionados com a melhoria de desempenho, flexibilidade, generalidade, eficiência, custo e outros (COMPTON; HAUCK, 2000; DEHON, 2000; COMPTON; HAUCK, 2002; HAUCK; DEHON, 2008). As principais motivações para o seu estudo são: considerar e analisar a demanda computacional de algumas importantes aplicações atuais e futuras; a inadequação de alguns modelos e estilos de computação atuais em termos de desempenho e flexibilidade e também a evolução dos dispositivos computacionais em termos de arquitetura e tecnologia (BONDALAPATI; PRASANNA, 2002; HARTENSTEIN, 2001; BISHOP; SULLIVAN, 2003; HAUCK; DEHON, 2008).

A computação reconfigurável ocorre quando um dispositivo realiza computação usando conexão espacial pós fabricação de elementos de computação (HAUCK; DEHON, 2008), enquanto os computadores tradicionais realizam computação fazendo conexões no tempo.

Os conceitos de configuração ou reconfiguração estão relacionados com o número de configurações que podemos realizar no dispositivo, ou seja, com a possibilidade de se definir ou alterar a configuração do dispositivo somente uma única vez ou então diversas vezes, sendo neste último caso possível reconfigurar-se o dispositivo muitas vezes ou constantemente se necessário. Também existe a possibilidade de se reconfigurar o dispositivo no estado inativo (parado) ou durante o seu funcionamento normal. E a reconfiguração pode ser feita em tempo de

compilação ou síntese, ou seja, o dispositivo tem a possibilidade de se reconfigurar durante a fase de execução ou processamento ou somente antes do início da fase de processamento, que pode ser durante a fase de compilação (MESQUITA et al, 2001; MARTINS et al, 2003).

As arquiteturas reconfiguráveis são aquelas em que é possível aplicar ou utilizar os conceitos de reconfigurabilidade e implementar as técnicas de reconfiguração apresentadas anteriormente. Nessas arquiteturas, os blocos (módulos) lógicos construtivos básicos podem ser reconfigurados, na sua lógica ou funcionalidade interna, e os blocos de interconexão são responsáveis pela interligação desses blocos lógicos construtivos e pela definição da estrutura da arquitetura a ser reconfigurada. Esses blocos lógicos construtivos normalmente implementam ou são as unidades funcionais de processamento, armazenamento, comunicação ou entrada e saída de dados (MARTINS et al, 2003).

As arquiteturas reconfiguráveis podem ser puramente reconfiguráveis ou então híbridas (mistas), com utilização dos modelos de *hardware* fixo e/ou *hardware* programável em conjunto com o modelo de *hardware* reconfigurável (BOBDA, 2007). Também é possível utilizar a reconfiguração em *software*, onde os objetos definidos como *hardware* podem ser redefinidos como blocos ou módulos construtivos (MARTINS et al, 2003).

Vários dispositivos podem ser reconfigurados em um sistema de computação, como: processador, memória *cache*, *pipeline*, memória *RAM* e etc. Cada um desses possui vários parâmetros que podem ser reconfigurados com o objetivo de melhorar o desempenho do sistema computacional.

Dentre os principais parâmetros que podem ser reconfigurados na memória *cache*, destacam-se: associatividade, políticas de escrita e de substituição, tamanho da memória cache, tamanho da linha, número de palavras por bloco, organização da cache e também a reconfiguração dos espaços destinados à *cache* em várias estruturas como múltiplos níveis (CARVALHO, 2004; KERR JUNIOR, 2008; COUTINHO; MENDES; MARTINS, 2008; Veindenbaum, 1999; Ranganathan, 2000; Dasarathan; Kulandaiyan, 2002; Gil et AL, 2010; ALBONESI, 1999; SANGIREDDY; KIM; SOMANI, 2004; CHEN et al, 2007).

2.5 Trabalhos Relacionados

Vários trabalhos foram propostos utilizando a computação reconfigurável e a memória cache, envolvendo a reconfiguração de diferentes parâmetros da cache.

Um desses trabalhos foi proposto por Veindenbaum (1999). Nesse trabalho a reconfiguração da cache é feita no tamanho dos blocos, dessa forma, varia-se o tamanho do bloco de 8B até 256B, sempre utilizando a potência de 2, de acordo com a localidade espacial. Portanto, o parâmetro utilizado para a reconfiguração, nesse trabalho, é o tamanho do bloco.

Já o trabalho desenvolvido por Ranganathan (2000) propõe uma cache que permite ter o seu espaço de armazenamento reconfigurado. Assim, parte desse espaço pode ser utilizado como memória cache e a outra parte como uma memória auxiliar na execução da carga de trabalho, sendo utilizada como uma *Lookup table* ou para auxiliar a pré busca (*prefetching*) ou o compilador.

O trabalho proposto por Dasarathan e Kulandaiyan (2002) utiliza os conceitos de adaptação para o projeto de uma política de substituição de blocos adaptável. Essa política não tem como base nenhuma outra política de substituição conhecida. Essa é baseada na freqüência de acessos ao bloco e no número de *hits* para definir se a substituição será feita dando prioridade ao número de *hits* ou ao tempo de acesso. Assim, o parâmetro escolhido, por esse trabalho, para a reconfiguração, é a política de substituição dos blocos.

Alguns outros trabalhos (ALBONESI, 1999, SANGIREDDY; KIM; SOMANI, 2004, CHEN et al, 2007) utilizam a computação reconfigurável no projeto de memórias cache, com o objetivo de reduzir o consumo de energia, para isso diminuem o tamanho da cache disponível. Dessa forma, o principal objetivo desses trabalhos é reduzir o consumo de energia.

Também existem trabalhos relacionados à implementação da memória cache reconfigurável em FPGA. Um desses trabalhos foi implementado por Gil et al (2010). Nesse trabalho, foi implementada em FPGA uma memória cache, que permite a reconfiguração da organização da memória cache, variando entre mapeamento direto e associativa por conjunto 2-way, com no máximo 4 slots, e do tamanho da palavra. O objetivo desse trabalho foi verificar a viabilidade de se implementar uma memória cache reconfigurável em hardware.

2.5.1 Principais trabalhos relacionados

A seguir serão apresentados os principais trabalhos relacionados, que são as propostas que reconfiguram o parâmetro relacionado ao grau de associatividade. Assim, esses trabalhos relacionados são considerados os principais devido a similaridade no parâmetro utilizado para a reconfiguração, em comparação com a arquitetura proposta. Os principais trabalhos relacionados serão considerados para verificação e comparação do desempenho da arquitetura proposta.

2.5.1.1 Carvalho e Martins

A arquitetura de memória cache proposta por Carvalho e Martins (CARVALHO; MARTINS, 2004; CARVALHO, 2005) é uma cache unificada associativa por conjunto, mas sem a obrigatoriedade de todos os slots possuirem a mesma associatividade. Essa arquitetura possui uma associatividade inicial e uma associatividade máxima, essa associatividade dos slots vai se readaptando à carga de trabalho, a cada intervalo de tempo pré-definido (quantum), objetivando sempre a melhor configuração, ou seja, sempre ao fim de cada quantum a associatividade é reconfigurada.

Essa reconfiguração é baseada em rótulos. Os rótulos são definidos de acordo com uma faixa de número de acessos e número de faltas (*cache miss*) de cada slot, conforme é apresentado no Quadro 1.

Rótulos	**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 1 - Quadro de classificação dos *slots*

Fonte: Milene, 2005

Os slots são classificados comparando o seu número de acessos e de faltas com a média geral. Os slots que possuirem o rótulo GG serão os slots que primeiramente receberão espaços, pois estão com um elevado *cache miss* e estão sendo muito acessados. Em segundo lugar como prioridade para receber espaços estão os slots classificados como GP, pois eles possuem muitas faltas mas não estão sendo muito acessados. Os slots classificados como PG já estão em uma condição ideal, pois apesar de estarem sendo muito acessados, não estão gerando muitas faltas. Já os slots com o rótulo PP serão os doadores de espaços, pois além de possuírem um pequeno número de *cache misses,* estão sendo pouco acessados, assim não necessitando de todos os espaços (associatividade) que possui.

Como um exemplo de classificação dessa arquitetura em um quantum temos o Tabela 1. A partir dessa classificação, são criadas duas listas, uma de *slots* doadores e outra de recebedores. Na lista de recebedores os *slots* GG ficam no início, sendo seguidos pelos *slots* GP. Na lista de doadores, estarão os *slots* PP.

Tabela 1 - Estatística ao fim de um *quantum*

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	

Fonte: Milene, 2005

A cada quantum, um dos *slots* que possuírem o rótulo GG ou GP, caso não tenha nenhum GG, receberá um espaço de um dos slots rotulados como PP, pois nessa arquitetura, a cada quantum, apenas um espaço é doado de um slot para o

outro. O *slot* é escolhido aleatoriamente dentre os de mesmo rótulo, caso haja mais de um com a mesma classificação.

Essa arquitetura foi simulada utilizando diversas cargas de trabalho do BYU *Trace Distribution Center* (BYU, 2009), que são cargas de trabalho reais com milhões de acessos cada. Para realizar essa simulação, os *traces* foram filtrados, para que somente os acessos à memória de instruções e dados fossem simulados. 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. Mas, houve casos em que o desempenho não foi considerado satisfatório (menor que o desempenho de uma cache com o mesmo número de comparadores). Isso se deve a alguns fatores negativos da política de adaptação.

2.5.1.2 Kerr e Midorikawa

A arquitetura Kerr e Midorikawa (KERR JUNIOR, 2008) foi proposta como uma evolução da arquitetura Carvalho e Martins, com as melhorias necessárias para melhorar o desempenho computacional.

A primeira melhoria feita por Kerr e Midorikawa foi na política de reconfiguração. O algoritmo de Carvalho e Martins classificava os slots em 4 possíveis classificações, já Kerr e Midorikawa classificam os slots em 16 diferentes classificações. As classificações são formadas pela combinação de 4 rótulos: 2P, 1P, 1G e 2G. Para calcular os 4 rótulos de falta, primeiramente, é calculada uma barreira central de faltas, a partir dessa barreira são separados o 2P e o 1P do 1G e 2G. Em seguida é calculada a barreira inferior que é a média de faltas dos conjuntos que possuem um número menor de faltas do que a barreira central. Essa barreira inferior divide o 2P do 1P. Em seguida é calculada a barreira superior da mesma forma, utilizando a média de faltas dos conjuntos que possuem um número maior de faltas do que a barreira central, dividindo-os em 1G e 2G, conforme é apresentado na figura 2. Os 4 rótulos de acesso também são calculados da mesma forma do que os rótulos de falta.

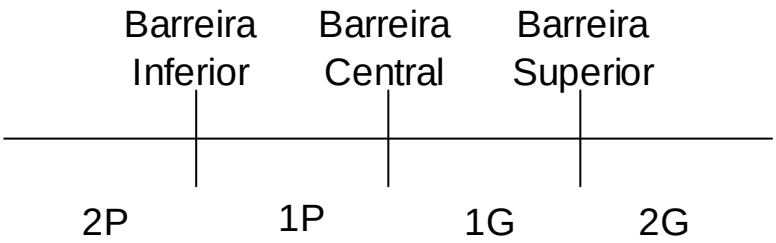


Figura 2: Classificações dos slots utilizadas pela arquitetura de Kerr e Midorikawa

Fonte: Kerr Junior, 2008

Os 4 rótulos de acesso e os 4 rótulos de falta são combinados e geram 16 possíveis classificações, conforme é apresentado no Quadro 2. Os conjuntos classificados com as maiores classificações possuem a maior prioridade para doar espaços e os conjuntos classificados com as menores classificações possuem a maior prioridade para receber espaços.

Classificação	**Prioridade**	**Descrição**
1	2G/2G	recebe dois espaços
2	2G/1G	recebe dois espaços
3	1G/2G	recebe dois espaços
4	1G/1G	recebe dois espaços
5	2G/1P	recebe um espaço
6	2G/2P	recebe um espaço
7	1G/1P	recebe um espaço
8	1G/2P	recebe um espaço
9	1P/2G	permanece estável
10	1P/1G	permanece estável
11	2P/2G	permanece estável
12	2P/1G	permanece estável
13	1P/1P	doa um espaço
14	1P/2P	doa um espaço
15	2P/1P	doa dois espaços
16	2P/2P	doa dois espaços

Quadro 2: Prioridades de doações da arquitetura de Kerr e Midorikawa

Fonte: Kerr Junior, 2008

As prioridades foram definidas seguindo o conceito de que *slots* com muitos acessos e muitas faltas precisam de mais espaços (primeiras classificações) e os *slots* com poucos acessos e poucas faltas (últimas classificações) podem doar esses espaços.

Em cada reconfiguração são doados um ou dois espaços por *slot*, sendo este um parâmetro a ser configurado antes da simulação. Essa é uma das diferenças da arquitetura de Kerr e Midorikawa para a arquitetura de Carvalho e Martins, pois esta última somente podia doar um espaço por *slot* a cada quantum.

A reconfiguração é feita seguindo um tamanho máximo de associatividade que cada *slot* pode possuir. Se o *slot* doador só possuir um espaço ele não doará e se o *slot* recebedor possuir a associatividade máxima ele não receberá mais espaços.

A arquitetura de Kerr e Midorikawa foi simulada utilizando diversos *traces* do BYU *trace distribution center* (BYU, 2009) e seu desempenho foi comparado ao de Carvalho e Martins e à arquitetura convencional. Na maior parte das simulações, o desempenho de Kerr e Midorikawa foi superior ao de Carvalho e Martins devido à melhoria no algoritmo de reconfiguração e à possibilidade de doar dois espaços por *slot* ao invés de apenas um.

2.5.1.3 Considerações finais sobre os principais trabalhos relacionados

Analisando os principais trabalhos relacionados (CARVALHO; MARTINS, 2004; KERR JUNIOR, 2008) é possível perceber que ambos utilizam um quantum fixo para reconfigurar a arquitetura, ou seja, a arquitetura sempre será reconfigurada ao final do quantum. Mesmo utilizando um quantum adequado (nem muito grande, nem muito pequeno), pode não ser necessário estar sempre reconfigurando a arquitetura em cada quantum, pois existem momentos em que caso ocorra a reconfiguração o desempenho pode piorar.

Outra desvantagem encontrada nas arquiteturas relacionadas é a política de reconfiguração. Pois, apesar do trabalho Kerr e Midorikawa classificar os slots em 16 conjuntos diferentes, como uma carga real é composta por diferentes padrões de acesso, com apenas 16 conjuntos talvez não seja possível representar tão adequadamente esses diferentes padrões de acesso à memória. Ou seja, slots com

características diferentes terão a mesma classificação e em um mesmo conjunto todos terão a mesma prioridade.

Como a maior parte dos processadores atuais possuem Split Cache, o mesmo deve ser utilizado para melhorar o desempenho da cache, pois evita o conflito entre os dados e as instruções pelo mesmo espaço. Em uma carga real, o comportamento dos acessos aos dados e às instruções pode ser bem diferente. Assim, o tamanho do espaço de armazenamento da cache de dados e da cache de instruções deve tentar seguir esse comportamento da carga. O trabalho relacionado de Kerr e Midorikawa também foi implementado utilizando Split Cache, mas a reconfiguração continuou sendo feita como era na arquitetura com caches unificadas, ou seja, doando e recebendo espaços apenas da sua própria cache. Assim, Kerr e Midorikawa não permitiam a doação de espaços entre as caches de instruções e de dados, desta forma, mesmo que uma carga de trabalho possua uma grande variabilidade no padrão de acessos aos dados ou às intruções, a arquitetura proposta por Kerr e Midorikawa não permite que a cache de dados doe espaços do seu espaço de armazenamento para a cache de instruções.

3 PROPOSTA DE UMA ARQUITETURA DE UM SPLIT CACHE RECONFIGURÁVEL

Neste capítulo será apresentada a arquitetura proposta. Inicialmente será apresentada a hipótese para a proposta da arquitetura. Em seguida, será apresentada a arquitetura geral e serão detalhados cada um dos seus 3 módulos: módulo de armazenamento reconfigurável, módulo de decisão de reconfiguração e módulo de determinação da configuração. Por fim, as considerações finais serão apresentadas.

