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

Arquitetura Autônoma e Inteligente de Memória *Cache* Reconfigurável

FRANCO RAFAEL CAMPOS CORRÊA

Belo Horizonte
2008

FRANCO RAFAEL CAMPOS CORRÊA

Arquitetura Autônoma e Inteligente de Memória *Cache* Reconfigurável

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Pontifícia Universidade Católica de Minas Gerais, como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Orientador:
Prof. Dr. Carlos Augusto Paiva da Silva Martins
Co-Orientador:
Prof. Dr. Petr Iakovlevitch Ekel

Belo Horizonte
2008

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

Corrêa, Franco Rafael Campos
C824a Arquitetura autônoma e inteligente de memória *cache* reconfigurável / Franco Rafael Campos Corrêa. Belo Horizonte, 2010.
65f. : 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.Otimização Matemática. 3. Arquitetura de Computador. I. Corrêa, Franco Rafael Campos. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU:621.3

Resumo

Nas últimas décadas, tem sido crescente o desenvolvimento de inovações arquiteturais com o objetivo de melhorar o desempenho dos sistemas de computação [Hennessy 2006] [Patterson 2005]. Como exemplos, pode-se citar a utilização de hierarquias de memórias e memórias *cache* [Smith 1982], o crescimento de paralelismo em *hardware*, o aumento da utilização de *hardware* configuráveis [Martins 2003], os *pipeline* superescalares [Rochange 2005], [Carvalho 2004] [Asaduzzaman 2006], utilização de *chips* com vários processadores (CMP – *Chip Multiprocessor*) [Hammond 1997] [Pengyong 2006], sistemas inteiros embutidos dentro de um *chip* (SoC – *System on a Chip*), dentre outros [Chidester 2002].

As memórias *cache* surgiram com o objetivo de proporcionar uma melhoria no desempenho dos sistemas de computação, pois normalmente as memórias principais possuem um tempo de acesso relativamente elevado. Atualmente a maioria das memórias *cache* possui uma configuração que não pode ser alterada para a execução de cada carga de trabalho, sendo assim ela não é ideal para cada uma das cargas de trabalho que podem ser executadas no processador.

Além dos aspectos arquiteturais, aspectos relacionados às técnicas e métodos de projetos das arquiteturas de sistemas de computação também podem ser aperfeiçoados. A maior parte dos processos de projeto, são realizados utilizando métodos baseados em tentativa e erro, métodos de simulação, e poucos utilizando métodos de otimização. A arquitetura proposta nos permite observar o quanto o desempenho de arquiteturas inteligentes aumenta quando comparada com arquiteturas comuns. Juntamente com a adição da inteligência, o aumento na autonomia da memória *cache* reconfigurável, torna ainda melhor o desempenho do sistema de computação.

Os resultados obtidos mostram que mesmo uma simples otimização, juntamente com a possibilidade de se possuir um hardware reconfigurável, pode aumentar e muito o desempenho de uma memória *cache*.

Abstract

Researches in architectural innovations with the purpose of improve the performance in computer systems has grown substantially in the last few decades [Hennessy 2006] [Patterson 2005]. For example, the increase in the use of parallel hardware's, reconfigurable hardware's [Martins 2003], superescalares pipelines [Rochange 2005], cache memories, the use of hierarchical memories [Smith 1982] [Asaduzzaman 2006] [Carvalho 2004] in processors, chip multiprocessors) [Hammond 1997] [Pengyong 2006], systems on chips, etc.

In this context, the cache memories appear to increase the performance of computer systems, because the main memories normally have a high average access time. The performance of the cache memories is related to the workload in execution. Researches

In the last few years, the memory architectures (including cache memory architectures) have been the one of the major problems in computer systems performance. One of the causes, perhaps the main one, is the fact that their architecture has a fixed configuration. The final configuration is made based on tests with multiple workloads. This means that the configuration set will not always perform well for each of the workloads, and even those that have performed well, in some cases, this performance could be even greater if the cache had another more appropriate setting.

The main objective of this research is to develop a reconfigurable cache memory architecture that can adapt itself to different kind of workloads in an autonomous and intelligent way. With the addition of autonomy and intelligence, we want to obtain reconfigurable caches memories with greater performance and with response times smaller than the current commercial caches.

The results show that even a simple optimization, along with the possibility to have a flexible hardware, can increase the performance of a cache. In all results our proposed architecture obtained better performance. We can say that the greater the is the size of the cache, the better is the chance of finding parameters close to ideal.

Sumário

Lista de Abreviações

CLB – *Configurable Logic Block*

CMP – *Chip Multiprocessor*

CSoC – *Configurable System on a Chip*

EPROM – *Erasable Programmable Read Only Memory*

FIFO – *First In First Out*

FO – Função Objetivo

FPGA – *Field Programmable Gate Arrays*

IOB – *Input/Output Block*

ISE – *Integrate Software Environment*

LSDC – Laboratório de Sistemas Digitais e Computacionais

LUT – *Look Up Tables*

LMB – *Local Memory Bus*

MAR – Módulo de Armazenamento Reconfigurável

MO – Módulo de Otimização

MSCSim – *Multilevel and Split Cache Simulator*

MPGA – *Mask Programmable Gate Array*

PAL – *Programmable Array Logic*

PLA – *Programmable Logic Array*

PPGEE – Programa de Pós-Graduação em Engenharia Elétrica

RTR – *Run-time Reconfiguration*

SPECfp – *Standard Performance Evaluation Corporation Floating Point*

SPECint – *Standard Performance Evaluation Corporation Integer*

SoC – *System on a Chip*

UCP – Unidade Central de Processamento (CPU - *Central Processor Unit*)

V(VHSIC)HDL – *Very High Speed Integrated Circuits Hardware Description Language*

Índice de Figuras

Índice de Tabelas

Capítulo 1 - INTRODUÇÃO

Neste primeiro capítulo será apresentado o contexto no qual está inserida a pesquisa, assim como o problema que motivou o início da mesma. Em seguida serão apresentados os objetivos e metas que se pretende atingir, a importância e relevância, a justificativa, o escopo da pesquisa e finalmente a organização da dissertação.

1.1 CONTEXTO

Nas últimas décadas, tem sido crescente o desenvolvimento de inovações arquiteturais com o objetivo de melhorar o desempenho dos sistemas de computação [Hennessy 2006] [Patterson 2005]. Como exemplos, pode-se citar a utilização de hierarquias de memórias e memórias *cache* [Smith 1982], o crescimento de paralelismo em *hardware*, o aumento da utilização de *hardware* configuráveis [Martins 2003], os *pipeline* superescalares [Rochange 2005], [Carvalho 2004] [Asaduzzaman 2006], utilização de *chips* com vários processadores (CMP – *Chip Multiprocessor*) [Hammond 1997] [Pengyong 2006], sistemas inteiros embutidos dentro de um *chip* (SoC – *System on a Chip*), dentre outros [Chidester 2002].

As memórias *cache* surgiram com o objetivo de proporcionar uma melhoria no desempenho dos sistemas de computação, pois normalmente as memórias principais possuem um tempo de acesso relativamente elevado. As memórias *cache* possuem como principal função, reduzir o tempo médio de acesso à memória, duplicando os dados contidos na memória principal em um módulo menor composto por dispositivos de acesso mais rápidos.

O desempenho das memórias *cache* está relacionado com a carga de trabalho (carga de trabalho) em execução. Cada programa que é executado no processador possui um conjunto de instruções e um conjunto de dados a serem processados. Os acessos às instruções e dados na memória formam uma seqüência de acessos chamada de *memory trace*. Para cada *memory trace* existe pelo menos uma configuração otimizada para a *cache*, ou seja, cada carga de trabalho que é executada em um processador, possui uma configuração de memória *cache* ideal. Atualmente a maioria das memórias *cache* possui uma configuração que não pode ser

alterada para a execução de cada carga de trabalho, sendo assim ela não é ideal para cada uma das cargas de trabalho que podem ser executadas no processador.

Além dos aspectos arquiteturais, aspectos relacionados às técnicas e métodos de projetos das arquiteturas de sistemas de computação também podem ser aperfeiçoados. A maior parte dos processos de projeto, são realizados utilizando métodos baseados em tentativa e erro, métodos de simulação, e poucos utilizando métodos de otimização.

Tem sido crescente a pesquisa relacionada com memórias *cache* reconfiguráveis. Esse tipo de memória consegue se adaptar às diferentes cargas de trabalho. Os grandes problemas desse tipo de *cache*, são a falta de inteligência e a falta de autonomia para realizar as reconfigurações.

1.2 PROBLEMA MOTIVADOR

Nos últimos anos as arquiteturas das memórias (inclusive das memórias *cache*) têm sido um dos principais gargalos no desempenho dos sistemas de computação. Uma das causas, talvez a principal, é o fato de as arquiteturas possuírem uma configuração fixa. A configuração final de uma *cache* é definida com base em testes com múltiplas cargas de trabalho. Isso significa que nem sempre aquela configuração terá desempenho satisfatório para cada uma das cargas de trabalho, e mesmo as que possuírem um bom desempenho, em alguns casos, esse desempenho poderia ser ainda maior se a *cache* possuísse outra configuração mais adequada.

Assim o problema que motiva o desenvolvimento desta pesquisa é a dificuldade de se projetar memórias *cache* reconfiguráveis que possuam uma maior inteligência e uma maior autonomia para escolher a melhor configuração para cada carga de trabalho a ser executada.

1.3 OBJETIVOS E METAS

Ao se analisar o problema motivador, pode-se verificar a possibilidade de se utilizar técnicas para melhorar o desempenho dessas memórias *cache* reconfiguráveis e ainda torná-las mais flexíveis.

1.3.1 Objetivo principal

O objetivo principal dessa pesquisa é desenvolver uma arquitetura de memória *cache* reconfigurável que possa se adaptar às diferentes cargas de trabalho de maneira autônoma e inteligente. Com a adição de autonomia e inteligência, pretende-se obter memórias *cache* reconfiguráveis com maior desempenho e com tempos de resposta menores que os das memórias *cache* comerciais atuais.

1.3.2 Objetivos Intermediários

Como objetivos intermediários, deseja-se propor e desenvolver uma arquitetura de memória *cache* reconfigurável que utiliza técnicas de otimização em seu mecanismo de reconfiguração, assim possibilitando uma maior flexibilidade e um desempenho mais próximo do ideal.

Além da possibilidade de reconfiguração, deseja-se conseguir com técnicas de otimização e tomada de decisões melhoria na inteligência e na autonomia de *cache* reconfiguráveis.

Deseja-se comparar a melhoria de desempenho entre memórias *cache* convencionais e *cache* que utilizam a arquitetura proposta nesta pesquisa.

1.3.3 Metas

1. Arquitetura autônoma e inteligente de memória *cache* reconfigurável.
2. Memória *cache* reconfigurável que utiliza técnicas de otimização em seu mecanismo de reconfiguração.
3. Implementação da arquitetura proposta em um dispositivo reconfigurável (opcional).

1.4 IMPORTÂNCIA E RELEVÂNCIA

A principal importância de uma pesquisa nesse contexto surge do fato do desempenho da memória *cache* ter grande impacto no desempenho dos sistemas de computação.

Esta pesquisa é relevante, pois ainda são poucos os estudos na área de memórias *cache* reconfiguráveis e menores ainda aqueles que utilizam técnicas de otimização. Como

atualmente a demanda por memórias mais rápidas tem aumentado, uma pesquisa nessa área teria muito a contribuir.

Cache tradicionais atuais possuem uma arquitetura fixa, a possibilidade da memória *cache* se adaptar a carga de trabalho pode trazer grandes ganhos de desempenho, economia de energia, flexibilidade, etc. [Carvalho 2004] [Carvalho 2005]. Além disso, a utilização de técnicas de otimização durante a reconfiguração da memória *cache* também pode proporcionar melhor desempenho para o sistema de computação. Sendo assim, a utilização de *hardware* reconfiguráveis pode proporcionar além da flexibilidade, uma certa economia de energia, sendo que os blocos lógicos que não estiverem sendo utilizados podem permanecer desligados [Albonesi 2003]. A utilização de técnicas de otimização e de tomada de decisões, pode garantir maior inteligência e maior autonomia às *cache* reconfiguráveis.

Analisando as memórias *cache* existentes no mercado atualmente, não foi encontrada nenhuma que possui a capacidade de se adaptar e/ou reconfigurar. Já no estado da arte não foi encontrada uma quantidade significativa de trabalhos relacionados à memórias *cache* adaptáveis e/ou reconfiguráveis que utilizem otimização de alguma forma. Nenhum trabalho relacionado à inteligência e autonomia de memórias *cache* reconfiguráveis foi encontrado. Logo, a possibilidade da utilização de técnicas de otimização em tempo de execução torna a proposta ainda mais relevante.

1.5 JUSTIFICATIVA

Uma das principais motivações desta pesquisa é a importância das memórias *cache* e do seu impacto no desempenho dos sistemas de computação, citado anteriormente, além do interesse pela área, desenvolvido a partir do trabalho de diplomação [Corrêa 2005] e trabalhos em dissertações anteriores do grupo de pesquisa [Carvalho 2004] [Carvalho 2005].

A crescente demanda por sistemas de computação mais velozes e otimizados tem enfrentado o problema da lenta redução no tempo de acesso das memórias. Enquanto os processadores evoluem rapidamente, as memórias evoluem numa taxa muito mais lenta. A utilização de *hardware* reconfiguráveis juntamente com técnicas de otimização pode proporcionar um aumento de desempenho e de flexibilidade que as memórias principais, *cache* e virtuais atuais não possuem.

Em conjunto, várias pesquisas nas áreas ligadas a essa pesquisa, tais como, *hardware* reconfiguráveis, hierarquia de memórias, entre outras, vem sendo realizadas no Laboratório de Sistemas Digitais e Computacionais (LSDC) no Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da PUC-Minas. Além das pesquisas, o laboratório ainda contém toda infra-estrutura e materiais necessários para o desenvolvimento dessa pesquisa.

1.6 ESCOPO DA PESQUISA

Pretende-se atingir todos os objetivos e metas propostas, mas devido a limitações de tempo, será dada maior ênfase no objetivo principal. As metas obrigatórias serão as duas primeiras citadas na subseção 1.3.3.

As contribuições desta pesquisa estão na área de hierarquia de memória, mais especificamente na área de arquiteturas de memórias *cache*. Pretende-se obter memórias *cache* próximas das ideais para cada carga de trabalho. A pesquisa não tem como objetivo apresentar contribuições na área de otimização. A otimização servirá como meio de se atingir as metas estabelecidas anteriormente.

Pelo fato desta pesquisa, no momento, ter fins apenas acadêmicos, não faz parte do escopo desta pesquisa implementar uma memória *cache* real, sendo considerada como um trabalho futuro a ser feito.

1.7 ORGANIZAÇÃO DA DISSERTAÇÃO

O restante dessa dissertação está organizada da seguinte maneira: no capítulo 2 é apresentada uma revisão dos estados da arte; no capítulo 3 é descrita a hipótese de solução e a arquitetura proposta; no capítulo 4 são apresentados e analisados os resultados obtidos através da pesquisa; no capítulo 5 são apresentadas as conclusões obtidas a partir dos resultados; no capitulo 6 estão as bibliografias consultadas e no apêndice estão alguns resultados detalhados dos testes realizados.

Capítulo 2 - REVISÃO DA LITERATURA

Neste capítulo, apresenta-se uma revisão do estado da arte dos principais objetos utilizados nesta pesquisa. As duas primeiras seções contém uma breve descrição dos fundamentos de hierarquia de memória e de memória *cache*, e a terceira seção contêm descrições e algumas técnicas de otimização e a quarta seção descreve o estado da arte de computação reconfigurável. Na quinta seção estão alguns dos principais trabalhos correlatos à pesquisa.

2.1 HIERARQUIA DE MEMÓRIA

Desde os primeiros computadores de propósito geral, as memórias têm sido um fator limitante no desempenho dos sistemas de computação. Quanto maiores as memórias mais lentas elas serão. Uma memória ideal seria grande o bastante para conter todos os dados necessários, e rápida o suficiente para que qualquer acesso tivesse intervalo de tempo praticamente zero. Como isso ainda não é possível, pesquisadores vem desenvolvendo técnicas para tornarem esses acessos menos lentos e as memórias maiores, e uma dessas soluções é a hierarquia de memória.

Hierarquia de memória é uma técnica utilizada para criar para a Unidade Central de Processamento (UCP) a ilusão de uma memória com capacidade equivalente à camada inferior da hierarquia e com velocidade de acesso equivalente à da primeira camada [Carvalho 2005] (ver **figura 1**).

Para tornar possível essa ilusão, os princípios de programação introduzem o conceito de localidade, ou seja, os programas existem num espaço de endereçamento limitado em cada instante de tempo. Essa localidade pode ser dividida em dois tipos [Patterson 2005]:

- Localidade **Temporal**: um item referenciado tem grande probabilidade de ser referenciado novamente num curto espaço de tempo. Isto significa que um dado que foi utilizado recentemente tem grandes chances de ser reutilizado em um curto espaço de tempo, como acontece em laços de repetição.

- Localidade **Espacial**: quando um item é referenciado, existe uma grande probabilidade de seus vizinhos também serem referenciados num curto espaço de tempo.

A implementação de hierarquia de memórias surge com o objetivo de conciliar grandes capacidades de armazenamento, custos e desempenho.

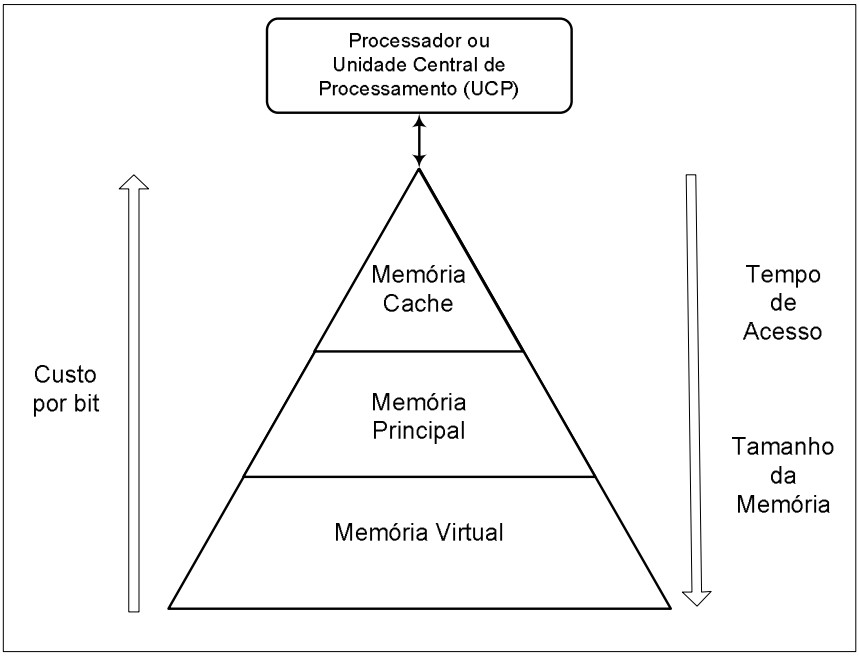

Figura 1 - Representação da hierarquia de memória de um computador

A **Figura 1** mostra a representação hierárquica das memórias de um computador. À medida que se aproximam do processador, as memórias se tornam mais rápidas, suas capacidades diminuem e seu custo por bit aumenta. Como é possível perceber, as memórias *cache* são mais velozes e possuem um tamanho reduzido.