3.1 Hipótese para a proposta de uma arquitetura de um *Split Cache* Reconfigurável

A computação reconfigurável vem sendo utilizada por diversos trabalhos (CARVALHO, 2004; KERR JUNIOR, 2008) com o objetivo de tentar resolver os problemas das arquiteturas fixas de memória cache.

Alguns dos problemas encontrados nas arquiteturas reconfiguráveis dos principais trabalhos relacionados são: a política de reconfiguração utilizando poucas faixas predeterminadas; a não verificação da necessidade de reconfiguração em cada momento pré-determinado e a impossibilidade das caches de dados e instruções poderem doar espaços entre si, pois em um determinado momento pode ser necessário mais espaços para a cache de dados e em outro momento pode ser necessário mais espaços para a cache de instruções.

A hipótese de solução dessa pesquisa para tentar solucionar esses problemas apresentados é o desenvolvimento de uma arquitetura de um Split Cache reconfigurável com três módulos. O módulo da arquitetura permitirá a doação entre caches, assim tentando resolver o problema da variabilidade dos acessos aos dados e às instruções. Outro módulo será responsável por verificar a necessidade da reconfiguração em cada momento. E o outro módulo utilizará uma política de reconfiguração diferente, ou seja, uma forma diferente de classificação dos slots para a doação/recebimento, sem utilizar faixas predeterminadas e sim utilizando

uma classificação individual para cada slot.

3.2 Apresentação da arquitetura proposta

Neste tópico será apresentada a arquitetura proposta. Em um primeiro momento, será apresentada a arquitetura de uma forma geral, mostrando o funcionamento de todos os módulos em conjunto. Em seguida, será apresentado o funcionamento do módulo de armazenamento reconfigurável. Após, será apresentado o módulo de decisão de reconfiguração. E por último, será apresentado o módulo de determinação da configuração.

3.2.1 Arquitetura Geral

A arquitetura do *Split Cache* reconfigurável é composta pelos módulos: módulo de armazenamento reconfigurável, módulo de decisão de reconfiguração e módulo de determinação de configuração. A Figura 3 apresenta as conexões, caminhos de dados e controle, entre os módulos da arquitetura do *Split Cache* Reconfigurável.

A cada quantum (intervalo de tempo parametrizável, é uma quantidade de acessos predefinida), o módulo de decisão de reconfiguração verifica a necessidade da reconfiguração do módulo de armazenamento reconfigurável, de acordo com as estatísticas de acesso da cache. Em seguida, de acordo com a decisão tomada pelo módulo de decisão de reconfiguração, o módulo de determinação da configuração irá determinar os slots que irão doar e receber espaços, a partir de dados estatísticos gerados pela cache com a execução da carga de trabalho.

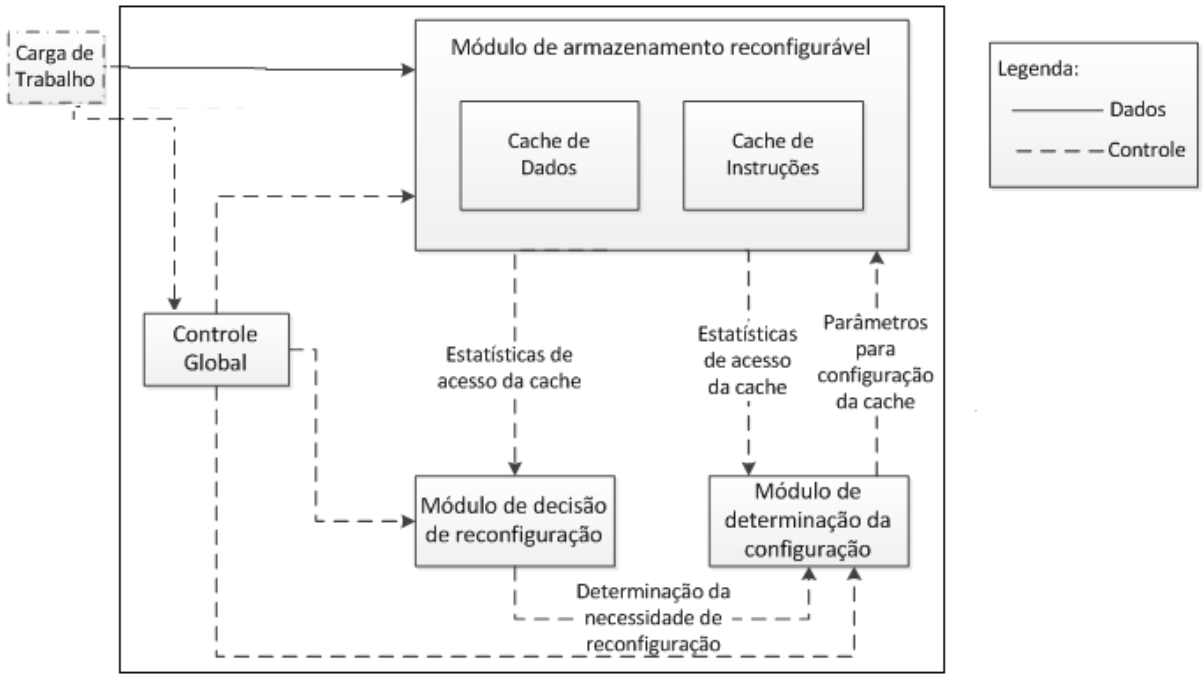


Figura 3: Arquitetura do *Split* Cache Reconfigurável

O quantum neste trabalho será um intervalo de tempo predeterminado para verificação de reconfiguração, ou seja, a cada quantum é verificada a necessidade de haver ou não a reconfiguração. Esse quantum é medido em função de uma determinada quantidade de acessos em uma carga de trabalho. O quantum não pode ser um valor muito grande, senão haverá um número muito pequeno de reconfigurações, havendo uma melhora muito pequena no desempenho, e o quantum também não pode ser um valor muito pequeno, senão haverá muita reconfiguração e o desempenho também poderá piorar.

3.2.2 Módulo de armazenamento reconfigurável

O *Split Cache* é uma arquitetura de *cache* dividida em duas partes, sendo uma responsável por armazenar os dados e outra responsável por armazenar as instruções, ambas com um tamanho fixo pré-definido. Atualmente, a maior parte dos processadores utilizam Split Cache em sua arquitetura. Um problema encontrado no uso de *Split Cache* com tamanho fixo é que as cargas de trabalho possuem diferentes comportamentos de acesso à memória cache de dados e à memória

cache de instruções. Assim, alguns slots de uma das caches podem ser pouco acessados, enquanto que outros slots da outra cache podem ser muito acessados, ou seja, alguns slots são subutilizados e outros são super utilizados. Para tentar resolver esse problema, devido as faltas por conflito, é necessário aumentar a associatividade dos slots super utilizados (CHANG; SHEU; CHEN, 2002). Assim, foi proposta neste trabalho, a arquitetura de um *Split Cache* Reconfigurável, com a associatividade reconfigurável, que se adequa ao comportamento da carga de trabalho, assim melhorando o desempenho computacional.

O *Split Cache* Reconfigurável é um *Split Cache* que possui a associatividade de cada slot reconfigurável, permitindo a cada *slot* ganhar ou perder espaços, recebendo ou doando espaços para a mesma ou para a outra *cache*. Para isso, cada *slot* tem a sua associatividade reduzida ou aumentada, à medida que doa ou recebe espaços de outro slot, podendo esses slots serem da mesma cache ou de caches diferentes (cache de dados ou cache de instruções).

A arquitetura proposta inicia a execução de uma carga de trabalho com uma configuração inicial semelhante à das memórias cache comuns, existentes em processadores atuais, e terá a possibilidade de se reconfigurar utilizando o módulo de determinação da configuração proposto. O único parâmetro que será reconfigurado será o grau de associatividade de cada slot, os outros parâmetros permanecem fixos. Cada cache possui um grau de associatividade inicial comum para todos os seus slots.

A reconfiguração acontece quando um *slot* doa um espaço para outro *slot* (da mesma ou da outra cache) que possui uma maior probabilidade de necessitar de mais espaços/entradas no próximo quantum, assim, podendo aumentar o número de *cache hits*. A doação pode ser intra cache ou inter caches, podendo ocorrer entre slots dentro da mesma cache ou entre slots da cache de dados e da cache de instruções.

A cada intervalo de tempo predeterminado (quantum), a cache pode ter a associatividade de algum *slot* aumentada ou diminuída, conforme decida o módulo de decisão de reconfiguração. O slot escolhido como o doador, pelo módulo de determinação da configuração, terá a sua associatividade reduzida em um espaço. O slot escolhido como o recebedor, pelo módulo de determinação da configuração, terá a sua associatividade aumentada em um espaço. Mas podem ser escolhidos um ou mais doadores e recebedores, assim podendo haver a doação de um ou mais

espaços por vez.

A Figura 4 mostra um exemplo de comportamento do módulo de armazenamento reconfigurável. A *cache* de dados e a cache de instruções, inicialmente, possuem grau de associatividade 2-*way* (2 entradas/espaços por *slot*), organização do tipo associativa por conjunto e possuem 8 *slot*s cada.

No exemplo da Figura 4, após alguns *quanta* (mais de um *quantum*), os *slots* 0, 1 e 3 da *cache* de instrução e o *slot* 1 da *cache* de dados doaram 4 espaços, sendo 3 desses espaços para o *slot* 4 da *cache* de dados e 1 espaço foi destinado para o *slot* 2 da *cache* de instruções.

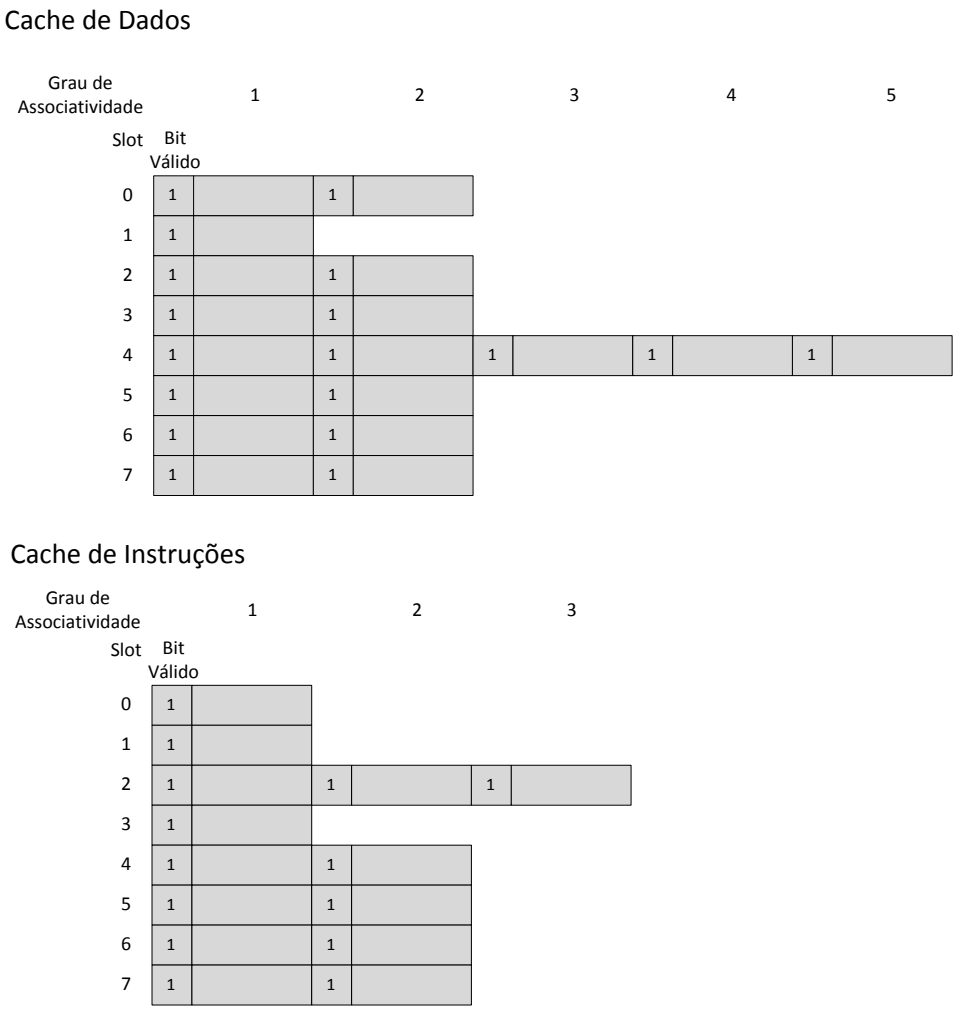


Figura 4: Estrutura do módulo de armazenamento reconfigurável

Quando a doação dos espaços for inter caches, entre a cache de dados e a cache de instruções, poderão ser reduzidos os erros do tipo capacidade, pois a cache recebedora terá seu espaço de armazenamento aumentado. Entretanto,

quando a doação for intra cache, ou seja, dentro da mesma cache, serão reduzidos os erros do tipo conflito, pois o slot terá a sua associatividade aumentada.

3.2.3 Módulo de decisão de reconfiguração

O módulo de decisão de reconfiguração, com o uso da heurística de decisão de reconfiguração, tem como finalidade verificar a necessidade de reconfiguração da arquitetura em cada *quantum*. Assim sendo, a arquitetura poderá não ser reconfigurada em todo *quantum*, o oposto do que ocorre nos trabalhos relacionados, em que há a reconfiguração em todos os *quanta*. A arquitetura só será reconfigurada quando for constatada pela heurística de decisão de reconfiguração a necessidade de haver a reconfiguração. Caso a configuração atual da arquitetura já seja a adequada para o próximo *quantum*, então não será necessária a reconfiguração, assim evitando que ocorram reconfigurações desnecessárias. Desta forma, os *slots* que perderiam espaços naquele *quantum* deixarão de perdê-los. Assim, resultando em um melhor desempenho devido a uma maior taxa de *cache hit* e a um menor tempo total de acesso, caso seja considerado um tempo para que ocorra a reconfiguração.

A métrica mais importante para a definição da necessidade da reconfiguração é a variabilidade do número de acessos aos *slots*, ou seja, se os acessos aos *slots* estão variando muito, o comportamento da carga também está variando. Assim, se durante um determinado quantum determinados slots estão sendo muito acessados e outros estão sendo pouco acessados e após outro *quantum*, esse comportamento muda e os slots que estavam sendo muito acessados passam a não ser tão acessados, isso significa que o comportamento da carga mudou e é necessária a reconfiguração.

Após diversas simulações e análises de comportamentos de cargas, foi desenvolvida uma heurística para verificar a necessidade de haver a reconfiguração ao final de cada *quantum*. Essa heurística consiste no armazenamento da quantidade de acessos que cada $slot_n$ teve em um determinado $quantum_X$. Após um segundo $quantum_Y$, esses novos acessos a cada $slot_n$ também são armazenados. Em seguida será calculado o módulo da diferença entre a quantidade de acessos de cada $slot_n$ nos *quanta* $_X$ e $_Y$ e em seguida esses valores serão somados (Equação 1).

$$índice\ de\ decisão = \sum_{n=1}^{n} |(\mathrm{slot_n})_X - (\mathrm{slot_n})_Y|$$

(1)

Caso o resultado desse índice de decisão, calculado no $quantum_Y$, seja maior ou igual a quantidade de acessos ocorridos em cada *quantum* (no $quantum_X$ e no $quantum_Y$), a reconfiguração será necessária, caso contrário, a execução da carga continua e a quantidade de acessos de cada slot no $quantum_X$ continua armazenada para uma futura comparação. Dessa forma, em um próximo $quantum_Z$ são armazenados os acessos a cada slot e é feita novamente a soma do módulo (valor absoluto) das diferenças dos acessos por slot entre o $quantum_X$ e o $quantum_Z$. Assim, a heurística consegue verificar as mudanças no comportamento da carga, mesmo que elas estejam ocorrendo de forma lenta, variando muito pouco em cada *quantum*, pois sempre será comparado o momento atual com o último momento em que houve a reconfiguração (ou com o primeiro *quantum* caso não tenha havido nenhuma reconfiguração), podendo este ser o *quantum* imediatamente anterior ou um *quantum* ocorrido há mais tempo.

No exemplo da Figura 5 é apresentado um exemplo de funcionamento do módulo de decisão de reconfiguração. O tamanho do quantum utilizado nesse exemplo foi de 20 acessos, sendo assim, a cada 20 acessos, a heurística de decisão de reconfiguração avaliará se será necessária, ou não, a reconfiguração.

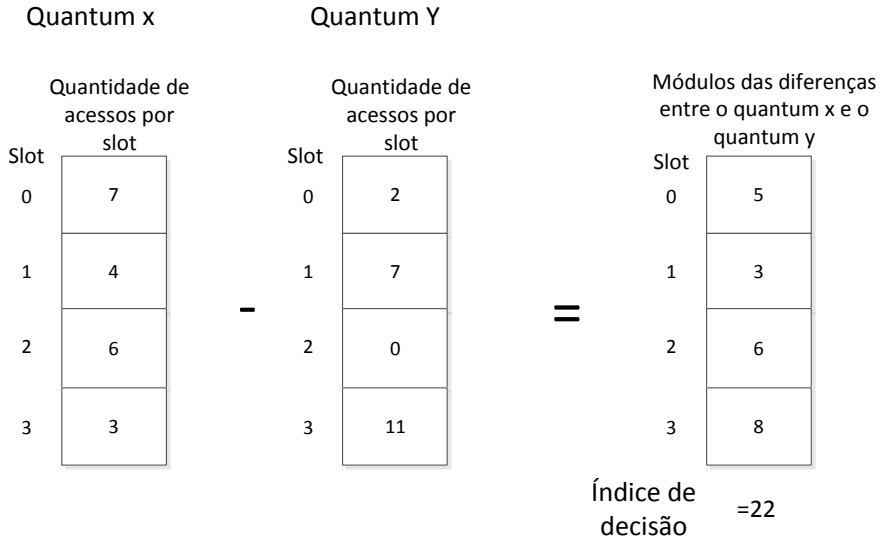


Figura 5: Funcionamento do módulo de decisão de reconfiguração

A Figura 5 apresenta o comportamento de uma cache com 4 *slots* utilizando o módulo de decisão de reconfiguração. Após o *quantum* de 20 acessos o módulo da heurística calculou o módulo da diferença entre a quantidade de acessos no $quantum_X$ e a quantidade de acessos no $quantum_Y$ e por último calculou a soma desses valores, chegando a índice de decisão igual a 22. Assim, de acordo com essa heurística, como o valor do índice de decisão foi maior do que a quantidade de acessos por *quantum* (20 acessos), então seria necessária a reconfiguração, pois a carga variou bastante o seu comportamento. Isso pode ser verificado pelo exemplo, pois analisando a quantidade de acessos que cada *slot* teve no $quantum_X$ e comparando-o com a quantidade de acessos que esses mesmos slots tiveram no $quantum_Y$, após 20 acessos, é possível perceber que o comportamento da carga de trabalho variou bastante, assim sendo necessária a reconfiguração, tentando adaptar a arquitetura à carga de trabalho. No exemplo foi utilizado uma cache unificada para facilitar o entendimento da heurística, mas no caso do *Split Cache*, caso seja decidida a necessidade de reconfiguração apenas em uma das caches, então a as duas caches poderão ser reconfiguradas. Nesse caso, de acordo com o módulo de determinação da configuração, a outra cache que não precisaria da reconfiguração, poderá ou não ser utilizada para doação de espaços.

Na Figura 6 é apresentado um exemplo em que o módulo de decisão de reconfiguração opta por não haver a reconfiguração da arquitetura durante alguns *quanta*, ocorrendo a reconfiguração da arquitetura apenas ao final do $quantum_W$.

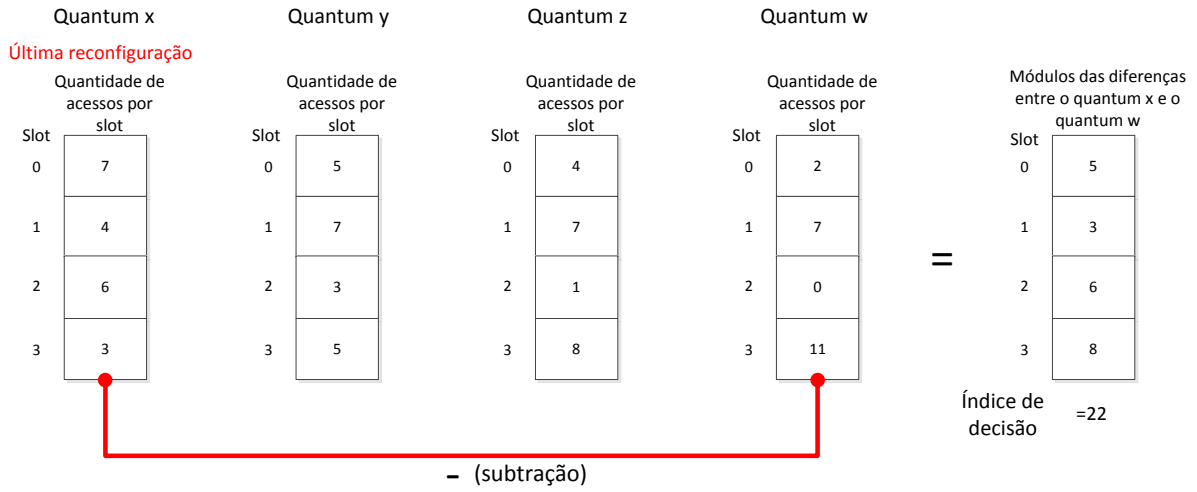


Figura 6: Verificação da necessidade de reconfiguração avaliando-se intervalos de tempo mais distantes

No exemplo da Figura 6 foi utilizada uma cache com 4 slots e um quantum com o tamanho de 20 acessos.

Na figura 6 é apresentada uma das situações que justifica o fato de se comparar o quantum atual, ao último quantum em que houve uma reconfiguração (ou o primeiro quantum da carga) e não ao quantum imediatamente anterior. Pois, com a Figura 6, é possível perceber que a mudança no comportamento da carga de trabalho foi lenta e só pôde ser percebida comparando-se intervalos de tempo mais distantes, comparando-se o $quantum_W$ com o $quantum_X$. No $quantum_Y$ e no $quantum_Z$ não houve reconfiguração, pois a mudança no comportamento da carga de trabalho não foi significativa.

Com o módulo de decisão de reconfiguração, a arquitetura, possivelmente, não será reconfigurada toda vez, resultando em um melhor desempenho devido a um menor tempo total de acesso e a altos índices de *cache hit*. Como a cache só será reconfigurada quando necessário, serão evitadas reconfigurações que poderão reduzir o desempenho da cache, penalizando slots que não precisariam ser penalizados, como no caso de pequenas mudanças no comportamento ou mudanças esporádicas.

Portanto, o principal objetivo do módulo de verificação da necessidade de reconfiguração é tentar encontrar os melhores momentos para que aconteça a reconfiguração, ou seja, tentar encontrar aqueles momentos em que a carga varia consideravelmente o seu comportamento, para que a arquitetura se reconfigure apenas nesses momentos e não em momentos pré-definidos.

3.2.4 Módulo de determinação da configuração

Ao final de cada quantum, se o módulo decisão de reconfiguração verificar a necessidade de haver a reconfiguração, então o módulo de determinação de configuração escolherá os melhores slots doadores e recebedores naquele momento.

A doação pode ocorrer entre slots da mesma cache ou entre slots das caches de dados e de instruções. Uma *cache* pode doar mesmo sem ter espaços vazios, para isso devem ser definidos quais *slots* doarão.

Os slots doadores devem ser escolhidos de acordo com o total de acessos

que cada slot está tendo (TA), número de cache misses ocorridos em cada slot (M) e a associatividade (A) que o slot possui no momento (quantidade de espaços), de acordo com a Equação 2.

$$índice\ de\ determinação = \frac{(TA \times M)}{A} \quad (2)$$

A equação 2 calcula o índice de determinação de reconfiguração de cada slot. Os slots que possuírem os menores índices de determinação serão os doadores. Com a utilização da equação 2, os principais parâmetros para se definir uma doação ou recebimento são ponderados com a utilização da multiplicação e divisão. Dessa forma, diferentemente dos trabalhos relacionados onde existem algumas faixas predeterminadas para classificar cada slot, os slots, por mais parecidos que sejam, sempre terão índices diferentes, sendo escolhido o que realmente necessita de mais espaço e não aleatoriamente como ocorre com os slots que ficam dentro da mesma faixa de valores nos trabalhos relacionados (Carvalho, 2005; Kerr Junior, 2008).

A equação 2 utiliza o número de cache misses de cada slot para verificar o desempenho desse slot na arquitetura, pois um slot com um número muito alto de cache misses estará prejudicando a taxa de cache hit da cache. O número total de acessos foi utilizado para comparar a representatividade de cada slot em relação aos outros slots da cache. E por último, o grau de associatividade do slot no final do quantum foi utilizado para se verificar a interferência da quantidade de espaços que o slot já possui no seu desempenho. Assim, no caso de um slot com um grande número de cache misses e total de acessos, ele não necessariamente irá receber um espaço, pois caso a sua associatividade já seja muito grande, provavelmente um novo espaço não conseguirá aumentar significativamente o seu desempenho. Mas, caso o seu espaço de armazenamento seja pequeno, a inserção de um novo espaço aumentará significativamente o seu espaço de armazenamento, aumentando o seu desempenho.

Inicialmente, o módulo de determinação da configuração calculará o índice de determinação para todos os slots das caches. Em seguida, são escolhidos os dois slots com os menores índices, conhecidos como slots doadores. Após a escolha dos slots doadores, os slots recebedores são escolhidos, sendo esses os dois slots com

os maiores índices. Por fim, cada um dos slots doadores cederá um espaço de armazenamento, que será recebido por cada um dos slots recebedores.

A Figura 7 apresenta um exemplo da utilização do índice de determinação de configuração para a escolha dos slots doadores e recebedores.

Cache de dados

Slot	M	TA	A	Índice de determinação
0	12	15	7	26
1	20	20	2	200
2	9	9	5	16
3	7	8	8	7

Cache de instruções

Slot	M	TA	A	Índice de determinação
0	6	17	3	34
1	8	21	2	84
2	2	9	3	6
3	1	1	2	1

Figura 7: Utilizacao do indice de indeterminacao, os slots 1 das duas caches receberão os espaços doados pelos slots 2 e 3 da cache de instruções.

No exemplo da Figura 7, os slots doadores são os slots com menores índices, sendo, respectivamamente, os slots 2 e 3 da cache de instruções. Por outro lado, os slots 1 de cada uma das caches foram escolhidos como recebedores, por terem os maiores índices de determinação. O slot 3 da cache de instruções teve o menor índice de determinação devido à pequena quantidade de acessos a esse slot. Já o slot 2 da cache de instruções teve o segundo menor índice de determinação devido ao fato de ter tido um pequeno número de cache misses proporcionamente a ao seu total de acessos. De outra forma, o slot 1 da cache de dados obteve o maior índice de determinação, o que pode ser explicado pelo grande número de cache misses em relação ao total de acessos a este slot e também por ter um pequeno número de espaços. Por fim, o slot 1 da cache de instruções teve o segundo maior índice de determinação, devido ao grande número de acessos e à pequena quantidade de espaços do slot.

O *slot* escolhido como recebedor terá a sua associatividade aumentada, recebendo um novo espaço no final do seu espaço de armazenamento. Dessa forma, durante os próximos acessos a esse slot, menor será a probabilidade dos blocos armazenados serem substituídos, pois haverá um maior espaço de armazenamento disponível, o que resultará em uma redução do cache miss por conflito ou por capacidade.

A quantidade de espaços a serem doados pela cache é um valor parametrizável, mas nessa proposta foi utilizado o valor de 2 espaços por reconfiguração. A associatividade do *slot* que for escolhido como doador será diminuida, assim esse *slot* perderá o último espaço de seu espaço de armazenamento.

Uma *cache* pode doar espaços até atingir o limite mínimo que é possuir um espaço para cada *slot*. Caso pudessem ser retirados todos os espaços de um determinado *slot*, esse *slot* teria que ter um índice relacionado ao mesmo ligando-o a outro *slot*, onde seriam armazenados os endereços futuros que pertenceriam a este, assim, uma busca em uma *cache* indexada desta forma poderia demandar mais tempo, reduzindo consideravelmente o desempenho da *cache*.

Cada *slot* possui uma associatividade máxima que deve ser parametrizada, pois uma associatividade máxima muito alta tornaria essa arquitetura inviável de se implementar em *hardware,* devido à complexidade do sistema de comparação de *tags*.

Em síntese, ao final de cada *quantum* é verificada a necessidade de reconfiguração. Caso o módulo de decisão de reconfiguração determine a necessidade de reconfiguração, o módulo de determinação de configuração escolherá os dois principais doadores e recebedores, sendo permitido todos serem ou não da mesma cache. Após a reconfiguração, a carga continua a ser executada e esses passos serão repetidos em cada quantum até o final da execução da carga de trabalho.

3.3 Considerações finais

Várias são as contribuições da arquitetura proposta em relação aos principais trabalhos relacionados. A primeira contribuição é a utilização do módulo de decisão de reconfiguração, que verifica a necessidade da reconfiguração ao final de cada quantum, evitando assim reconfigurações desnecessárias. A segunda contribuição é a utilização de uma política de reconfiguração diferente, baseada na ponderação do total de acessos, número de cache misses e grau de associatividade de cada slot, o que permite classificar individualmente cada slot, diferentemente da classificação por faixas, feita pelos trabalhos relacionados. E por último, outra contribuição é a possibilidade de doação de espaços intra e inter caches, diferentemente dos trabalhos relacionados, que permitem apenas a doação intra cache.

4 RESULTADOS

Neste capítulo primeiramente será apresentado o ambiente de experimentação utilizado para verificação dos resultados. No tópico seguinte serão apresentados os resultados de desempenho da arquitetura proposta, comparando-os com os desempenhos das arquiteturas convencionais de *Split Cache* para os mesmos traces. Por fim serão comparados os resultados obtidos com a arquitetura proposta e os resultados obtidos pelos trabalhos relacionados.

4.1 Ambiente de experimentação

Neste tópico será apresentado o ambiente experimental de verificação da arquitetura para obtenção dos resultados, sendo esse composto pelo simulador utilizado, pelas cargas de trabalho simuladas e pelos parâmetros utilizados para simulação da arquitetura.

4.1.1 Simulador de Split Cache Reconfigurável

As simulações foram realizadas e verificadas em um simulador, que tem como base os simuladores MSCSim (Coutinho; Mendes; Martins, 2006) e Web-MHE (Mendes; Coutinho; Martins, 2006). Esses simuladores foram desenvolvidos por nós nos últimos anos.

O simulador tem como propósito ser um simulador de memória cache capaz de simular e verificar diferentes configurações de Split Cache, sendo reconfiguráveis ou não.

Com este simulador é possível simular traces obtidos de cargas de trabalho sintéticas e reais (BYU, 2009; SPEC, 2009). Para converter os traces das cargas de trabalho do SPEC CPU2000 (SPEC, 2009) para o nosso simulador, foi utilizada a mesma ferramenta utilizada pelos trabalhos relacionados (Carvalho; Martins, 2004).

Esta ferramenta retira os campos desnecessários do trace, permanecendo apenas os campos de endereço de acesso e tipo de acesso, sendo dado ou instrução e leitura ou escrita.

O simulador foi desenvolvido em JAVA (J2SE - *Java Standard Edition*), por ser uma linguagem “livre”, robusta e independente de plataforma, que possibilita o desenvolvimento orientado por objetos.

Vários parâmetros, como capacidade de armazenamento da memória cache, número de slots, número de blocos, tamanho do bloco, política de substituição (FIFO e LRU), grau de associatividade, tamanho do quantum e possibilidade de reconfiguração, podem ser simulados no SSCR.

4.1.2 Traces utilizados

Para realizar a verificação funcional do desempenho da arquitetura de cache proposta foram realizados diversos testes utilizando traces reais obtidos do Brigham Young University Trace Distribution Center (BYU, 2009). Os traces escolhidos foram os mesmos utilizados pelos trabalhos relacionados (Carvalho, 2005; Kerr Junior, 2008) e foram obtidos utilizando os benchmarks SPEC CPU2000, tanto os benchmarks de inteiros como os de ponto flutuante. Esses traces escolhidos foram obtidos utilizando um processador Pentium III 733MHz com Windows 2000. Os traces escolhidos estão descritos no Quadro 3, juntamente com o seu tipo (inteiro - Int ou ponto flutuante - PF) e sua descrição.