Com base nos conceitos de localidade, as informações são copiadas de uma camada inferior para uma camada superior na hierarquia. Um acerto (*hit*) ocorre quando a informação que a unidade de processamento procura se encontra na camada imediatamente inferior. Caso contrário, ocorre uma falta (*miss*). Quando ocorre uma falta, a informação é procurada na camada inferior, e assim repetindo essa operação até que a informação seja encontrada.

O tempo médio de acesso à memória é calculado de acordo com a taxa de acertos e de faltas. Quanto mais acertos, menor o tempo médio de acesso, pois assim não é necessário buscar a informação nas camadas inferiores que são mais lentas.

2.2 MEMÓRIA *CACHE*

Dentro da hierarquia de memória, apresentada na seção anterior, a camada que se encontra entre a unidade de processamento e a memória principal é chamada de memória *cache* [Patterson 2005]. A *cache* é uma pequena quantidade de memória estática de alto desempenho com a finalidade de aumentar o desempenho do processador buscando na memória principal itens que serão utilizados pela unidade de processamento.

Embora as *cache* sejam muito mais rápidas que as memórias principais, o custo por bit de fabricação de uma *cache* também é muito maior. Esse é o motivo de não se construir sempre um computador somente com memória *cache* ou memória estática.

A memória *cache* é dividida em blocos de palavras consecutivas. Quando se fala em um bloco de tamanho 4, significa que nesse bloco são encontradas 4 palavras consecutivas da memória principal. Um processador lê palavras de tamanhos específicos, normalmente de 32 ou 64 bits.

Atualmente existem três tipos de organização de *cache*: mapeamento direto, completamente associativa e associativa por conjunto [Smith 1982], [Tanembaum 1998].

Na *cache* do tipo mapeamento direto (**Figura 2**), cada bloco só pode ficar em uma posição da *cache* também conhecida como *slot*, permitindo verificar, com apenas um acesso, se a palavra buscada se encontra na *cache*. O principal problema deste tipo de organização é que vários blocos competem por um mesmo *slot* o que pode gerar conflitos.

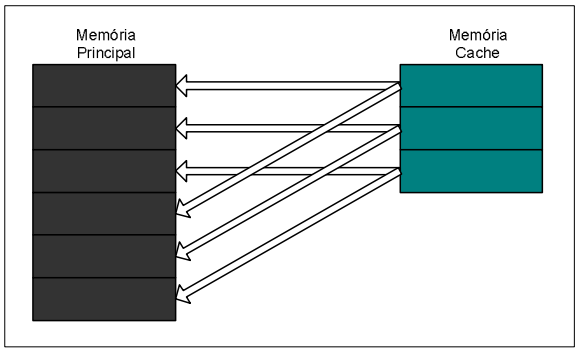

Figura 2 - Estrutura de uma *cache* do tipo mapeamento direto.

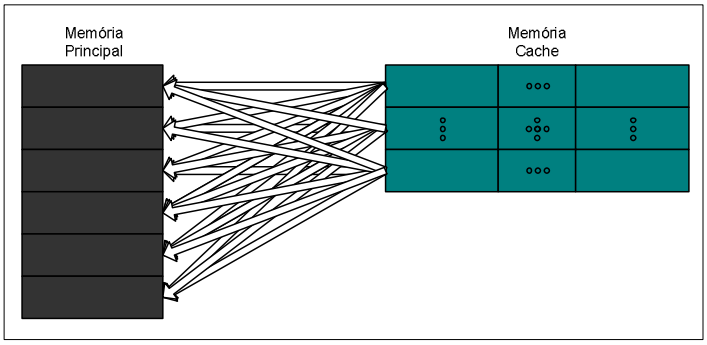

Figura 3 - Estrutura de uma *cache* do tipo completamente associativa.

Em *cache* completamente associativas (**Figura 3**) um bloco da memória principal pode ser colocado em qualquer *slot* da *cache*. Este tipo de *cache* elimina o conflito que antes ocorria nas arquiteturas de mapeamento direto. Embora blocos não disputem mais por uma mesma posição da *cache*, nesse tipo de organização para saber se existiu um *cache miss*/*hit*, deve-se verificar todos os *slots* da *cache*.

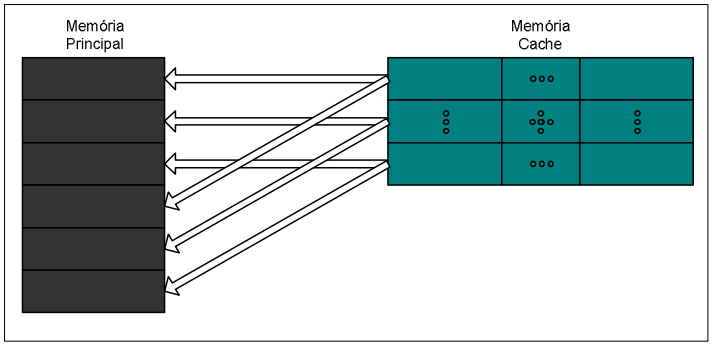

Figura 4 - Estrutura de uma *cache* do tipo associativa por conjunto.

A *cache* associativa por conjunto (**Figura 4**) é uma organização intermediária das citadas anteriormente. Os blocos podem ser encontrados em somente um *slot* e cada *slot* tem N entradas, onde N é o grau de associatividade da *cache*. Esta *cache* é chamada associativa por conjunto N-*way*. Assim, como na completamente associativa, esta *cache* também tem que realizar várias comparações para saber se ocorreu um *miss*/*hit* (neste caso N comparações).

Na verdade, as duas primeiras organizações são casos específicos da terceira. A *cache* do tipo mapeamento direto pode ser considerada como uma associativa por conjunto 1-*way*. E a *cache* completamente associativa pode ser considerada uma associativa por conjunto de 1 *slot* N-*way*.

Quando uma informação buscada pela unidade de processamento não é encontrada na *cache*, ocorre uma falta ou um *cache miss*. São três os tipos de *cache miss*:

- Compulsório: ocorrem sempre quando é a primeira vez que um bloco é procurado na *cache*. Para reduzir o impacto causado por esse tipo de *cache miss*, pode-se utilizar técnica de busca antecipada (*prefetching*) [Lee 1994], ou aumentar o tamanho dos blocos;
- Capacidade: ocorre quando blocos precisam ser substituídos por não existir mais espaço na memória *cache*, ou seja, um bloco que já esteve na *cache* teve que ser substituído por outro bloco e novamente é procurado na *cache*. Esse tipo de *cache miss* pode ser reduzido aumentando o tamanho ou a capacidade da *cache* [Chang 2002];
- Conflito: ocorre numa memória *cache* mapeamento direto ou associativa por conjunto, quando blocos diferentes disputam o mesmo local na *cache*. Aumentar o número de *slots* ou aumentar a associatividade são algumas possíveis soluções para diminuir a quantidade dessas faltas [Chang 2002].

Na **Figura 5** está representado um esquema de identificação de *cache miss*.

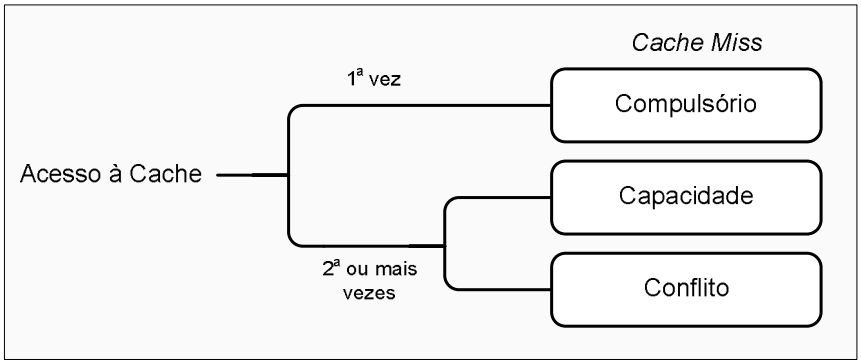

Figura 5 - Esquema de identificação dos tipos de *cache miss*

2.3 OTIMIZAÇÃO E TÉCNICAS DE OTIMIZAÇÃO

Problemas de otimização têm como objetivo maximizar ou minimizar uma função objetivo (FO). Além da FO existe também um conjunto de restrições ambos relacionados às variáveis de decisão. A resposta de um problema (ótimo global), é o maior (ou o menor) valor da função objetivo para o qual o valor das variáveis não viole nenhuma das restrições. A idéia de testar todas as combinações do domínio em busca da solução pode ser impraticável visto que o número de soluções pode ser extremamente grande.

São exemplos clássicos de problemas de otimização combinatória: o problema da mochila, o problema da cobertura mínima por conjuntos, o problema do caixeiro viajante, o problema da floresta de Steiner. Todos esses exemplos podem ser aplicados facilmente em projetos de redes de telecomunicações, redes de energia elétrica, rede de computadores, circuitos, roteamento de veículos, entre outros.