Trace	Tipo	Descrição
applu	PF	Equações diferenciais
bzip2	Int	Compressão
crafty	Int	Xadrez
eon	Int	Visualização Computacional
gap	Int	Interpretador

gcc	Int	Compilador C
gzip	Int	Compressão
mcf	Int	Otimização Combinatoria
parser	Int	Processamento de texto
swim	PF	Fluidos
twolf	Int	Simulador place and route
vortex	Int	Banco de dados
vpr route	Int	Roteamento em FPGA
wupwise	PF	Quântica

Quadro 3: Traces utilizados nas simulações

4.1.3 Parâmetros gerais utilizados

As arquiteturas de cache utilizadas nas simulações são compostas por *Split Cache*. Foram simuladas memórias cache convencionais e reconfiguráveis, sendo aquela uma arquitetura tradicional/fixa, composta por duas caches com 50% de espaço de armazenamento para dados e 50% para instruções. Essa arquitetura convencional não permite a reconfiguração do seu espaço de armazenamento em nenhuma hipótese. Assim, ela não possui heurística de decisão de reconfiguração e nem heurística de determinação de configuração, tendo características comuns às memórias cache encontradas nas arquiteturas dos computadores atuais.

Para realizar as simulações e as comparações de desempenho foi considerada uma memória *Split Cache* com espaço de armazenamento total de 64KB. Cada uma das memórias cache (dados e instruções) utilizada foi composta inicialmente por um espaço de armazenamento de 32KB. Cada memória cache possui 256 slots, grau de associatividade 4 e tamanho do bloco igual a 32 bytes, sendo composto por 8 palavras de 32 bits. Nessas simulações não foi considerado o tempo gasto para a reconfiguração. Todos esses parâmetros utilizados foram os mesmos simulados pelos trabalhos relacionados, assim permitindo a comparação dos resultados.

A arquitetura proposta simulada foi parametrizada com o número de espaços a serem doados/recebidos por reconfiguração igual a 2, assim o módulo de determinação da configuração escolherá os 2 slots com os menores índices para doarem e os dois slots com os maiores índices para receberem. Também foi parametrizado o grau de associatividade máximo de cada slot em 32.

4.2 Resultados de simulação de diferentes instâncias da arquitetura proposta

Para verificar o desempenho da nossa arquitetura proposta foram feitas diversas simulações e comparações. Neste primeiro momento foram comparadas 3 instâncias da arquitetura proposta com a arquitetura convencional e com uma arquitetura que utiliza a reconfiguração em todos os *quanta*.

A arquitetura convencional é do tipo Associativa por Conjunto 4-way com os mesmos parâmetros gerais utilizados pela arquitetura proposta.

A primeira instância da arquitetura, denominada heur_ver_rec_1.000, utiliza a heurística do módulo de decisão de reconfiguração a cada 1.000 acessos, ou seja, possui um quantum com o tamanho de 1.000 acessos.

Já a segunda instância da arquitetura, denominada heur_ver_rec_20.000, utiliza a heurística do módulo de decisão de reconfiguração a cada 20.000 acessos, ou seja, possui um quantum com o tamanho de 20.000 acessos.

E a terceira instância da arquitetura, denominada heur_ver_rec_100.000, utiliza a heurística do módulo de decisão de reconfiguração a cada 100.000 acessos, ou seja, possui um quantum com o tamanho de 100.000 acessos.

Essas três instâncias foram simuladas para tentarmos verificar qual o melhor valor de quantum para a arquitetura proposta.

Também foi utilizada para comparação uma arquitetura reconfigurável, denominada Rec_sempre_20000, utilizando o módulo de determinação de configuração e o módulo de armazenamento reconfigurável propostos, mas sem a utilização do módulo de decisão de reconfiguração. O propósito da utilização dessa arquitetura Rec_sempre_20000 foi para verificar o ganho com a utilização da heurística de decisão de reconfiguração, que determina se a reconfiguração será ou não necessária no quantum. Também foram simuladas as arquiteturas que não

utilizam a heurística de decisão de reconfiguração (Rec_sempre) com os *quanta* de 1.000 e 100.000 acessos, e em todos os casos simulados a arquitetura que não utiliza heurística de decisão com quantum de 20.000 acessos obteve o melhor resultado dentre essas arquiteturas que reconfiguram em todos os *quanta*. Essas estatísticas não foram apresentadas, pois o propósito principal nesse caso é apresentar a importância do módulo da heurística de decisão de reconfiguração em comparação com uma arquitetura que não utiliza esse módulo. Por isso, essa arquitetura Rec_sempre_20000 utilizou um quantum de 20.000 acessos.

4.2.1 Trace APPLU

Conforme é apresentado na Figura 8, a cache convencional obteve 8,51% de cache miss de dados e 1,33% de cache miss de instruções ao executar o *trace* APPLU. O cache miss total que a cache convencional obteve para esse *trace* foi de 9,84%.

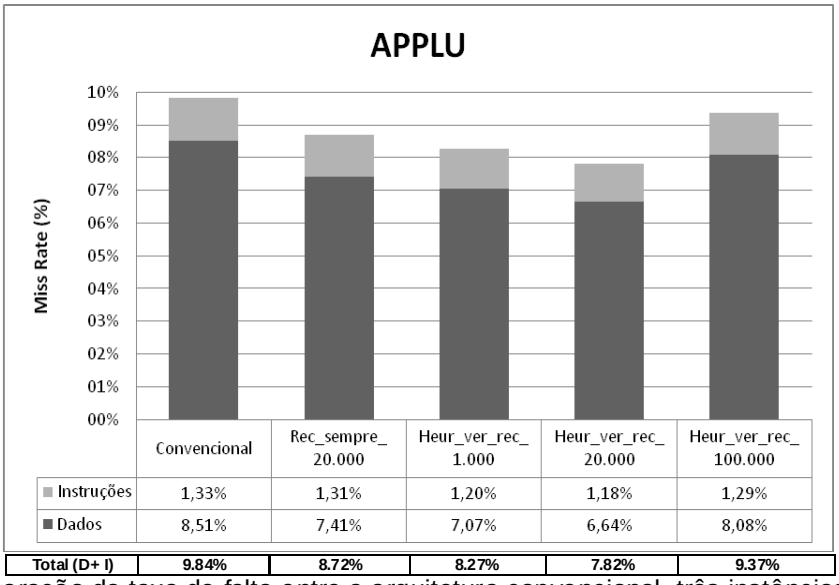


Figura 8: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace APPLU.

Para obter o valor do *quantum* que gera o melhor desempenho para a arquitetura proposta, essa arquitetura foi simulada com diversos valores de

quanta. Com um *quantum* de 1.000 acessos o valor da taxa de cache miss total foi de 8,27%, já com um *quantum* de 20.000 acessos o valor da taxa de cache miss total foi de 7,82%. E com o *quantum* de 100.000 acessos, o valor da taxa de cache miss total foi de 9,37%. O *quantum* de 20.000 acessos obteve uma melhoria de 5,44% em relação ao *quantum* de 1.000 acessos e 16,54% em relação ao *quantum* de 100.000 acessos. Essa melhoria no desempenho é relativa e foi calculada considerando o quociente da divisão do número de cache miss da arquitetura com o *quantum* de 20.000, sobre o número de cache miss da arquitetura com o *quantum* de 100.000, ou seja, todos os valores de melhoria considerados foram calculados utilizando o quociente da divisão da arquitetura com melhor desempenho sobre a arquitetura com pior desempenho.

A arquitetura que não utiliza a heurística de decisão de reconfiguração (Rec_sempre_20000) obteve uma taxa de cache miss de dados de 1,31% e de instruções de 7,41%, resultando em um valor da taxa de cache miss total de 8,72%. Já a arquitetura com a heurística de decisão verificando a reconfiguração a cada 20.000 (heur_ver_rec_20.000) obteve uma taxa de cache miss total de 7,82%. Dessa forma, a arquitetura heur_ver_rec_20.000 conseguiu uma melhoria de 10,32% em relação à arquitetura Rec_sempre_20000.

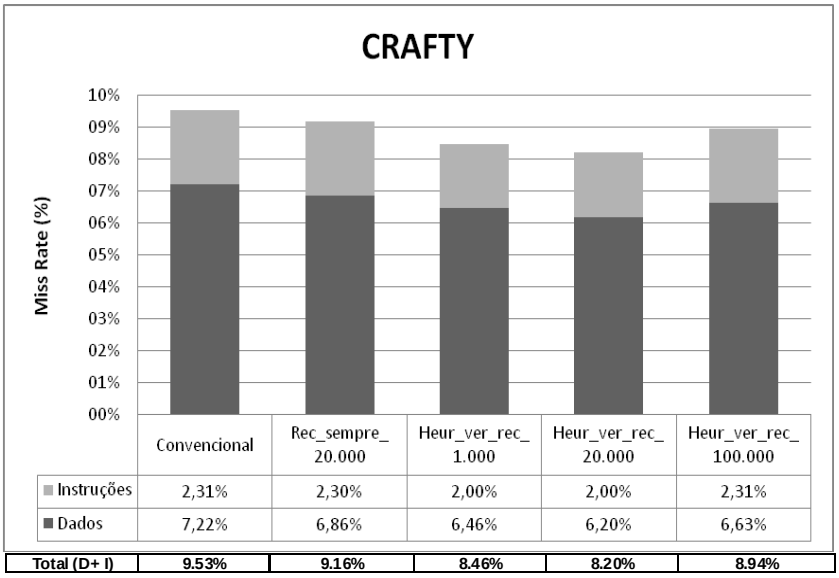
A arquitetura com o *quantum* de 100.000 obteve o pior desempenho comparado a todas as outras arquiteturas simuladas para essa carga, com exceção da convencional. Isso pode ser explicado, pois podendo reconfigurar apenas a cada 100.000 acessos a arquitetura não conseguiu se adaptar ao comportamento dessa carga com tanta eficiência como as outras arquiteturas que reconfiguraram mais vezes.

Para essa carga, com 10.352.349 acessos, a arquitetura proposta com utilização da heurística de decisão de reconfiguração a cada 20.000 acessos obteve a menor taxa de cache miss e reconfigurou 238 vezes, cerca de 46% das vezes que ele poderia reconfigurar caso reconfigurasse sempre a cada 20.000. A arquitetura não precisou se reconfigurar mais vezes, pois os acessos à cache, provavelmente, deveriam estar seguindo um padrão de comportamento em alguns momentos da execução da carga de trabalho, assim, sendo ideal a arquitetura permanecer com a sua configuração inalterada nessas situações, sem se reconfigurar. Além de ter tido a menor taxa de cache miss, a arquitetura proposta também conseguiria obter um menor tempo médio de acesso, caso tivesse sido considerado o tempo de

reconfiguração.

4.2.2 Trace CRAFTY

Conforme é apresentado na Figura 9, a arquitetura convencional obteve o pior desempenho de todas as arquiteturas, com 9,5267%, para o trace crafty.



	Convencional	Rec_sempre_ 20.000	Heur_ver_rec_ 1.000	Heur_ver_rec_ 20.000	Heur_ver_rec_ 100.000
Instruções	2,31%	2,30%	2,00%	2,00%	2,31%
Dados	7,22%	6,86%	6,46%	6,20%	6,63%
Total (D+ I)	**9.53%**	**9.16%**	**8.46%**	**8.20%**	**8.94%**

Figura 9: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace CRAFTY.

Para essa carga, as 3 arquiteturas com a heurística de decisão da reconfiguração (Heur_ver_rec_1.000, Heur_ver_rec_20.000 e Heur_ver_rec_100000) conseguiram reduzir a taxa de cache miss em comparação à arquitetura que não utiliza essa heurística (Rec_sempre_20.000), em 7.64%, 10.48% e 2.40%, respectivamente.

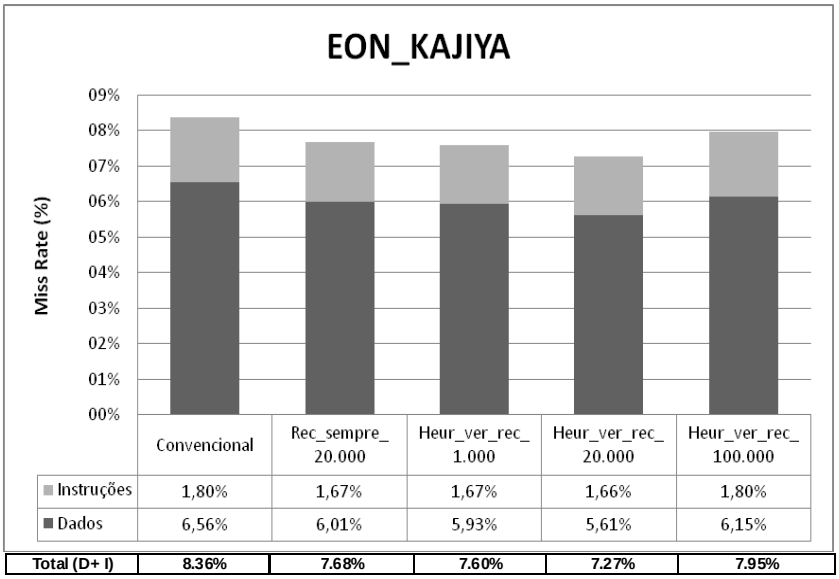
A arquitetura com a heurística de decisão da reconfiguração a cada 20.000 acessos (Heur_ver_rec_20.000) foi a arquitetura com melhor desempenho ao simular o trace crafty, com uma taxa de miss total de 8,20%. A taxa de miss total dessa arquitetura foi melhor do que a arquitetura convencional em 13,95%, a arquitetura Rec_sempre_20.000 em 10,48%, a arquitetura Heur_ver_rec _1.000 em 3,07% e a arquitetura Heur_ver_rec _100.000 em 8,27%. Dessa forma, a arquitetura

que verifica a necessidade de reconfiguração a cada 20.000 acessos (Heur_ver_rec_20.000) foi a que melhor se adaptou ao trace crafty.

Para o trace crafty, a arquitetura com a heurística de decisão da reconfiguração a cada 20.000 acessos reconfigurou em cerca de 53% das vezes que ela poderia reconfigurar.

4.2.3 Trace EON_KAJIYA

Em relação ao trace eon_kajiya, novamente a arquitetura que obteve a maior taxa de cache miss foi a arquitetura convencional. Essa arquitetura obteve uma taxa de cache miss total de 8,36%, de acordo com a Figura 10.



	Convencional	Rec_sempre_ 20.000	Heur_ver_rec_ 1.000	Heur_ver_rec_ 20.000	Heur_ver_rec_ 100.000
Instruções	1,80%	1,67%	1,67%	1,66%	1,80%
Dados	6,56%	6,01%	5,93%	5,61%	6,15%
Total (D+ I)	**8.36%**	**7.68%**	**7.60%**	**7.27%**	**7.95%**

Figura 10: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace EON_KAJIYA.

Para o trace eon_kajiya, a arquitetura que reconfigura sempre a cada 20.000 acessos (Rec_sempre_20000) obteve uma taxa de miss menor do que a arquitetura Heur_ver_rec_100.000 em 3,39%. Mas, as outras duas arquiteturas com a heurística de decisão de reconfiguração (Heur_ver_rec_1.000 e Heur_ver_rec_20.000) obtiveram uma taxa de miss menor em 1,04% e 5,33% em relação à arquitetura que não utiliza a heurística de decisão de reconfiguração, sendo essa a

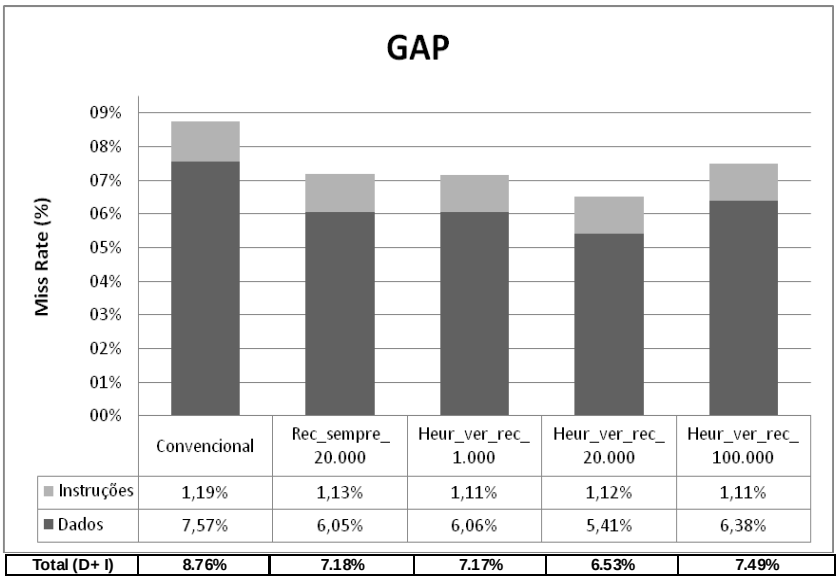
rec_sempre_20.000.