Os problemas de otimização podem ser classificados de várias maneiras, algumas apresentando métodos exatos e eficientes na sua resolução e outras com métodos não-exatos (heurísticas). Quanto à relação entre as variáveis de decisão na função objetivo e nas restrições se pode classificar o problema de otimização como sendo de programação linear ou de programação não-linear. Quanto à natureza da função objetivo existe a função convexa na qual o ótimo local é igual ao global e a função côncava na qual o ótimo local pode não coincidir com o global.

Algumas técnicas para solução de problemas de otimização são:

- Algoritmos simplex;
- Algoritmos gulosos;
- Algoritmos de grafos;
- Programação dinâmica;

O **Algoritmo simplex** consiste num procedimento algébrico que permite a determinação de pelo menos uma solução ótima para cada problema, sendo os conceitos subjacentes à resolução essencialmente geométricos [Hillier 2001] [Press 2002]. Como método de otimização linear, esse algoritmo tem como objetivo otimizar uma função objetivo sujeita a algumas restrições. O método simplex é de fácil implementação e aplicação. Permite localizar a região ótima, apesar de não oferecer informações claras com respeito ao comportamento das variáveis [Eiras 1996].

Algoritmos gulosos recebem esse nome porque a cada passo se escolhe o maior valor possível, sem refazer suas decisões, isto é, uma vez que um determinado valor foi escolhido, não se retira mais esse valor do conjunto de soluções [Brassard 1995]. Este algoritmo sempre fornece uma solução ótima para o problema se existir uma quantidade adequada de valores a ser escolhido, entretanto se a quantidade de valores for limitada, o algoritmo guloso pode não chegar a solução ótima ou não chegar em nenhuma solução, mesmo ela existindo. Outra estratégia para resolver este problema é a **programação dinâmica**, que sempre irá obter um resultado. Entretanto, algoritmos gulosos são mais simples de ser implementados.

Algoritmos de grafos são estruturas combinatórias constituídas por dois conjuntos (arestas e vértices) e uma relação de incidência entre eles. Dependendo da aplicação as arestas podem ou não ter direção. Esse tipo de configuração, de arestas e vértices, ocorre em várias aplicações conhecidas, tais como circuitos elétricos, redes de computadores, rodovias, etc. Grafos são modelos úteis para a análise de problemas em que entre pontos e conexões pode haver alguma interpretação conceitual [Gross 1998].

Alguns algoritmos de grafos conhecidos são [Grafos 2006]:

- Algoritmo de Dijkstra: para calcular o caminho mais curto num grafo com peso absoluto das extremidades;
- Algoritmo de Bellman-Ford: para calcular o caminho mais curto num grafo pesado;
- Algoritmo de Kruskal: encontra a árvore de extensão mínima para um grafo;
- Algoritmo de Prim: encontra a árvore de extensão mínima para um grafo;
- Algoritmo de Boruvka: encontra a árvore de extensão mínima para um grafo;
- Algoritmo de Floyd-Warshall: para resolver o problema do caminho mínimo entre todas as partes de vértices em um grafo com direção e peso.

Os algoritmos de **programação dinâmica** procuram resolver problemas de otimização através de uma seqüência de problemas mais simples que o original. É uma técnica utilizada quando nem todas as variáveis envolvidas estão interrrelacionadas simultaneamente [Ballard 1982].

Em um trabalho sobre otimização de memórias *cache* que realizei anteriormente, foi utilizada uma técnica conhecida como **Planejamento de Experimentos**. Além de diminuir o número de ensaios, o uso do planejamento de experimentos permite também:

- Analisar um número considerável de variáveis, que podem ser discretas ou contínuas (chamadas de fatores);
- Eliminar variáveis que não exercem uma influência significativa sobre o desempenho da memória;
- Analisar a influência que uma variável exerce sobre a outra;
- Verificar se é necessário mudar os intervalos de variação das variáveis para a obtenção de modelos adequados;
- Executar avaliações estatísticas de confiabilidade dos resultados obtidos;
- Detectar níveis ótimos.

Aplicando o planejamento de experimentos, é possível otimizar os resultados e ao mesmo tempo melhorar sua precisão. Os projetistas de sistemas de memória podem reduzir o tempo de trabalho, diminuindo o número de pontos de experiência por fator, sem ser obrigado a limitar o número de fatores, como ocorre no método clássico da experimentação [Goupy 1988].

Como em uma memória não é possível a existência de parâmetros contínuos foi utilizado um algoritmo de otimização para variáveis discretas. Atualmente, há vários algoritmos capazes de solucionar os problemas de Otimização Discreta, que informam os valores ótimos das funções objetivo, mas que são muito demorados em sua maioria. Foi utilizado um algoritmo que informa o valor quase ótimo de cada variável, para cada uma das funções objetivo analisadas, levando em consideração as restrições informadas.

O algoritmo de Otimização Discreta utilizado [Ekel 2006] foi descrito com base em uma abordagem que soluciona problemas de otimização com coeficientes fuzzy, em funções objetivo que possuem restrições. Nesta abordagem, a análise de modelos de otimização discreta fuzzy se baseia na modificação de algoritmos de otimização discreta. Os algoritmos são associados a métodos de funções normalizadas e são baseados na combinação de procedimentos formais e heurísticos.

O algoritmo de Otimização Discreta considerado permite obter uma solução que se aproxima da solução ótima, depois de um número pequeno de passos a serem seguidos.

2.4 COMPUTAÇÃO RECONFIGURÁVEL

A computação reconfigurável é uma das opções atraentes que surgiram com o objetivo de suprir o aumento da demanda dos recursos computacionais. Para proporcionar maior flexibilidade e um desempenho relativo elevado foi necessário possibilitar que dispositivos pudessem ter sua estrutura alterada após a fabricação. Atualmente, um tipo de dispositivo reconfigurável muito utilizado é o FPGA (*Field Programmable Gate Array*) [Martins 2003].

É importante ressaltar que os FPGAs não foram os primeiros dispositivos reconfiguráveis a surgir. Dentro dessa classe existem outros dispositivos que também implementam funções lógicas. Dentre esses dispositivos estão o PAL (*Programmable Array Logic*), EPROM (*Erasable Programmable Read Only Memory)* e *PLA (Programmable Logic Array*). Esses dispositivos não são capazes de implementar funções lógicas mais complexas, logo surgiram os mais recentes dispositivos programáveis como o MPGA (*Mask Programmable Gate Array*) e o FPGA (*Field Programmable Gate Array*) [Martins 2003]. A principal diferença entre o MPGA e o FPGA é que no MPGA as suas funções lógicas devem ser definidas antes da última etapa do processo de fabricação.

Os sistemas de computação reconfiguráveis podem ser implementados em arquiteturas híbridas, quando a implementação utiliza *hardware* reconfigurável e *hardware* programável, ou pura, quando a implementação é feita somente em *hardware* reconfigurável.

Dispositivos reconfiguráveis possibilitam o reuso de *hardware*, que nos sistemas de computação convencionais eram feitos para atender a propósitos específicos, proporcionando assim uma maior flexibilidade e aumentando a vida útil do dispositivo [Lien 2001]. A reconfiguração possibilita a redução do tamanho dos dispositivos, o seu tempo de projeto, podendo também reduzir o custo final do dispositivo.

Outra vantagem em se aplicar esse tipo de dispositivo é a possibilidade da configuração ser feita manual ou remotamente. A configuração remota é muito útil quando o dispositivo a ser reconfigurado encontra-se em local de difícil acesso. Por exemplo, se um robô enviado ao espaço necessita de atualização, a configuração remota pode evitar o envio de uma *miss*ão espacial, o que seria totalmente inviável [Mesquita 2001].

A reconfiguração de FPGAs pode ser feita de duas maneiras. Pode ser feita uma reconfiguração parcial ou então uma reconfiguração total do dispositivo. A reconfiguração total é realizada quando o dispositivo é inteiramente alterado. A reconfiguração parcial acontece quando apenas parte do *hardware* é alterado. Nesse caso, pode ou não ser necessária

a parada do sistema. Quando as partes do sistema que não estão sendo reconfiguradas permanecem em atividade durante a reconfiguração, a configuração é chamada de parcial dinâmica, caso contrário, é chamada de parcial estática. Além das configurações parcial e total, também existe a configuração dinâmica (RTR - *run-time reconfiguration)*, onde não há necessidade de se parar, reiniciar o circuito e/ou remover elementos durante a reconfiguração [Mesquita 2001]**.** Um FPGA também deve permitir ao usuário configurar o *hardware* de diferentes maneiras [Lien 2001]. Em [Tanougast 2003] é proposta uma metodologia de particionamento temporal para sistemas RTR, que melhoram a utilização de área lógica, mantendo a flexibilidade e ajudando o projetista a especificar os requisitos do seu sistema.

Em [Kusse 1998] são feitas comparações de consumo de energia entre vários dispositivos. O artigo ainda introduz um módulo para economia de energia em FPGAs. A utilização desse módulo pode ser eficaz para que se resolva o problema de consumo de energia em sistemas embutidos. Apesar de todos os benefícios proporcionados pela utilização de dispositivos reconfiguráveis, a utilização de FPGAs traz novos problemas para os projetistas. Alguns desses problemas estão relacionados ao desempenho, especificação e programação da funcionalidade desejada [Martins 2003]. Um dos mais preocupantes atualmente é o tempo gasto para se efetuar uma reconfiguração do dispositivo. Se esse tempo for muito alto, o dispositivo não poderá ser utilizado se for necessário um sistema de tempo real. Se o *hardware* permitir reconfiguração dinâmica então esse tempo pode não afetar o desempenho do dispositivo, mas se a cada reconfiguração o sistema pare, esse tempo deve ser o menor possível. Existem alguns métodos que diminuem esse tempo, como o citado em [Hauck 1999]. Nele, os autores mostram um algoritmo que eles desenvolveram, que reduz radicalmente a quantidade de dados a serem transferidos durante a reconfiguração.

Na **Figura 6** estão representados os principais componentes de um FPGA. Podem ser vistos blocos lógicos configuráveis também chamados de CLBs (*Configurable Logic Blocks*), blocos de entrada/saída ou IOBs (*Input/Output Blocks*), blocos de RAM, além de recursos de relógio. Os blocos lógicos (CLBs) são compostos por LUTs (*look up tables*), que implementam a funcionalidade lógica, *flip-flops* para implementar a lógica seqüencial, e os multiplexadores criam a conectividade lógica [Renovell 2001] [Martins 2003][Xilinx 2007].

Os módulos são configuráveis através de um arquivo binário de configuração que altera o roteamento programável e os *bits* de memória de configuração do dispositivo para que o FPGA implemente a funcionalidade do projeto. O processo de projeto tem como produto final este arquivo binário que é chamado de *Bitstream*. Arquivos de configuração contêm uma

mescla de comandos e dados. Eles podem ser lidos e escritos através de uma das interfaces de configuração do FPGA [Xilinx 2007] [Mesquita 2001].

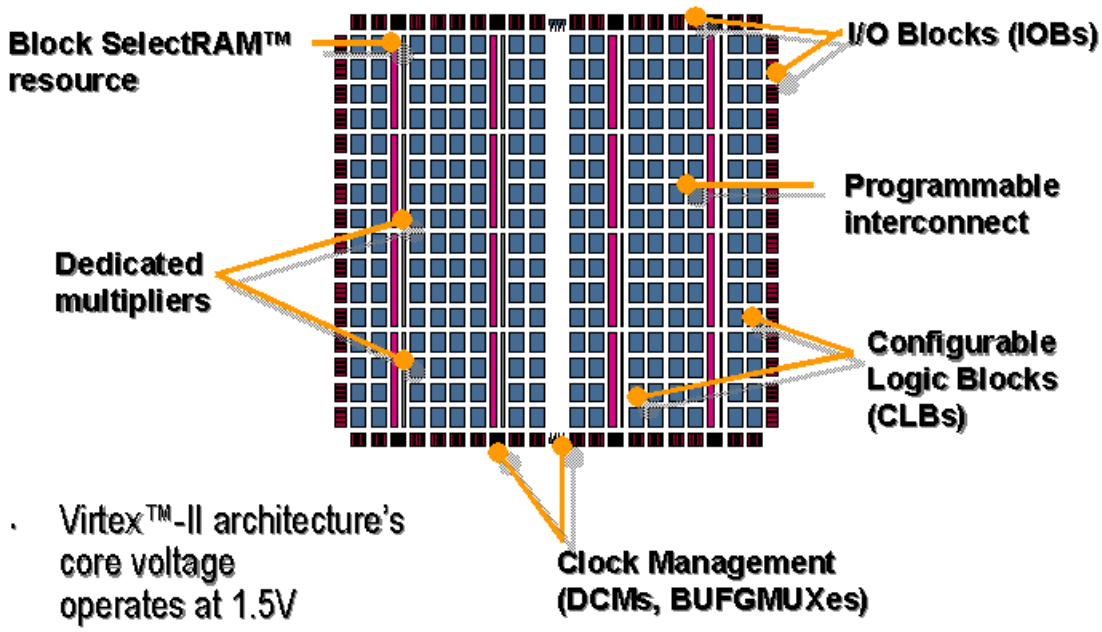

Figura 6 - Arquitetura interna de um FPGA [Xilinx 2007]

Para configurar esses dispositivos os desenvolvedores devem possuir grande conhecimento da arquitetura do *hardware* que está sendo configurado, levando em conta os seus blocos lógicos, suas interconexões e ainda como é feito o particionamento nesse *hardware*. Para facilitar a configuração, surgiram novas maneiras de desenvolvimento, linguagens e ferramentas. A utilização de ferramentas de configuração requer que o desenvolvedor conheça os componentes lógicos e o *hardware* a ser configurado. No caso da configuração utilizando linguagem, são utilizadas linguagens de descrição de *hardware* como V(VHSIC)HDL *(Very High Speed Integrated Circuits) Hardware Description Language* ou Verilog. Com isso o desenvolvedor não necessita mais ter conhecimento em projeto de *hardware* o que facilita o desenvolvimento do sistema [Martins 2003].

Pensando nas dificuldades encontradas no projeto e na busca por uma *cache* ideal para cada carga de trabalho, essa pesquisa propõe a utilização de técnicas de otimização não só no projeto das memórias *cache*, mas também no mecanismo de reconfiguração de *cache* reconfiguráveis. Assim, podem-se utilizar técnicas mais elaboradas e mais complexas quando

o objetivo for otimizar a *cache* em tempo de projeto, e técnicas mais simples que podem ser aplicadas em tempo de execução.

2.5 TRABALHOS CORRELATOS

Nesta seção são citados alguns trabalhos correlatos à pesquisa, alguns trabalhos correlatos aos problemas, alguns correlatos à solução e correlatos aos dois ao mesmo tempo.

Trabalhos relacionados à arquitetura de *cache* reconfiguráveis, em sua grande maioria, utilizam monitores para obtenção de dados e estatísticas das cargas de trabalho [Johnson 1998]. Estes monitores podem ser implementados em *software*, *hardware* ou até mesmo no sistema operacional da máquina. Existe também a possibilidade de utilizar *tags* para identificar as características da carga de trabalho a ser executada.

Em uma memória *cache* muitos parâmetros podem ser variados, como por exemplo, o tamanho do bloco ou o grau de associatividade. Em [Carvalho 2004] é proposta uma arquitetura de uma memória *cache* com associatividade reconfigurável, que possui melhor desempenho e uma taxa de erro (*cache miss*) menor, quando comparada a uma *cache* associativa por conjunto com o mesmo número de comparadores (sem considerar o overhead de configuração).

Existem trabalhos que propõem mudanças na arquitetura da *cache* com base nas localidades temporal e espacial. Um exemplo é a alteração no tamanho do bloco, como proposto em [Veidenbaum 1999], que visa otimizar o tamanho do bloco, variando seu tamanho durante a execução do programa, para determinada carga de trabalho.

Embora as *cache* proporcionem maior desempenho para o sistema computação, elas são responsáveis por uma parte significante do consumo de energia, seja dentro do processador ou não. Em sistemas de computação embutidos o consumo de energia é uma questão muito delicada e deve ser um requisito importante a ser considerado. Resultados sobre avaliação e utilização de uma *cache* de dados reconfigurável podem ser encontrados em [Benitez 2006]. A reconfiguração da *cache* é feita baseada em duas técnicas: um processo de aprendizado determina a melhor configuração da *cache* para cada fase do programa, e um processo de cognição detecta essas fases. Comparada a microarquiteturas convencionais, a arquitetura proposta mostra melhor desempenho e maior economia de energia.

O foco do trabalho de [Asaduzzaman 2006] é melhorar o desempenho de decodificação de H.264/AVC (decodificador de vídeo) através da otimização da *cache* para dispositivos móveis e embarcados. Utilizando simulação de uma *cache* de dois níveis, com o primeiro nível sendo *cache* split em *cache* de instruções e de dados e o segundo nível uma *cache* unificada. Utilizando *Cache*grind [*Cache*grind 2006] para caracterizar a carga de trabalho de decodificação e VisualSim [Mirabilis 2006] para modelar a arquitetura e executar a simulação para a carga de trabalho. Os resultados da simulação mostram que o desempenho de decodificação pode ser melhorado através da otimização da *cache*.

Trabalhos relacionados à otimização de memórias *cache* são mais difíceis de serem encontrados. É muito comum a utilização de simuladores para verificar o desempenho da *cache*. Em algumas abordagens se utiliza a simulação exaustiva de todas as soluções possíveis para se achar a configuração ótima. Quando o número de soluções se torna impraticável, algumas abordagens utilizam heurísticas [Li 2001] [Shiue 1999] [Wilton 1996] [Sato 2000].

Uma abordagem diferente é apresentada por [Ghosh 2004]. Para encontrar uma *cache* otimizada, passa-se por três fases: pré-processamento, processamento principal, e pós-processamento. Durante a primeira fase é construída uma árvore binária a partir da leitura do memory trace. Na segunda fase é computada uma tabela de faltas baseada na árvore binária construída na fase um. Finalmente na terceira fase é gerada a configuração otimizada da *cache*.

Capítulo 3 - ARQUITETURA AUTÔNOMA E INTELIGENTE DE MEMÓRIA *CACHE* RECONFIGURÁVEL

Neste capítulo será apresentada inicialmente a hipótese para a proposta de uma arquitetura autônoma e inteligente de memória *cache* reconfigurável. Serão apresentadas a arquitetura geral e algumas possíveis instâncias da arquitetura proposta, e finalmente as considerações finais deste capítulo.

3.1 HIPÓTESE PARA A PROPOSTA DE UMA ARQUITETURA AUTÔNOMA E INTELIGENTE DE MEMÓRIA *CACHE* RECONFIGURÁVEL

Como já se sabe, as memórias têm sido um fator limitante no desempenho dos sistemas de computação. Uma memória ideal seria grande o bastante para conter tudo que se precisa, e rápida o suficiente para que qualquer acesso tivesse intervalo de tempo praticamente zero. Sendo assim, pesquisadores vêm desenvolvendo técnicas para tornarem as memórias maiores e os acessos mais rápidos. As memórias *cache* foram criadas para tentar solucionar alguns desses problemas [Smith 1982] [Carvalho 2005] [Hennessy 2006].