Portanto, a arquitetura que melhor conseguiu se adaptar à carga eon_kajiya foi a arquitetura reconfigurável com a heurística de decisão de reconfiguraçao a cada 20.000 acessos (Heur_ver_rec_20.000).

A arquitetura Heur_ver_rec _20.000 reconfigurou em cerca de 59% das vezes em que poderia haver a reconfiguração.

4.2.4 Trace GAP

De acordo com a Figura 11, para o trace gap, a arquitetura convencional obteve a maior taxa de cache miss total em comparação às outras quatro arquiteturas (Rec_sempre_20000, Heur_ver_rec_1.000, Heur_ver_rec_20.000 e Heur_ver_rec_100.000), sendo esta de 8,76%.



	Convencional	Rec_sempre_ 20.000	Heur_ver_rec_ 1.000	Heur_ver_rec_ 20.000	Heur_ver_rec_ 100.000
Instruções	1,19%	1,13%	1,11%	1,12%	1,11%
Dados	7,57%	6,05%	6,06%	5,41%	6,38%
Total (D+ I)	**8.76%**	**7.18%**	**7.17%**	**6.53%**	**7.49%**

Figura 11: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace GAP.

A arquitetura Rec_sempre_20000 obteve uma taxa de cache miss total 4,14% menor do que a arquitetura com a heurística de decisão de reconfiguração a cada 100.000 acessos (Heur_ver_rec_100.000). Mas as arquiteturas reconfiguráveis com quantum de 1.000 e 20.000 acessos (Heur_ver_rec _1.000 e Heur_ver_rec _20.000)

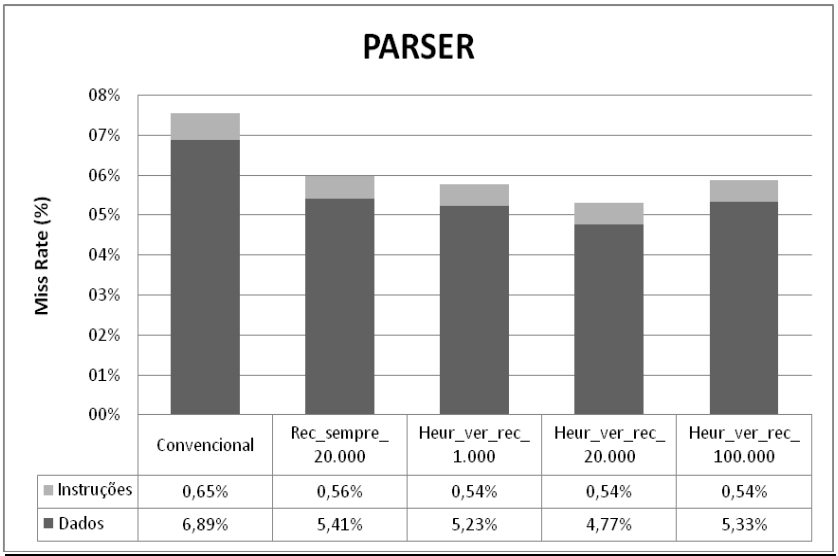
conseguiram melhorar a taxa de cache miss em 0,13% e 9,05%, respectivamente, em comparação à arquitetura que reconfigura sempre a cada 20.000 acessos (Rec_sempre_20000).

A arquitetura que melhor se adaptou ao trace gap foi a arquitetura reconfigurável que utiliza a heurística de decisão de reconfiguração a cada 20.000 acessos (Heur_ver_rec_20.000). Essa obteve uma melhora de 25,47% na taxa de cache miss total em relação ao Split Cache convencional (Convencional).

A arquitetura reconfigurável Heur_ver_rec_20.000 reconfigurou 263 vezes na simulação do trace gap, cerca de 51% das vezes em que poderia ocorrer a reconfiguração.

4.2.5 Trace PARSER

A arquitetura convencional obteve uma taxa de cache miss total de 7,5431% para o trace parser, de acordo com a Figura 12.



	Convencional	Rec_sempre_ 20.000	Heur_ver_rec_ 1.000	Heur_ver_rec_ 20.000	Heur_ver_rec_ 100.000
Instruções	0,65%	0,56%	0,54%	0,54%	0,54%
Dados	6,89%	5,41%	5,23%	4,77%	5,33%
Total (D+ I)	7.54%	5.97%	5.77%	5.31%	5.87%

Figura 12: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace PARSER.

Conforme é apresentado na Figura 12, a arquitetura que reconfigura sempre a cada 20.000 acessos (Rec_sempre_20000) obteve uma taxa de cache miss total

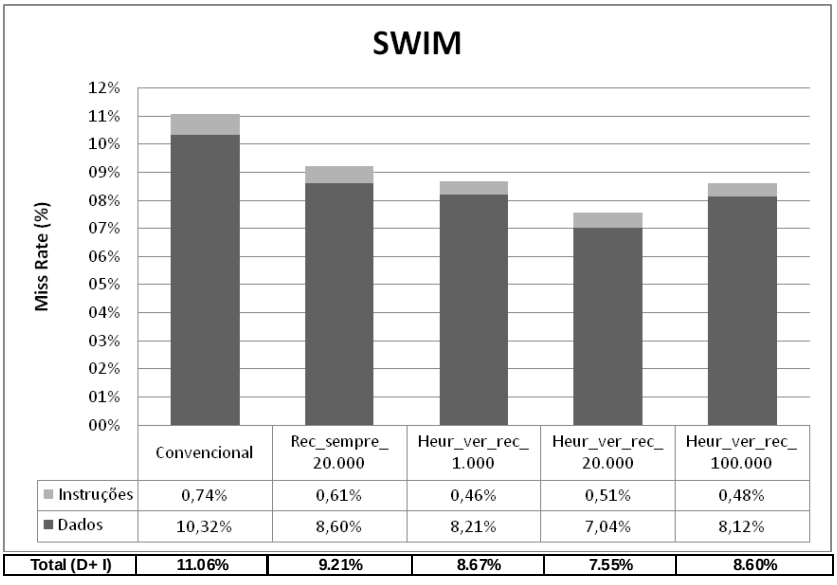
maior que as arquiteturas com a heurística de decisão de reconfiguração, Heur_ver_rec_1.000, Heur_ver_rec _20.000 e Heur_ver_rec _100.000, em 3,35%, 11,05% e 1,67%, respectivamente.

A arquitetura que melhor conseguiu se adaptar ao trace parser foi a arquitetura reconfigurável que utiliza a heurística de decisão de reconfiguração a cada 20.000 acessos. Essa arquitetura reduziu a taxa de cache miss total em 7.97%, 9.54% e 29.57%, em relação às arquiteturas com heurística de decisão de reconfiguração a cada 1.000 e 100.000 acessos e à arquitetura convencional, respectivamente.

A arquitetura com a menor taxa de cache miss total (Heur_ver_rec_20000) reconfigurou-se em 47% das vezes que poderia ser reconfigurada.

4.2.6 Trace SWIM

De acordo com a Figura 13, a arquitetura convencional obteve a maior taxa de cache miss total em comparação com todas as arquiteturas, com 11,06%, para o trace swim.



	Convencional	Rec_sempre_ 20.000	Heur_ver_rec_ 1.000	Heur_ver_rec_ 20.000	Heur_ver_rec_ 100.000
Instruções	0,74%	0,61%	0,46%	0,51%	0,48%
Dados	10,32%	8,60%	8,21%	7,04%	8,12%
Total (D+ I)	**11.06%**	**9.21%**	**8.67%**	**7.55%**	**8.60%**

Figura 13: Comparação da taxa de falta entre a arquitetura convencional, três instâncias da arquitetura proposta e esta sem a utilização da decisão de reconfiguração para o trace SWIM.

Para o trace swim, as 3 arquiteturas que utilizam a heurística de decisão de reconfiguração (Heur_ver_rec_1.000, Heur_ver_rec_20.000 e Heur_ver_rec_100.000) conseguiram uma taxa total de cache miss menor em comparação à arquitetura que não utiliza essa heurística (Rec_sempre_20.000). A arquitetura Heur_ver_rec_1.000 obteve uma taxa de cache miss de instruções menor que a arquitetura Heur_ver_rec_20.000, mas como a taxa de cache miss de dados nesse trace é mais representativa que a taxa de cache miss de instruções e a taxa de cache miss de dados da arquitetura Heur_ver_rec_20000 foi consideravelmente menor do que a da arquitetura Heur_ver_rec_1000, então o desempenho geral em relação à taxa de cache miss daquela foi melhor do que o desta em 12,92%.

A arquitetura com heurística de decisão de reconfiguração a cada 20.000 acessos foi a arquitetura com melhor desempenho ao simular o trace swim, com uma taxa de miss total de 7,55%. A taxa de miss total dessa arquitetura foi melhor do que a arquitetura Rec_sempre_20000 em 18,02%, a arquitetura Heur_ver_rec_1.000 em 12,92% e a arquitetura Heur_ver_rec_100.000 em 12,21%. Dessa forma, a arquitetura que utiliza a heurística decisão de reconfiguração a cada 20.000 acessos foi a que melhor se adaptou ao trace swim.

Para o trace swim, a melhor arquitetura (Heur_ver_rec_20.000) reconfigurou em cerca de 42% das vezes que ele poderia reconfigurar.

4.2.7 Análise das diferentes instâncias da arquitetura proposta e da eficiência do módulo de decisão de reconfiguração

Em todos os traces simulados, a arquitetura proposta com a heurística de decisão de reconfiguração com o quantum de 20.000 acessos (Heur_ver_rec_20.000) conseguiu obter a menor taxa de cache miss total, em relação às outras quatro arquiteturas simuladas. Isso pode ser verificado na Tabela 2.

Tabela 2: Taxas de cache miss obtidas pelas arquiteturas simuladas

Traces / Arquiteturas	Applu	Crafty	Eon	Gap	Parser	Swim
Convencional	9,84%	9,53%	8,36%	8,76%	7,54%	11,06%
Rec_Sempre_20.000	8,72%	9,16%	7,68%	7,18%	5,97%	9,21%
Heur_ver_rec_1.000	8,27%	8,46%	7,60%	7,17%	5,77%	8,67%
Heur_ver_rec_20.000	**7,82%**	**8,20%**	**7,27%**	**6,53%**	**5,31%**	**7,55%**
Heur_ver_rec_100.000	9,37%	8,94%	7,95%	7,49%	5,87%	8,60%

O trace Swim foi aquele, dentre todos os anteriormente analisados, em que a arquitetura com a heurística de decisão de reconfiguração a cada 20.000 acessos (Heur_ver_rec_20.000) obteve o melhor resultado em comparação com a arquitetura que não utilizou a heurística de decisão de reconfiguração (Rec_sempre_20.000), aquela conseguiu reduzir a taxa de cache miss total em 18,03% em relação à esta.

De todos os traces simulados, o trace Swim foi o trace em que ocorreu o menor número de reconfigurações. A arquitetura com a heurística de decisão de reconfiguração a cada 20.000 acessos se reconfigurou em apenas 42% das vezes em que ela poderia se reconfigurar, caso se reconfigurasse sempre a cada 20.000 acessos.

Essa melhoria na taxa de cache miss total se deve tanto a essa heurística de de decisão de reconfiguração, que optou por reconfigurar menos vezes, quanto à heurística de determinação da configuração, que provavelmente fez melhores escolhas de slots doadores e recebedores.

O trace em que a arquitetura proposta com a heurística de decisão de reconfiguração a cada 20.000 acessos (Heur_ver_rec_20.000) obteve o menor ganho na redução da taxa de cache miss em comparação com a arquitetura que não utiliza essa heurística (rec_sempre_20.000) foi o eon_kajiya. Nesse caso obteve uma melhoria de 5,28% em relação à taxa de cache miss total. Nesse trace, a arquitetura Heur_ver_rec_20.000 se reconfigurou em cerca de 59% das vezes em que poderia haver a reconfiguração. Esse foi o trace em que a arquitetura se reconfigurou mais vezes. Isso provavelmente pode explicar o porque da melhoria no desempenho não ter sido tão considerável quanto a dos outros traces, pois o fato de ter se reconfigurado mais vezes pode ter variado muito o comportamento em que a arquitetura vinha tentando se manter depois de ter atingido uma configuração “ideal”

após se reconfigurar algumas vezes. Ou seja, o fato de se reconfigurar muitas vezes pode penalizar muito alguns slots que não vinham sendo muito utilizados, mas que provavelmente serão num futuro próximo.

4.3 Comparações com os principais trabalhos relacionados

Para comprovar a eficiência da arquitetura proposta foram feitas diversas simulações e os resultados comparados aos obtidos pelos principais trabalhos relacionados Kerr_Midorikawa (Kerr Junior, 2008) e Carvalho_Martins (Carvalho, 2005). Para realizar a comparação com os trabalhos relacionados, a arquitetura proposta foi simulada com os mesmos traces e com os mesmos parâmetros (tamanho total da cache, tamanho do bloco, número de slots e associatividade inicial) utilizados pelos trabalhos relacionados.

Os resultados alcançados pelas arquiteturas de Kerr_Midorikawa e Carvalho_Martins foram obtidos da dissertação de Kerr Junior (2008), onde kerr_Midorikawa simulam a sua própria arquitetura e a arquitetura de Carvalho_Martins, com os mesmos traces e os mesmos parâmetros utilizados pela arquitetura proposta neste trabalho. E com o objetivo de reduzir um pouco mais a taxa de cache miss total foi proposta por Kerr_Midorikawa (somente através de gráficos) uma mistura dos dois resultados da arquitetura de kerr_Midorikawa e da arquitetura de Carvalho_Martins, utilizando as menores taxas de cache miss de cada cache de cada arquitetura, intitulado Mixed Carvalho_Martins&Kerr_Midorikawa.

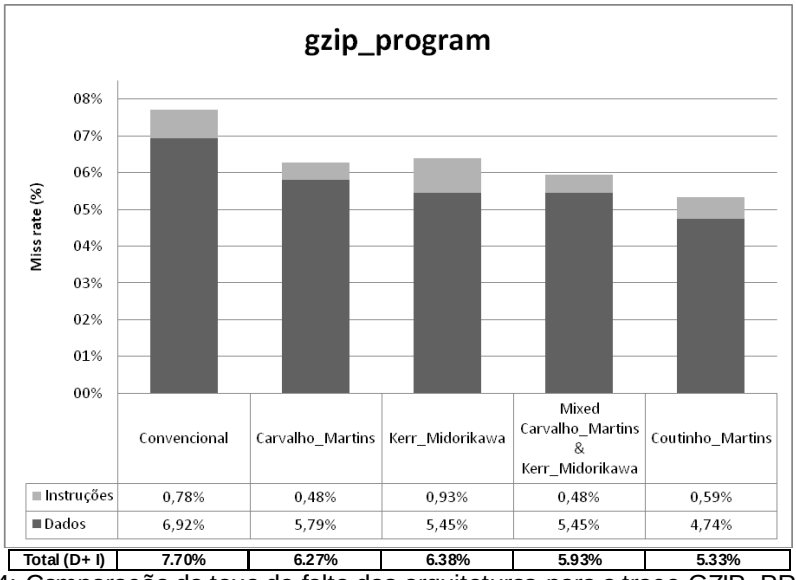
Em um primeiro momento, a arquitetura proposta neste trabalho foi configurada de forma fixa (não reconfigurável/convencional) com os mesmos parâmetros utilizados pela arquitetura fixa convencional utilizada por Kerr_Midorikawa e Carvalho_Martins para cada um dos traces e foram obtidos os mesmos resultados de taxa de cache miss, assim comprovando que os parâmetros utilizados pelas 3 arquiteturas são os mesmos.

As simulações foram realizadas utilizando a instância da arquitetura proposta que obteve o melhor desempenho para os traces já simulados no tópico anterior (heurística de decisão de reconfiguração a cada 20.000 acessos – Heur_ver_rec_20.000). Essa instância será conhecida nesse subtópico como Coutinho_Martins.

4.3.1 Traces GZIP

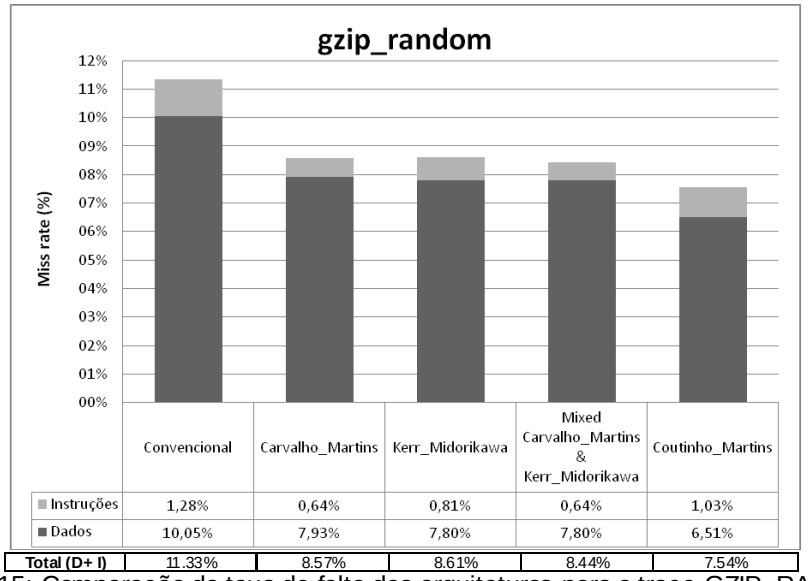
Simulando os traces gzip (gzip_program, gzip_random e gzip_source), a arquitetura convencional obteve as piores taxas de cache miss total em comparação com a arquitetura proposta e as arquiteturas relacionadas.

Nas Figuras 14, 15 e 16 serão apresentadas as taxas de cache miss (miss rate), considerando os traces GZIP, para as caches de dados e de instruções e as taxas de cache miss total para as arquiteturas: convencional, Carvalho_Martins, Kerr_Midorikawa, Mixed Carvalho_Martins&Kerr_Midorikawa e a proposta Coutinho_Martins.



	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	0,78%	0,48%	0,93%	0,48%	0,59%
Dados	6,92%	5,79%	5,45%	5,45%	4,74%
Total (D+ I)	**7.70%**	**6.27%**	**6.38%**	**5.93%**	**5.33%**

Figura 14: Comparação da taxa de falta das arquiteturas para o trace GZIP_PROGRAM.



	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	1,28%	0,64%	0,81%	0,64%	1,03%
Dados	10,05%	7,93%	7,80%	7,80%	6,51%
Total (D+ I)	**11.33%**	**8.57%**	**8.61%**	**8.44%**	**7.54%**

Figura 15: Comparação da taxa de falta das arquiteturas para o trace GZIP_RANDOM.

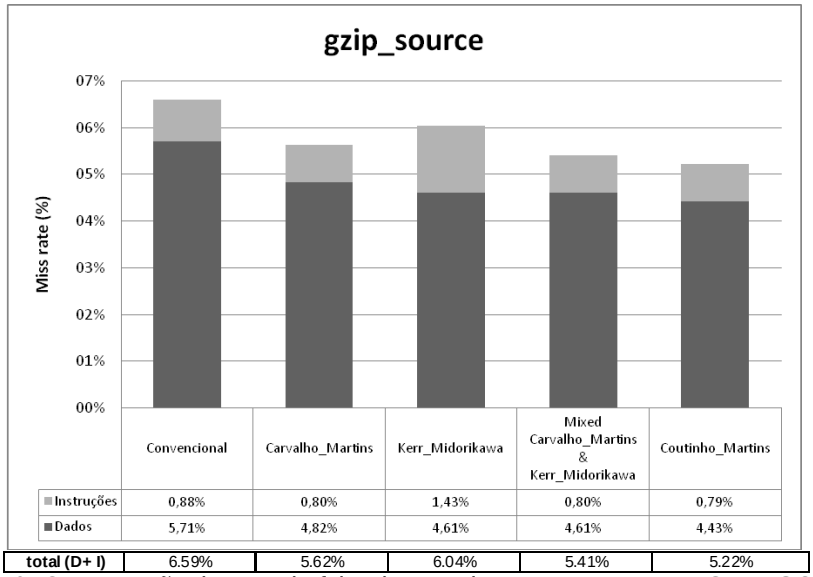


Figura 16: Comparação da taxa de falta das arquiteturas para o trace GZIP_SOURCE.

Em todos os traces GZIP simulados a arquitetura Carvalho_Martins obteve uma taxa de cache miss total menor do que as arquiteturas Convencional e Kerr_Midorikawa. Isso ocorreu devido à alta taxa de cache miss de instruções que a arquitetura de Kerr_Midorikawa obteve e que não conseguiu ser compensada com uma taxa mais baixa de cache miss de dados. Com o objetivo de melhorar essa taxa de cache miss total foi proposta por Kerr_Midorikawa (somente através de gráficos) uma mistura dos dois resultados das duas arquiteturas, utilizando as melhores taxas de cache miss, ou seja, a taxa de cache miss de instruções da arquitetura (Carvalho_Martins) e a taxa de cache miss de dados da arquitetura (Kerr_Midorikawa).

A taxa de melhoria/redução calculada nos resultados é relativa e foi obtida considerando o quociente da divisão do número de cache miss da arquitetura com melhor desempenho sobre a arquitetura com pior desempenho.

A arquitetura proposta (Coutinho_Martins), devido à utilização da heurística de decisão de reconfiguração e da heurística de determinação de configuração diferentemente das arquiteturas propostas, conseguiu uma redução na taxa de cache miss total em comparação com as outras arquiteturas relacionadas. Para o trace gzip_program, a arquitetura proposta conseguiu uma redução de 16,46% em relação à arquitetura de Kerr_Midorikawa, 14,99% em relação à arquitetura de Carvalho_Martins e 10,12% em relação à mistura das duas arquiteturas (Mixed Carvalho_Martins&Kerr_Midorikawa). Para o trace gzip_random, a arquitetura

proposta conseguiu uma redução de 12,43% em relação à arquitetura de Kerr_Midorikawa, 12,02% em relação à arquitetura de Carvalho_Martins e 10,66% em relação à Mixed Carvalho_Martins&Kerr_Midorikawa. E para o trace gzip_source a arquitetura proposta conseguiu uma redução de 13,58% em relação à arquitetura de Kerr_Midorikawa, 7,12% em relação à arquitetura de Carvalho_Martins e 3,51% Mixed Carvalho_Martins&Kerr_Midorikawa.

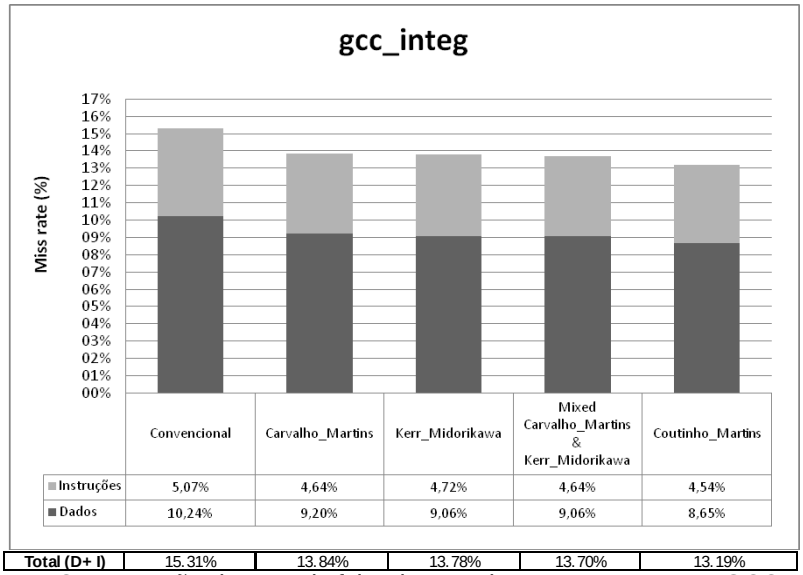
Apesar de que, para os traces gzip_program e gzip_random, a arquitetura proposta obteve uma taxa de cache miss de instruções maior que os trabalhos relacionados, a arquitetura proposta conseguiu compensar essa maior taxa de cache miss de instruções com uma maior redução na taxa de cache miss de dados, que nesses traces as reduções foram mais significativas do que a piora nas taxas de misses de instruções.

Para essas três cargas de trabalho a arquitetura proposta reconfigurou em cerca de 49%, 44% e 52%, respectivamente para gzip_program, gzip_random e gzip_source, do número total de possíveis reconfigurações.

Sendo assim, para os traces GZIP executados, a arquitetura proposta (Coutinho_Martins) conseguiu um melhor desempenho em comparação às arquiteturas relacionadas. Isso ocorreu devido à heurística de determinação de configuração que pondera os slots para doarem ou receberem espaços, e não os coloca em faixas pré definidas para selecioná-los. Também ocorreu devido à possibilidade de doação de espaços entre a cache de dados e a cache de instruções e também, principalmente, devido à existência da heurística de decisão de reconfiguração, que determina os momentos em que é necessária a doação para que não haja piora no desempenho.

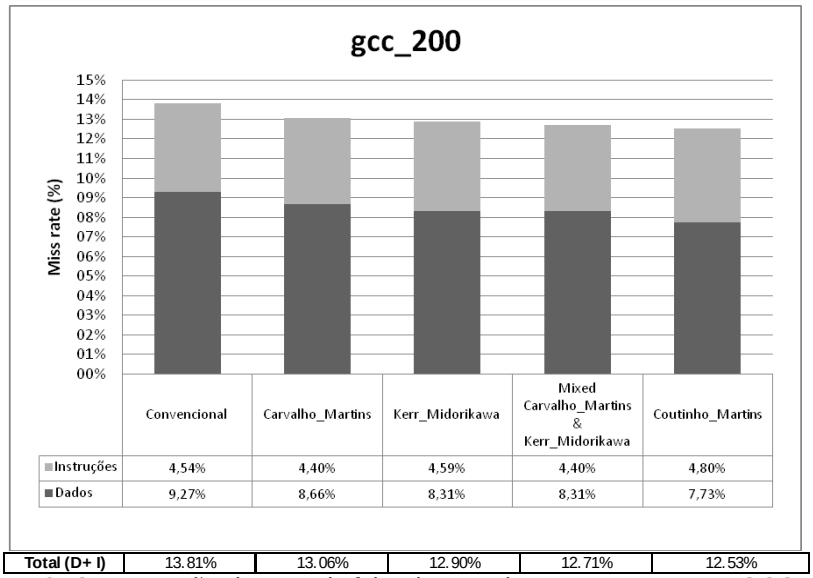
4.3.2 Traces GCC

Nas Figuras 17, 18, 19 e 20 serão apresentadas as taxas de cache miss (miss rate), considerando os traces GCC, para as caches de dados e de instruções e as taxas de cache miss total para as arquiteturas: convencional, Carvalho_Martins, Kerr_Midorikawa, Mixed Carvalho_Martins&Kerr_Midorikawa e a proposta Coutinho_Martins.



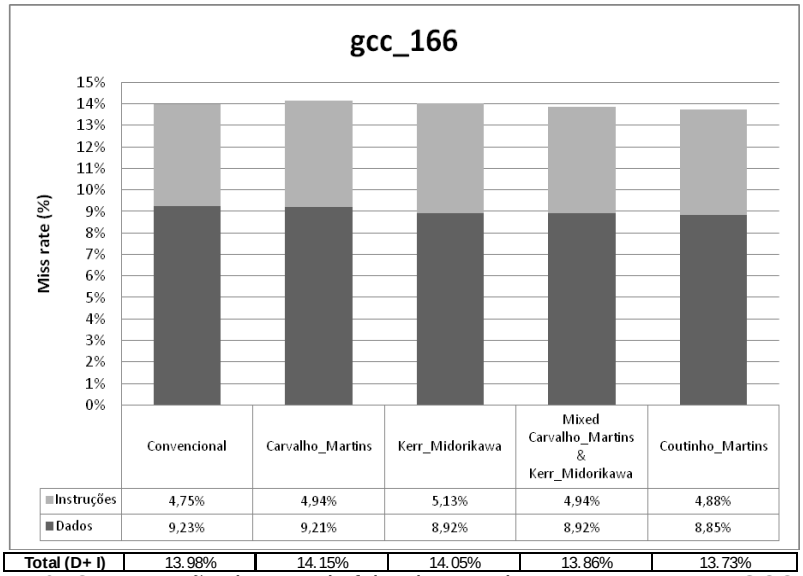
	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	5,07%	4,64%	4,72%	4,64%	4,54%
Dados	10,24%	9,20%	9,06%	9,06%	8,65%
Total (D+ I)	15.31%	13.84%	13.78%	13.70%	13.19%

Figura 17: Comparação da taxa de falta das arquiteturas para o trace GCC_INTEG.



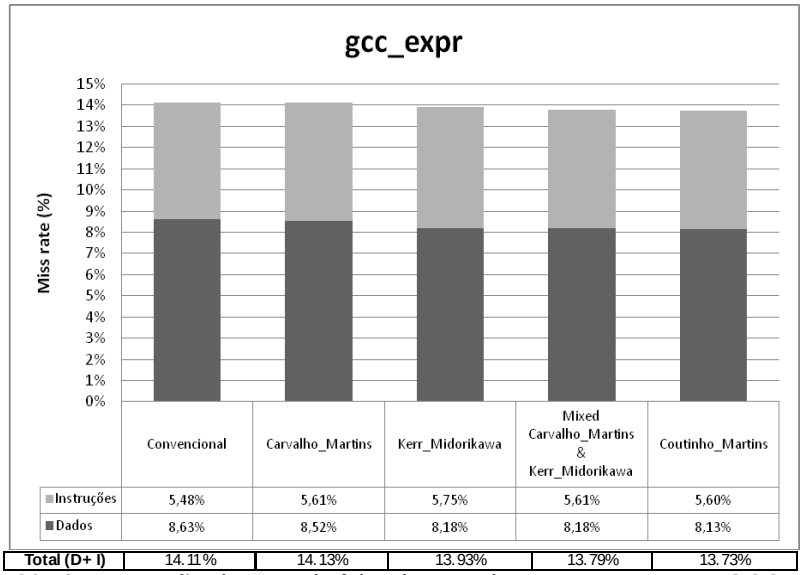
	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	4,54%	4,40%	4,59%	4,40%	4,80%
Dados	9,27%	8,66%	8,31%	8,31%	7,73%
Total (D+ I)	13.81%	13.06%	12.90%	12.71%	12.53%

Figura 18: Comparação da taxa de falta das arquiteturas para o trace GCC_200.



	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	4,75%	4,94%	5,13%	4,94%	4,88%
Dados	9,23%	9,21%	8,92%	8,92%	8,85%
Total (D+ I)	13.98%	14.15%	14.05%	13.86%	13.73%

Figura 19: Comparação da taxa de falta das arquiteturas para o trace GCC_166.



	Convencional	Carvalho_Martins	Kerr_Midorikawa	Mixed Carvalho_Martins & Kerr_Midorikawa	Coutinho_Martins
Instruções	5,48%	5,61%	5,75%	5,61%	5,60%
Dados	8,63%	8,52%	8,18%	8,18%	8,13%
Total (D+ I)	14.11%	14.13%	13.93%	13.79%	13.73%

Figura 20: Comparação da taxa de falta das arquiteturas para o trace GCC_EXPR.