Embora as memórias *cache* sejam rápidas, elas apresentam um problema causado pelo fato de possuírem uma arquitetura fixa (possuem uma única configuração). Essa arquitetura na maior parte dos casos possui sua configuração determinada a partir de testes com uma grande variedade de cargas de trabalho, possuindo assim, um desempenho mediano. Mesmo com cargas onde a memória *cache* possui um desempenho razoável, esse desempenho poderia ser ainda maior se a *cache* possuísse uma configuração ideal para cada carga de trabalho.

Para tentar minimizar alguns dos problemas dos *hardware* fixos algumas soluções foram propostas, dentre elas a de permitir que o hardware pudesse ter sua arquitetura alterada. Com essa possibilidade, alguns trabalhos surgiram com a idéia de construção de memórias *cache* que pudessem alterar sua arquitetura utilizando dispositivos reconfiguráveis, sendo denominadas memórias *cache* reconfiguráveis [Li 2001] [Ghosh 2004] [Carvalho 2004].

Mesmo as memórias *cache* reconfiguráveis possuem problemas. Um dos problemas encontrados foi a falta de capacidade para a determinação de uma configuração ideal para

cada uma das cargas de trabalho diferentes. Para dar uma maior autonomia e inteligência à *cache*, métodos de otimização e de tomada de decisões podem ser incorporados na arquitetura da *cache* reconfigurável. Assim não seriam necessárias ações humanas para a escolha de uma configuração ideal para cada carga de trabalho.

Diante de problemas como esse, podemos dizer que a hipótese de solução dessa pesquisa é o desenvolvimento de uma arquitetura autônoma e inteligente de memória *cache* reconfigurável. Com a possibilidade de reconfiguração são eliminados os problemas das arquiteturas fixas. Com a utilização de técnicas de otimização e de tomada de decisões é possível fazer com que a arquitetura tenha a capacidade de se adaptar inteligentemente e de maneira autônoma às diferentes cargas de trabalho, obtendo um desempenho melhor que as *cache* normais e com maior autonomia que as *cache* reconfiguráveis testadas.

A proposta de uma arquitetura inteligente de *cache* reconfigurável é interessante, pois a adaptação está praticamente em tudo que nos cerca. Com a utilização de técnicas de otimização podemos dizer que estamos dando à arquitetura um tipo de inteligência e autonomia.

3.2 APRESENTAÇÃO DA ARQUITETURA PROPOSTA

Nas subseções a seguir apresentaremos a arquitetura geral proposta, assim como algumas de suas diferentes instâncias.

3.2.1 Arquitetura Geral

As *cache* tradicionais atuais possuem apenas um módulo principal em sua arquitetura, que é o módulo de armazenamento, ou a *cache* propriamente dita. Possuem as configurações de seus módulos de armazenamento fixas, geralmente com um desempenho mediano para a maior parte das cargas de trabalho. Já as *cache* reconfiguráveis, ainda que somente em pesquisas, possuem suas arquiteturas ou comportamentos ajustáveis, alteráveis e/ou adaptáveis. Como pudemos observar nos trabalhos correlatos, um conjunto muito pequeno de *cache* reconfiguráveis utiliza técnicas de otimização para melhorarem seu desempenho. Nesse contexto será apresentada a arquitetura inteligente que é proposta nessa pesquisa.

A arquitetura proposta (**Figura 7**) é composta de cinco módulos principais. Os módulos podem ser reconfigurados e replicados conforme a necessidade de cada sistema.

Cada módulo possui uma função principal, mas podem ser substituídas ou alteradas conforme a necessidade da arquitetura.

Os cinco tipos de módulos são:

- **Módulo de Determinação da Função Objetivo;**
- **Módulo de Otimização;**
- **Módulo de Gerenciamento de Parâmetros de Reconfiguração;**
- **Módulo de Armazenamento Reconfigurável;**
- **Módulo de Distribuição.**

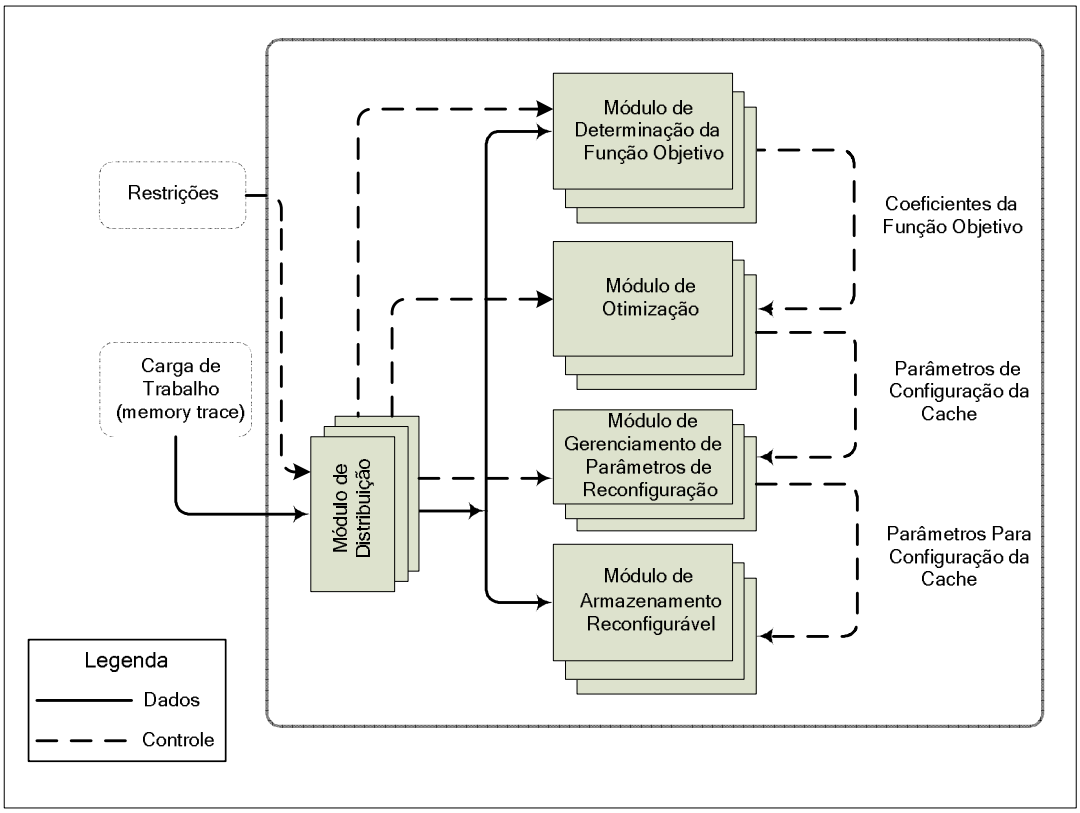

Figura 7 - Arquitetura Autônoma e Inteligente de Memória *Cache* Reconfigurável

O **Módulo de Determinação da Função Objetivo** (módulo 1) é responsável por determinar uma função que identifique e caracterize cada *memory trace* de acordo com o seu padrão de acesso à memória. O módulo recebe o *memory trace*, realiza suas funções (no caso dos testes realizados, simulações para o Planejamento de Experimentos) e disponibiliza na sua

saída os coeficientes da função objetivo. A técnica utilizada nesse módulo pode ser alterada, caso o Planejamento de Experimentos não atenda às demandas da arquitetura como um todo.

Esse módulo é reconfigurável, podendo ser intrínseco ou extrínseco ao dispositivo reconfigurável. A principal diferença entre ser intrínseco ou extrínseco é que quando o módulo se encontra extrínseco existe a possibilidade de se utilizar técnicas mais complexas e mais específicas de determinação da função objetivo, sem que o desempenho da *cache* seja afetado. Outra vantagem de utilizar esse módulo externamente, é a possibilidade de realizar todo o trabalho *offline*. Quando utilizado intrinsecamente (dentro de um hardware reconfigurável), o módulo consegue obter maior desempenho, pois executará suas funções com maior velocidade.

O **Módulo de Otimização** (módulo 2) é responsável por determinar os parâmetros arquiteturais da *cache* que sejam os ideais ou próximos do ideal para cada carga de trabalho. Para determinar essa configuração, inicialmente foi escolhida uma técnica de otimização discreta definida em [Ekel 2006]. Este módulo é reconfigurável permitindo assim que outros tipos de técnicas e métodos de otimização sejam utilizados. A técnica utilizada nos testes é muito simples, rápida e eficiente. Um de seus problemas é a possibilidade de não se atingir o ótimo global, mas em compensação não demora tanto tempo quanto um método de busca exaustiva e ainda obtém bons resultados.

No módulo de otimização os coeficientes obtidos do módulo anterior (Módulo de Determinação da Função Objetivo) são processados juntamente com as restrições (em nosso estudo de caso, o tamanho da *cache*) e como resultados são disponibilizados na sua saída os parâmetros (grau de associatividade, número de *slots* e tamanho do bloco em palavras) da *cache*. Esses parâmetros ficam então disponíveis para o próximo módulo (Módulo de Gerenciamento de Parâmetros de Reconfiguração), que é responsável por efetuar a reconfiguração do Módulo de Armazenamento Reconfigurável.

Assim como o primeiro módulo, o módulo de otimização também pode estar externo ao dispositivo reconfigurável. Isso torna possível a utilização de técnicas de otimização mais complexas e mais exaustivas sem que o desempenho da *cache* seja afetado.