A arquitetura convencional obteve as piores taxas de cache miss total em comparação às arquiteturas relacionadas e proposta para os traces gcc, com exceção dos traces gcc_expr e gcc_166. A arquitetura de Carvalho_Martins teve uma melhora na taxa de cache miss total em relação à arquitetura convencional nos traces gcc_integ e gcc_200. Para os traces gcc, a arquitetura de Kerr_Midorikawa obteve uma taxa de cache miss total menor do que Carvalho_Martins. Isso ocorreu devido a uma maior redução na taxa de cache miss de dados, pois apesar de possuir uma taxa de cache miss de instruções maior que Carvalho_Martins, a arquitetura de Kerr_Midorikawa conseguiu uma redução na taxa de cache miss de dados mais significativa do que o aumento na taxa de cache miss de instruções.

A arquitetura proposta (Coutinho_Martins) conseguiu reduzir a taxa de cache miss total em comparação com as arquiteturas relacionadas e convencional. Para o trace gcc_integ, a arquitetura proposta conseguiu uma redução de 4,28% em relação à arquitetura de Kerr_Midorikawa, 4,70% em relação à arquitetura de Carvalho_Martins e 3,72% em relação à mistura das duas arquiteturas. Para o trace gcc_200, a arquitetura proposta conseguiu uma redução de 2,87% em relação à arquitetura de Kerr_Midorikawa, 4,06% em relação à arquitetura de Carvalho_Martins e 1,42% em relação à mistura das duas arquiteturas. Para o trace gcc_166 a arquitetura proposta conseguiu uma redução de 2,28% em relação à arquitetura de Kerr_Midorikawa, 2,97% em relação à arquitetura de Carvalho_Martins e 0.94% em relação à mistura das duas arquiteturas. E para o

trace gcc_expr a arquitetura proposta conseguiu uma redução de 1,44% em relação à arquitetura de Kerr_Midorikawa, 2,83% em relação à arquitetura de Carvalho_Martins e 0.44% em relação à mistura das duas arquiteturas.

Apesar de que para o trace gcc_200 a arquitetura proposta obteve uma taxa de cache miss de instruções maior que os trabalhos relacionados, essa arquitetura conseguiu compensar essa piora na taxa de cache miss de instruções com uma melhora na taxa de cache miss de dados mais significativa do que a piora na taxa de miss de instruções.

Devido à heurística de decisão de reconfiguração, os traces gcc_integ, gcc_200, gcc_166 e gcc_expr executados reconfiguraram em cerca de 47%, 49%, 59% e 50%, do número total de possíveis reconfigurações.

Dessa forma, para os traces gcc executados, a arquitetura proposta (Coutinho_Martins) conseguiu um melhor desempenho em comparação às arquiteturas relacionadas, devido à heurística de determinação de configuração; à possibilidade de doação de espaços entre a cache de dados e a cache de instruções e também, principalmente, devido à heurística de decisão de reconfiguração, que determina os momentos em que é necessária a doação, evitando, assim, uma piora no desempenho.

4.3.3 Traces BZIP2

Nas Figuras 21, 22 e 23 são apresentadas as taxas de cache miss (miss rate), considerando os traces BZIP, para as caches de dados e de instruções e as taxas de cache miss total para as arquiteturas: convencional, Carvalho_Martins, Kerr_Midorikawa, Mixed Carvalho_Martins&Kerr_Midorikawa e a proposta Coutinho_Martins.

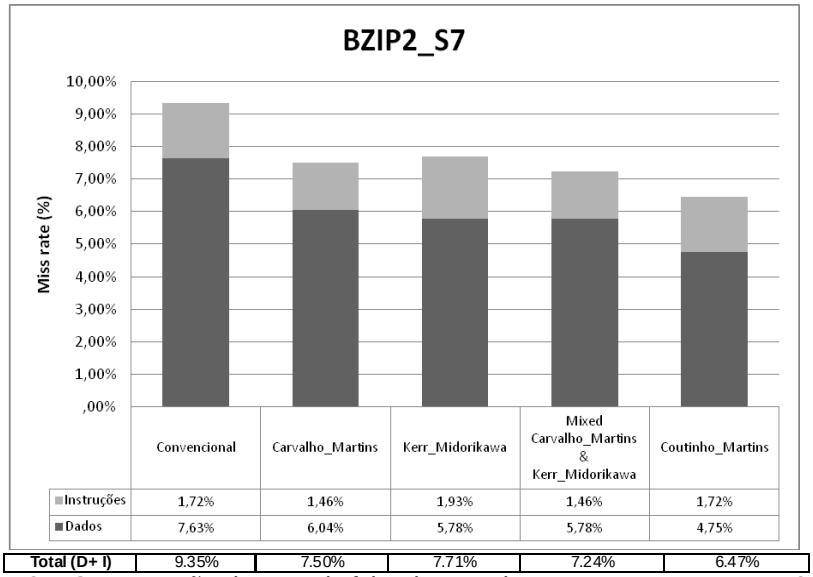


Figura 21: Comparação da taxa de falta das arquiteturas para o trace BZIP2_S7.

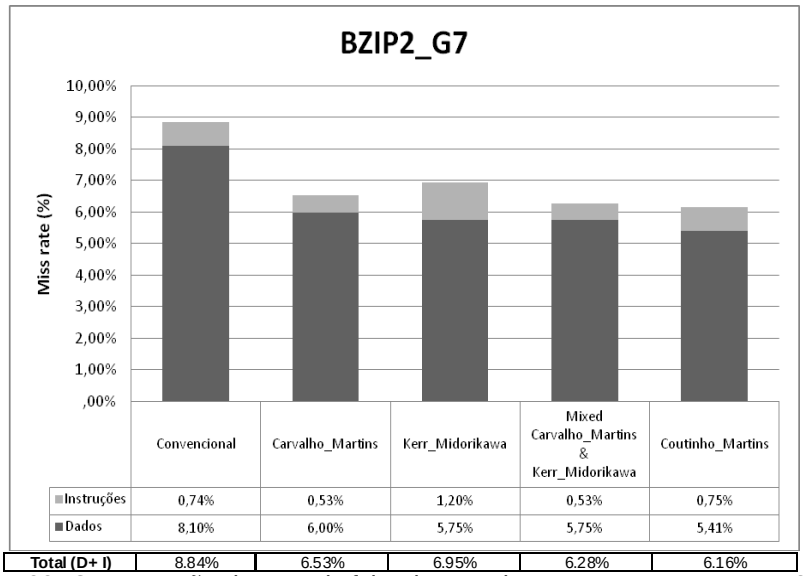


Figura 22: Comparação da taxa de falta das arquiteturas para o trace BZIP2_G7.

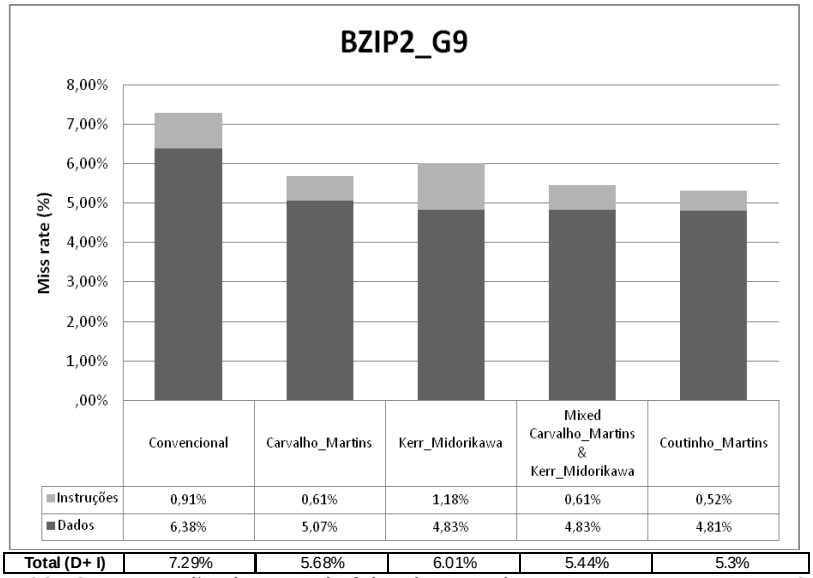


Figura 23: Comparação da taxa de falta das arquiteturas para o trace BZIP2_G9.

Para os traces bzip2, a arquitetura convencional obteve as piores taxas de cache miss total em comparação com a arquitetura proposta e as arquiteturas dos trabalhos relacionadas. A arquitetura de Carvalho_Martins teve uma melhora na taxa de cache miss total em relação à Kerr_Midorikawa. Isso ocorreu devido à alta taxa de cache miss de instruções que a arquitetura de Kerr_Midorikawa obteve e que não conseguiu ser compensada com uma taxa mais baixa de cache miss de dados.

A arquitetura proposta (Coutinho_Martins) conseguiu uma taxa de cache miss total menor em comparação com as arquiteturas dos trabalhos relacionados. Para o trace bzip2_s7, a arquitetura proposta conseguiu uma redução de 16,08% em relação à arquitetura de Kerr_Midorikawa, 13,73% em relação à arquitetura de Carvalho_Martins e 10,64% em relação à mistura das duas arquiteturas. Para o trace bzip2_g7, a arquitetura proposta conseguiu uma redução de 11,37% em relação à arquitetura de Kerr_Midorikawa, 5,67% em relação à arquitetura de Carvalho_Martins e 1,91% em relação à mistura das duas arquiteturas. E para o trace bzip2_g9, a arquitetura proposta conseguiu uma redução de 11,31% em relação à arquitetura de Kerr_Midorikawa, 6,16% em relação à arquitetura de Carvalho_Martins e 2,02% em relação à mistura das duas arquiteturas.

Apesar de que, para os traces bzip2_s7 e bzip2_g7, a arquitetura proposta obteve uma taxa de cache miss de instruções maior que o trabalho relacionado Carvalho_Martins, a arquitetura proposta conseguiu compensar essa maior taxa de cache miss de instruções com uma maior redução na taxa de cache miss de dados, que nesses traces as reduções foram mais significativas do que a piora nas taxas de misses de instruções.

Devido à heurística de decisão de reconfiguração, os traces bzip2_s7, bzip_g7e bzip_g9 executados reconfiguraram em cerca de 45%, 54% e 51%, do número total de possíveis reconfigurações.

Sendo assim, para os traces bzip2 executados, a arquitetura proposta (Coutinho_Martins) conseguiu um melhor desempenho em comparação às arquiteturas relacionadas. Isso se deve à eficiência dos principais módulos da arquitetura proposta, módulo de decisão de reconfiguração e módulo de determinação de configuração e a possibilidade de doação de espaços entre a cache de dados e a cache de instruções.

4.3.4 Análise dos resultados obtidos com a comparação entre arquitetura proposta e os principais trabalhos relacionados

Em todos os traces simulados, a arquitetura proposta (Coutinho_Martins) conseguiu obter a menor taxa de cache miss total, em relação à arquitetura convencional e aos trabalhos relacionados Carvalho_Martins, Kerr_Midorikawa e Mixed Carvalho_Martins&Kerr_Midorikawa. Isso pode ser verificado na Tabela 3.

Tabela 3: Taxas de cache miss total das arquiteturas para os traces simulados

Traces / Arquiteturas	Gzip_ program	Gzip_ random	Gzip_ source	Gcc_ integ	Gcc_ 200	Gcc_ 166	Gcc_ expr	Bzip2_ s7	Bzip2_ g7	Bzip2_ g9
Convencional	7,70%	11,33%	6,59%	15,31%	13,81%	13,98%	14,11%	9,35%	8,84%	7,29%
Carvalho_Martins	6,27%	8,57%	5,62%	13,84%	13,06%	14,15%	14,13%	7,50%	6,53%	5,68%
Kerr_Midorikawa	6,38%	8,61%	6,04%	13,78%	12,90%	14,05%	13,93%	7,71%	6,95%	6,01%
Mixed_Carvalho Kerr	5,93%	8,44%	5,41%	13,70%	12,71%	13,86%	13,79%	7,24%	6,28%	5,44%
Coutinho_Martins	5,33%	7,54%	5,22%	13,19%	12,53%	13,73%	13,73%	6,47	6,16%	5,33%

Em relação a todos os traces simulados, no trace Gzip_random a arquitetura proposta conseguiu obter a maior redução na taxa de cache miss total em relação a arquitetura Mixed Carvalho_Martins&Kerr_Midorikawa (união das melhores taxas de cache miss das duas arquiteturas), essa redução foi de 10,66%.

Para o trace Gzip_random, a arquitetura proposta obteve o menor número de reconfigurações durante a execução da carga, reconfigurando em aproximadamente 44% das vezes.

Já para o trace gcc_expr, a arquitetura proposta conseguiu obter a menor redução na taxa de cache miss total em relação à arquitetura Mixed Carvalho_Martins&Kerr_Midorikawa, sendo essa redução de 0,44%.

No trace gcc_expr, a arquitetura proposta se reconfigurou em aproximadamente 50% do número total de possíveis reconfigurações. Esse foi o segundo maior número de reconfigurações ocorridas nos traces simulados.

O trace em que a arquitetura proposta se reconfigurou mais vezes foi o trace gcc_166, no qual a arquitetura se reconfigurou em 59% das vezes. Esse trace

gcc_166 foi o *trace* em que a arquitetura proposta obteve a segunda menor redução na taxa de cache miss total em relação à arquitetura Mixed Carvalho_Martins&Kerr_Midorikawa, sendo essa redução de 0,94%

4.4 Conclusões obtidas com a análise dos resultados

A partir dos resultados é possível observar que a heurística de decisão de reconfiguração a cada 20.000 acessos obteve o melhor resultado, ou seja, a menor taxa de cache miss total, para todos os *traces* simulados, dentre as três diferentes configurações de *quanta* executadas.

Com o resultado das simulações também é possível perceber que a arquitetura que não utilizou o módulo de decisão de reconfiguração (Rec_sempre_20.000) conseguiu um desempenho inferior para todos os *traces* simulados, comparando-o com a arquitetura que utiliza o módulo de decisão de reconfiguração com o *quantum* de 20.000 acessos. Assim, comprovando o ganho na utilização do módulo de decisão de reconfiguração.

De acordo com os resultados das comparações de desempenho entre a arquitetura proposta e as arquiteturas dos principais trabalhos relacionados é possível perceber que, devido à utilização dos módulos de determinação de configuração e de decisão de reconfiguração, a arquitetura proposta obteve uma significativa redução na taxa de cache miss total em comparação às arquiteturas relacionadas. Apesar de a arquitetura proposta ter obtido uma taxa de cache miss de instruções maior do que os trabalhos relacionados, para alguns *traces*, a arquitetura proposta conseguiu uma redução mais significativa na taxa de cache miss de dados, assim conseguindo uma redução considerável na taxa de cache miss total.

A arquitetura proposta também apresenta uma melhoria no desempenho devido à possibilidade de se reconfigurar menos vezes, ou seja, de não se reconfigurar obrigatoriamente todas as vezes em todos os *quanta*. Dessa forma, ao optar por não se reconfigurar ao final de um determinado *quantum*, o *slot* que doaria os espaços não será mais penalizado e provavelmente, de acordo com os resultados, esses espaços serão bastante necessários para aquele *slot* no próximo *quantum*.

Assim, com a utilização do módulo de decisão de reconfiguração, o tempo total de acesso também pode ser reduzido, caso seja considerado o tempo gasto para a reconfiguração, o que não foi considerado nesse trabalho.

A melhoria no desempenho também se deve à heurística do módulo de determinação de configuração, que pondera todos os *slots* para avaliar os melhores para receber e para doar, pois, provavelmente, com essa heurística a arquitetura proposta deve ter feito melhores escolhas de *slots* doadores e recebedores. E essa melhoria no desempenho também se deve à possibilidade de doação intra e inter caches, ou seja, dentro da mesma cache ou entre a cache de dados e a cache de instruções. Em todas as simulações, a arquitetura proposta reconfigurou em cerca de 40% a 60% das vezes nas quais poderia haver a reconfiguração.

5 CONCLUSÕES

Neste capítulo será apresentada a discussão dos resultados obtidos com a proposta da arquitetura de um Split Cache Reconfigurável e as comparações de desempenho realizadas. Em seguida serão apresentadas as principais contribuições com a proposta da arquitetura. E por último serão apresentados alguns dos possíveis trabalhos futuros relacionados com a arquitetura proposta.