O **Módulo de Gerenciamento de Parâmetros de Reconfiguração** (módulo 3) funciona como uma pequena tabela. Ele armazena as configurações otimizadas encontradas pelo Módulo de Otimização (módulo 2). Além de armazenar as configurações ele também é

responsável por controlar as configurações do módulo de Armazenamento reconfigurável (módulo 4). A possibilidade de armazenar os parâmetros otimizados é interessante, pois os módulos 1 e 2, que consomem mais tempo para execução de suas tarefas, não terão que refazer todo o trabalho para uma carga de trabalho que já tenha passado por eles.

Quando o sistema de computação é iniciado, essa *cache* reconfigurável pode ser configurada com parâmetros anteriores de cargas de trabalho que já foram executadas, ou por exemplo, os mesmos parâmetros da *cache* do processador Pentium 4 da Intel. O Módulo de Gerenciamento dos Parâmetros de Reconfiguração define qual a melhor configuração a ser utilizada. Se houver alguma configuração já definida para a carga de trabalho atual, o módulo simplesmente escolhe os parâmetros otimizados e permite a configuração do Módulo de Armazenamento Reconfigurável. Caso seja a primeira vez que a carga de trabalho é executada, o módulo 3 pode por exemplo utilizar uma configuração do Pentium 4, como dito anteriormente.

Este módulo pode ser implementado em hardware ou em software podendo estar intrínseco ou extrínseco ao dispositivo reconfigurável de acordo com as necessidades do usuário.

O **Módulo de Armazenamento Reconfigurável** (módulo 4) é o módulo responsável por assumir a configuração de uma *cache* com os parâmetros obtidos do módulo 3. Como o módulo 4 também é reconfigurável, é possível utilizar diferentes *cache*, diferentes arquiteturas e até diferentes tipos de *cache* reconfiguráveis, como por exemplo, a *cache* de [Carvalho 2004]. Esse módulo é responsável por executar as operações que uma *cache* normal executaria, tais como a busca de um endereço, verificação de um acerto ou uma falta e a comunicação com o processador e a memória principal.

O Módulo de Armazenamento Reconfigurável é o único que não permite a sua implementação externa ao dispositivo reconfigurável, pois ele precisa ser implementado inteiramente em hardware reconfigurável.

A possibilidade dos módulos 1 e 2 serem externos ao dispositivo reconfigurável pode trazer muitas vantagens. Uma vantagem visível é determinar as funções objetivo antes da carga de trabalho ser enviada para a *cache*. Esses dois módulos poderiam ser, por exemplo, implementados em software, o que facilitaria muito a manipulação e reconfiguração de suas funções.

O **Módulo de Distribuição** (módulo 5), é uma espécie de multiplexador que tem como função enviar os acessos à memória tanto para o módulo 1 quanto para o Módulo de Armazenamento Reconfigurável. Assim que um memory trace é disponibilizado para o módulo 4, o mesmo também é enviado para o módulo 1. O Módulo de Armazenamento Reconfigurável mantém a sua configuração até que os parâmetros resultantes do Módulo de Otimização sejam obtidas pelo Módulo de Gerenciamento de Parâmetros de Reconfiguração.

Este módulo também é responsável por enviar sinais de controle para os módulos 1, 2 e 3 informando se a carga de trabalho é ou não conhecida, permitindo ao módulo 3 tomar a decisão de qual seria a melhor configuração a ser implantada no Módulo de Armazenamento Reconfigurável.

3.2.2 Possíveis Instâncias da Arquitetura Proposta

Nesta subseção serão ilustradas algumas variações existentes da arquitetura geral proposta. Como dito anteriormente, a arquitetura pode ser reconfigurada, podendo ter seus módulos multiplicados, intrínsecos, extrínsecos, além de poder ter a função dos seus módulos alterada.

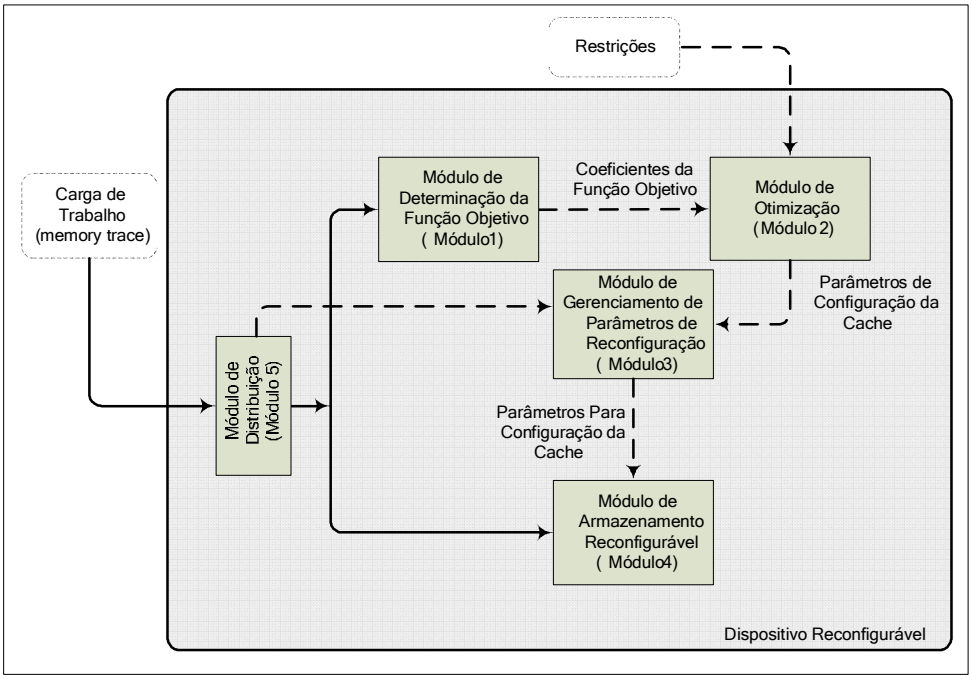

Figura 8 - Arquitetura Completamente Intrínseca.

A **Figura 8** é uma representação de um tipo de arquitetura completamente intrínseca, na qual os módulos estão todos inseridos no dispositivo reconfigurável. Por estarem implementados em *hardware* reconfigurável possuem a vantagem de executarem suas funções com maior rapidez, mas no caso da reconfiguração são mais demorados e complicados.

Esse tipo de arquitetura obtém melhor desempenho quando as cargas de trabalho já são conhecidas e o comportamento dos módulos 1 e 2 não são alterados durante a execução. Uma alteração em seus módulos faria a *cache* parar seu funcionamento (em casos onde a reconfiguração dinâmica não é possível), o que atrapalharia o seu desempenho.

Na **Figura 9** está representada uma arquitetura extrínseca. Nessa arquitetura, alguns módulos se encontram fora do dispositivo reconfigurável, permitindo que eles sejam implementados em software, por exemplo. Esse tipo de arquitetura permite a alteração do comportamento dos seus módulos com maior facilidade, além também de permitir que eles funcionem independentes do hardware. Esse tipo de arquitetura pode, em alguns casos, possuir desempenho menor que o de arquiteturas completamente intrínsecas, pelo fato de não estar totalmente implementada em hardware reconfigurável.

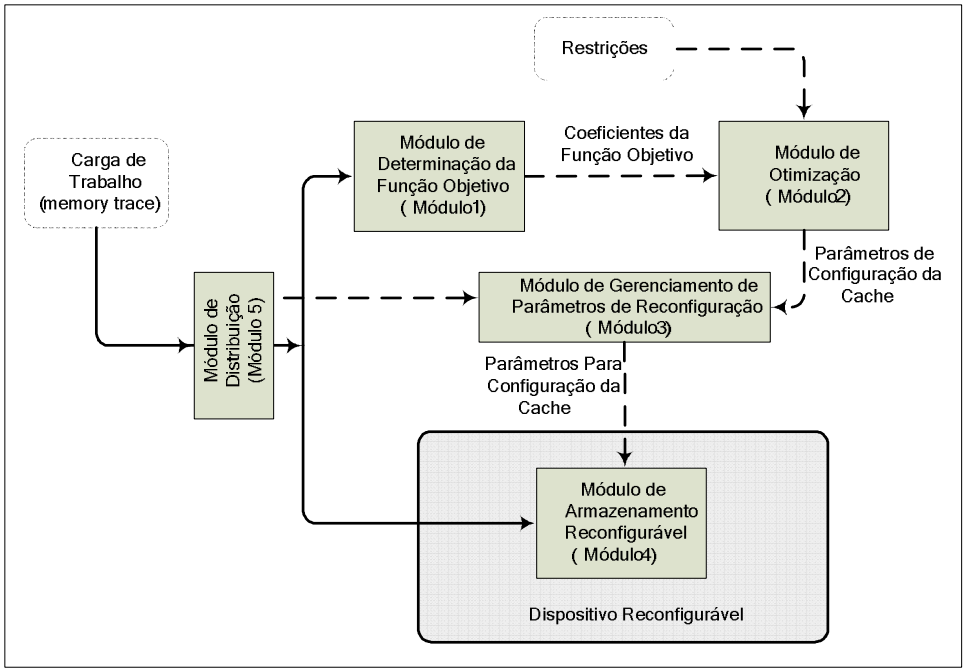

Figura 9 - Arquitetura Extrínseca.

As **figuras 10** e **11** ilustram outras instâncias possíveis da arquitetura geral proposta. Nelas podemos ver a multiplicidade dos módulos 1 e 4. A possibilidade de serem intrínsecos ou extrínsecos varia de acordo com a necessidade do usuário. Com a replicação desses módulos podemos melhorar o desempenho da arquitetura obtendo tempos de resposta cada vez menores, como poderemos ver no capítulo de resultados.