5.1 Discussão dos resultados

A arquitetura do *Split Cache* reconfigurável proposta é composta pelos módulos: módulo de armazenamento reconfigurável, módulo de decisão de reconfiguração e módulo de determinação de configuração. O módulo de armazenamento reconfigurável permite a reconfiguração do grau de associatividade de cada slot, por meio da doação e do recebimento de espaços intra ou inter caches. O módulo de decisão de reconfiguração tem como finalidade verificar a necessidade de reconfiguração da arquitetura em cada quantum. O módulo de determinação da configuração tem como funcionalidade a escolha dos melhores slots doadores e recebedores a cada quantum.

A arquitetura proposta se diferencia dos trabalhos relacionados em vários aspectos, como: existência de uma heurística de decisão de reconfiguração ao final de cada quantum, que verifica a necessidade da reconfiguração; existência de uma heurística de determinação da configuração da cache, que permite que cada slot seja classificado individualmente para a escolha dos doadores e recebedores, diferentemente da classificação por faixas utilizada pelos trabalhos relacionados; e a possibilidade de haver doação de espaços entre as duas caches (cache de dados e cache de instruções), doação inter caches, pois nos trabalhos relacionados só é permitida a doação intra caches, mas devido à grande variabilidade nos acessos aos dados e às instruções, pode ser necessária a doação de espaços de uma cache para a outra.

Foram feitas diversas simulações para avaliar a eficiência de desempenho da

arquitetura proposta. Para essas simulações foram utilizados diversos traces reais do BYU *trace distribution center*. Inicialmente foram feitos testes para verificar os melhores parâmetros a serem utilizados para as próximas simulações de comparações de desempenho com os principais trabalhos relacionados. Os três principais parâmetros testados foram: o valor do *quantum,* o ganho na utilização do módulo de decisão de reconfiguração e o ganho na utilização do móludo de determinação de configuração. Foi possível verificar que, para aqueles traces simulados, o uso do módulo de decisão de reconfiguração trouxe benefícios, pois o desempenho da arquitetura proposta com esse módulo foi melhor do que o da arquitetura proposta sem o uso desse módulo, provavelmente porque ao utilizar sempre a reconfiguração ao final de todos os *quanta*, a arquitetura poderá estar alterando um comportamento que já poderia ser considerado satisfatório naquele momento. Também foi possível verificar que o melhor valor de *quantum*, dentre os três valores simulados, a ser utilizado para se decidir a necessidade da reconfiguração foi o *quantum* de 20.000 acessos. E, simular a arquitetura proposta sem a utilização do módulo de decisão de reconfiguração, ou seja, com o uso apenas do módulo de determinação de configuração, também permitiu concluir que a utilização da heurística de determinação de configuração também trouxe benefícios para a arquitetura, pois o desempenho dessa instância da arquitetura proposta foi melhor, para todos os traces simulados, do que o da arquitetura convencional.

Após a definição dos parâmetros, foi comparado o desempenho, em relação à taxa de cache miss, entre a arquitetura proposta e os principais trabalhos relacionados (Carvalho_Martins e Kerr_Midorikawa). Apesar de que em alguns traces, a arquitetura proposta obteve uma taxa de cache miss de instruções maior do que os trabalhos relacionados, a arquitetura proposta obteve para esses traces, uma taxa de cache miss de dados significativamente inferior, conseguindo, assim, reduzir a taxa de cache miss total em comparação aos trabalhos relacionados, considerando todos os traces simulados. De acordo com a heurística de decisão de reconfiguração, em todas as simulações realizadas, a arquitetura se reconfigurou entre 40% e 60% do total de vezes em que poderia haver a reconfiguração. O uso das duas heurísticas propostas trouxe uma melhora de 0,44% à 10.66%, em relação à combinação das melhores taxas de cache miss das duas arquiteturas relacionadas, Mixed Carvalho_Martins&Kerr_Midorikawa.

Assim, a arquitetura proposta, além de ter conseguido melhorar o desempenho, reduzindo a taxa de cache miss total em comparação com a arquitetura convencional e com os principais trabalhos relacionados, a arquitetura proposta também pode conseguir reduzir o tempo total de reconfiguração, pois ela não se reconfigurou em 100% das vezes, ela se reconfigurou em uma média de 50% das vezes.

Portanto, todos os objetivos e metas propostos foram alcançados com o desenvolvimento e a verificação da arquitetura do Split Cache Reconfigurável com os módulos de decisão de reconfiguração e o de determinação de configuração.

5.2 Principais Contribuições

Nesta pesquisa propusemos uma arquitetura de um Split Cache Reconfigurável capaz de decidir pela necessidade ou não da reconfiguração da arquitetura em cada *quantum* e capaz de realizar a reconfiguração de forma mais eficiente. Portanto, as principais contribuições com a proposta da arquitetura de Split Cache Reconfigurável são:

- Proposta e implementação da arquitetura de Split Cache Reconfigurável;
- Proposta e implementação do módulo de decisão de reconfiguração;
- Proposta e implementação do módulo de determinação da melhor configuração;
- Verificação e comparação de desempenho dessa arquitetura com uma arquitetura convencional e com os trabalhos relacionados;
- Análise dos resultados de desempenho obtidos com a comparação entre a arquitetura proposta e os principais trabalhos relacionados.

Devido a limitações de tempo para a finalização desta dissertação, a nossa pesquisa ficou limitada à simulação em software dos resultados, sendo assim, a arquitetura não foi implementada em hardware.

5.3 Trabalhos futuros

A partir dos resultados obtidos com esse trabalho, é possível observar alguns aspectos que ainda podem ser melhorados na arquitetura, podendo assim, provavelmente, melhorar ainda mais o desempenho da memória cache.

Um primeiro aspecto a ser estudado é a possibilidade de se utilizar pré-busca para se determinar quais são os *slots* doadores e recebedores de acordo com o futuro, pois atualmente, os *slots* são determinados utilizando os acessos recentemente passados.

Estudar a possibilidade de se reconfigurar uma arquitetura com mais de um nível de cache, ou seja, utilizar a política de reconfiguração adaptada para mais de um nível de cache, pois atualmente só é utilizado no primeiro nível de cache.

Estudar a possibilidade do módulo de decisão de reconfiguração verificar a necessidade da reconfiguração independentemente de um valor fixo de *quantum*.

Simular a arquitetura com outros parâmetros, como outros valores para a quantidade de espaços que são doados por vez, pois nessa proposta esse valor foi fixado em 2 espaços por reconfiguração e outros valores para o *quantum*, pois foram utilizados apenas os *quanta* com 1.000, 20.000 e 100.000 acessos.

Simular a arquitetura proposta com mais *traces* reais para se avaliar e comparar o desempenho.

E por fim, verificar o desempenho da arquitetura implementando-a em FPGA, para analisar o *overhead* com o uso das heurísticas em FPGA e o tempo gasto para a reconfiguração, pois atualmente os resultados foram obtidos da mesma forma que os trabalhos relacionados, por simulação.

REVISÃO BIBLIOGRÁFICA

AGERWALA, T.; CHATTERJEE, S. Computer architecture: Challenger and opportunities for the next decade. IEEE Micro, v. 25, n.3, p.58-69, Mai. 2005.

AIKEN, Howard H.; HOPPER, Grace M. The Automatic Sequence Controlled Calculator. Electrical Engineering, Vol. 65 No.8-9, pp.384-391, Aug 1946.

ALBONESI, D. H. Selective Cache Ways: On-Demand Cache Resource Allocation. In Proceedings of the 32nd Annual International Conference on Microarchitecture, 1999.

BISHOP, P.; SULLIVAN, C. A. Reconfigurable future, In Proc. of IEEE International Conference on Field-Programmable Technology. pages 2-7, 2003.

BOBDA, C. Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications (1st ed.). Springer Publishing Company, Incorporated, 2007.

BONDALAPATI, K.; PRASANNA, V. K. Reconfigurable computing systems. Proceedings of the IEEE, 90(7): 1201-1217, 2002.

BONDALAPATI, K.; PRASANNA, V. K. Reconfigurable computing: Architectures, models and algorithms. Current Science, 78(7): 828–837, Apr. 2000.

BORKAR, S. Y. et al. Platform 2015: Intel R processor and platform evolution for the next decade. Intel, 2005.

BOSE, P. Workload characterization: A key aspect of microarchitecture design. IEEE Computer Society, v. 26, n. 2, p. 5-6, Mar. 2006.

BYU, BRIGHAM YOUNG UNIVERSITY TRACE DISTRIBUTION. Website: http://traces.byu.edu/. Acessado em dez/09.

BURGER D.; GOODMAN, J. R.; KAGI, A. Memory Bandwidth Limitations of Future Microprocessors, In Proc. of 23rd Internacional Symposium of Computer Architecture, 1996, p. 78-89.

CARVALHO, M. B. Proposta e desenvolvimento de uma Arquitetura de Memória Cache Reconfigurável, Dissertação, 2005.

CARVALHO, M. B.; MARTINS, C. A. P. S. Arquitetura de Cache com Associatividade Reconfigurável. In: V Workshop em Sistemas Computacionais de Alto Desempenho, pp. 50-57, 2004.

CHANG, CHIH-YUNG; SHEU, JANG-PING; CHEN, HSI-CHIUEN Reducing Cache Conflicts by Multi-Level Cache Partitioning and Array Elements Mapping. J. Supercomput. 22, 2, 197-219, 2002.

CHEN, L.; ZOU, X.; LEI, J.; LIU, Z. Dynamically Reconfigurable Cache for Low-Power Embedded System. In Proceedings of the Third International Conference on Natural Computation - Volume 05 (ICNC '07), Vol. 5. IEEE Computer Society, Washington, DC, USA, 180-184, 2007.

COMPTON C.; HAUCK, S. An introduction to reconfigurable computing, IEEE Computer, Invited Paper, Abr. 2000.

COMPTON C.; HAUCK, S. Reconfigurable computing: a survey of systems and software. ACM Comput. Surv. 34, 2, 171-210, 2002.

CONTI, C. J.; GIBSON, D. H.; PITKOWSKY, S. H. Structural aspects of the System/360 Model 85 Part I: General organization, IBM Systems Journal, v.7, n.1, 1968, p.2-15.

COUTINHO, L. M. N.; MENDES, J. L. D.; MARTINS, C. A. P. S. MSCSim - Multilevel and Split Cache Simulator, In Proc. of 36th Frontiers in Education Conference. pages T1F 7-12, 2006.

COUTINHO, L. M. N.; MENDES, J. L. D.; MARTINS, C. A. P. S. Dynamically Reconfigurable Split Cache Architecture. In Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs (RECONFIG '08). IEEE Computer Society, Washington, DC, USA, 163-168, 2008.

DASARATHAN, D.; KULANDAIYAN, S. Adaptive Cache Replacement Technique, The 9th International Conference on High Performance Computing (HiPC 2002)

DEBENEDICTIS, E. P. Will moore's law be sufficient? In Proc. of 2004 IEEE/ACM Conference on Supercomputing, pages 57-68. IEEE Computer Society, ACM SIGARCH Publisher, Nov. 2004.

DEHON, A. The density advantage of configurable computing, IEEE COMPUTER, v. 33, n. 4, 2000, p. 41–49.

DUBEY, P. A platform 2015 workload model recognition, mining and synthesis moves computers to the era of tera. Intel Corporation, 2005.

EIGENMANN, R.; LILJA, D. J. Von Neumann Computers. In Wiley Encyclopedia of Electrical and Electronics Engineering, Vol. 23, pp. 387-400, 1999.

GIL, A. D. S.; BENITEZ, J. I. B.; CALVINO, M. H.; GOMEZ, E. H. "Reconfigurable Cache Implemented on an FPGA," Reconfigurable Computing and FPGAs, International Conference on, pp. 250-255, 2010 International Conference on Reconfigurable Computing and FPGAs, 2010

HAIKALA, I. J.; KUTVONEN, P. H. Split Cache Organizations. In Performance'84 Proceedings of the Tenth International Symposium on Computer Perfomance Modelling, Measurement and Evaluation, pages 459-472. 1985.

HANDY, JIM. The Cache Memory Book. Morgan Kaufman, 2nd Edition, 1998.

HARTENSTEIN, R. A decade of reconfigurable computing: a visionary retrospective. In. Proc of Conference on Design, Automation and Test in Europe. pages 642-649, Mar. 2001.

HAUCK, S.; DEHON, A. Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation, Morgan Kaufmann/Elsevier, 2008.

HENNESSY, J. L.; PATTERSON, D. A. Computer Organization and Design: Hardware/Software Interface, Morgan Kaufmann, 4th Edition, 2008.

HENNESSY, J. L.; PATTERSON, D. A. Computer Architecture: A quantitative approach, Morgan Kaufman, 4th Edition, 2006.

IRWIN, M. J.; SHEN J. P. “Revitalizing Computer Architecture Research”, CRA Conference on Grand Research Challenges, Dec. 2005, http://www.cra.org/Activities/grandchallenges/architecture/home.html.

KERR JUNIOR, Roberto Borges. Proposta e desenvolvimento de um algoritmo de associatividade reconfigurável em memórias cache, 2008.

MARTINS, C. A. P. S.; Ordonez, E. D. M.; Corrêa, J. B. T.; Carvalho, M. B Computação Reconfigurável: Conceitos, tendências e aplicações. XXII Jornada de atualização em informática – JAI, v.2, p.339-388, 2003.

MENDES, J. L. D.; COUTINHO, L. M. N.; MARTINS, C. A. P. S. Web memory hierarchy learning and research environment, In: Proc. of Workshop On Computer Architecture Education, 33th Internacional Symposium On Computer Architecture. pages 25-32, 2006.

MESQUITA, D., MORAES, F. G., PALMA, J. C. S., MÖLLER, L. H., CALAZANS, N. L. V. Reconfiguração Parcial e Remota de Cores FPGAs. In: Workshop IBERCHIP, 7, 2001, Montevidéu.

PATTERSON, D.; ANDERSON, T.; CARDWELL, N.; FROMM, R.; KEETON, K.; KOZYRAKIS, C.; THOMAS, R.; YELICK, K. A Case for Intelligent RAM, IEEE Micro, v. 17, n. 2, 1997, p. 34-44.

RAMANATHAN, R. M.; BRUENING, F. Architecting the era of tera. Intel Corporation, 2006.

RANGANATHAN, P.; ADVE, S.; JOUPPI, N. P. Reconfigurable Caches and Their Application to Media Processing, Proceedings of the International Symposium onComputer Architecture, pp. 214–224, 2000.

SANGIREDDY, R.; KIM, H.; SOMANI, A.K. Low-power high-performance reconfigurable computing cache architectures, IEEE Transactions on Computers, Volume 53, Issue 10, pp. 1274-1290, 2004.

SARIKAYA, R.; BUYUKTOSUNOGLU, A. A Unified Prediction Method for Predicting Program Behavior. IEEE Trans. Comput. 59, 2, 272-282, 2010.

SCHALLER, R. R. Moores Law: Past, Present and Future. IEEE Spectrum, v. 34, n. 6, p. 52-59, Jun. 1997.

SMITH, A. J. Cache Memories. Computing Surveys, vol. 14, no. 3, pp. 473-530, Sept. 1982.

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

STIERHOFF, G. C.; DAVIS, A. G. A History of the IBM Systems Journal. IEEE Annals of the History of Computing, v. 20, n.1 (jan. 1998), 29-35.

VEIDENBAUM, A.; TANG, W.; GUPTA, R.; NICOLAU, A.; JI, X. Adapting Cache Line Size to Application Behavior, Proceedings of the 13th ACM International Conference on Supercomputing, pp. 145-154, 1999.

VON NEUMANN, J. First Draft of a Report on the EDVAC. Moore School of Electrical Engineering, Univ. of Pennsylvania, Philadelphia, June 30,1945.

VON NEUMANN, J. First Draft of a Report on the EDVAC. IEEE Annals of the History of Computing. v.15, n.4 (Oct. 1993), 27-75. DOI= http://dx.doi.org/10.1109/85.238389

XU, Z.; SOHONI, S.; MIN, R.; HU, Y. An Analysis of Cache Performance of Multimedia Applications. Transaction on Computers, 53(1): 20-38, Jan. 2004.