Na **Figura 10**, podemos notar a presença de vários módulos 1 (módulos 1.1, 1.2, ..., 1.8), que podem, por exemplo, ser utilizados para agilizarem a determinação dos parâmetros para otimização. Na nossa pesquisa foi utilizado o método de planejamento de experimentos para tornar possível a obtenção de uma função objetivo que descreva cada carga de trabalho. Com a utilização de oito módulos 1, poderíamos obter a função com o tempo de apenas uma simulação, isto é, a simulação seria feita nos 8 módulos paralelamente.

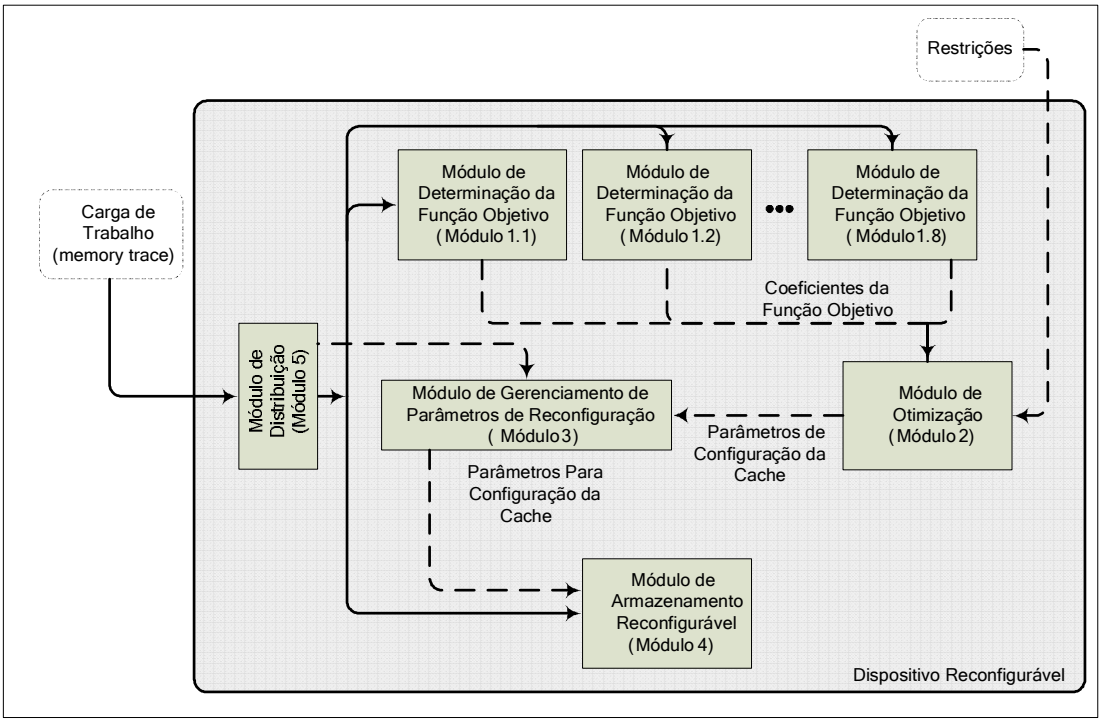

Figura 10 - Arquitetura intrínseca com Módulos de Determinação da Função Objetivo replicados

Na **Figura 11** existem dois módulos de armazenamento reconfigurável (módulo 4.1 e 4.2). Se os módulos possuírem a mesma arquitetura eles poderiam trabalhar em paralelo reduzindo o tempo de acesso da *cache*. Os módulos podem implementar configurações diferentes e enquanto um é configurado outro opera normalmente eliminando assim o

overhead de configuração. Resultados de testes utilizando este tipo de arquitetura estão expostos no capítulo de resultados.

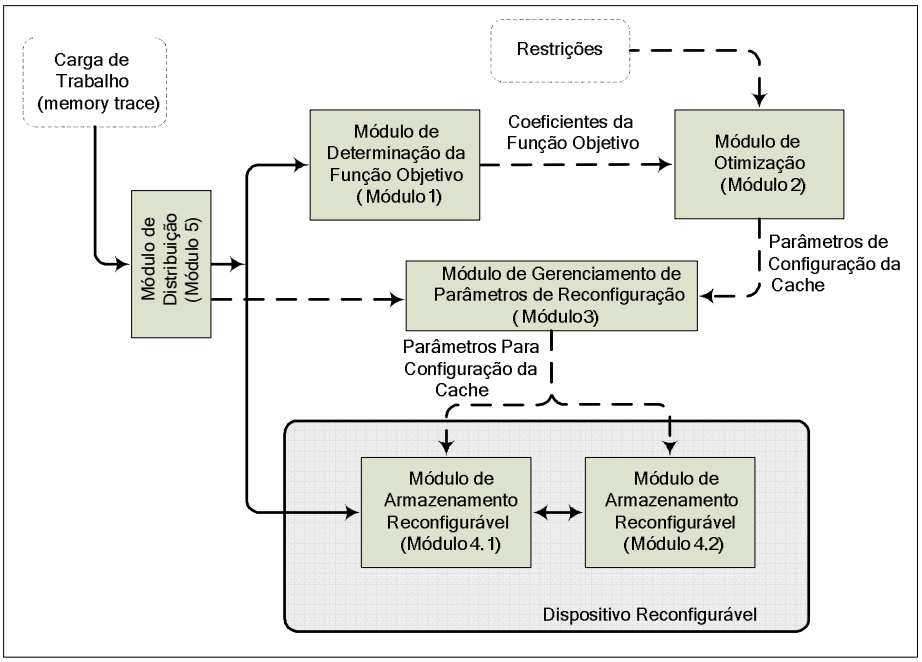

Figura 11 - Arquitetura com módulos de armazenamento replicados

A arquitetura proposta nesta pesquisa pode ser instanciada de diversas formas, cada uma com suas vantagens e desvantagens. Fica a critério do usuário do sistema de computação, a escolha de quais módulos ele deseja replicar, quais módulos estarão intrínsecos e quais estarão extrínsecos.

3.3 CONSIDERAÇÕES FINAIS

Várias são as contribuições da arquitetura proposta em relação ao estado da arte. A primeira contribuição é a inserção de um módulo de otimização que proporciona uma maior inteligência à arquitetura. A segunda contribuição é o Módulo de Determinação da Função Objetivo, que junto com o Módulo de Otimização permite à *cache* se reconfigurar de uma maneira mais autônoma. A terceira contribuição é o Módulo de Gerenciamento de Parâmetros de Reconfiguração que permite que o Módulo de Armazenamento Reconfigurável sempre possua uma configuração otimizada para cada carga de trabalho a ser executada. Assim sendo,

a grande contribuição dessa pesquisa está no fato de tornar possível o projeto de uma arquitetura de memória *cache* inteligente e autônoma que permite um ganho de desempenho e flexibilidade.

Capítulo 4 - RESULTADOS

Neste capítulo são apresentados o simulador de *cache* reconfigurável desenvolvido e utilizado, uma descrição das cargas de trabalho utilizadas, a verificação dos principais módulos da arquitetura geral proposta, os resultados de desempenho obtidos e suas análises, e as considerações finais do capítulo.

4.1 SIMULADOR DE *CACHE* NORMAL E RECONFIGURÁVEL

O simulador de *cache* utilizado foi uma adaptação do MSCSim (Multilevel and Split *Cache* Simulator) [Mendes 2006]. As adaptações foram feitas para permitir que o usuário escolha simular uma *cache* normal ou uma *cache* reconfigurável com a arquitetura proposta no capítulo 3. A simulação da *cache* normal não sofreu modificações com relação ao MSCSim original. A opção de *cache* reconfigurável foi adicionada para atender as necessidades da pesquisa.

Assim, como descrito no capítulo 3, a arquitetura proposta de *cache* reconfigurável funciona com base na função objetivo gerada pelo Módulo de Determinação da Função Objetivo. Assim que a função é processada no Módulo de Otimização e os parâmetros arquiteturais da *cache* são disponibilizados para o Módulo de Gerenciamento de Parâmetros de Reconfiguração, este módulo armazena esses valores para que o Módulo de Armazenamento Reconfigurável possa assumir a nova configuração sempre que a carga de trabalho correspondente à função objetivo seja utilizada.

Quando um usuário escolhe simular uma *cache* reconfigurável, ele deve inicialmente definir os valores máximos de cada parâmetro (grau de associatividade, número de *slots* e tamanho do bloco) assim como o tamanho máximo que a *cache* poderá assumir. Esses valores que o usuário definiu serão utilizados pelo método de otimização discreta como restrições.

Nessa versão do simulador apenas a política de substituição FIFO (First In First Out) foi implementada para facilitar e acelerar a obtenção de resultados, já que não é foco desta pesquisa avaliar os impactos causados por diferentes tipos de políticas de substituição. As

cache simuladas (tanto a normal quanto a *cache* reconfigurável proposta) possuem grau de associatividade, número de *slots*, tamanho da palavra, tamanho do bloco, tempo de acesso à memória principal, e tempo de acesso à memória *cache* parametrizados, podendo ser definidos pelo usuário.

Foi escolhido o MSCSim pelo fato de já ser um simulador estável, verificado e utilizado por várias pessoas, inclusive como ferramenta de auxílio no aprendizado de hierarquia de memórias nos laboratórios do curso de Ciência da Computação da PUC-Minas, e ainda pelo fato do código fonte estar disponível.

4.2 CARGAS DE TRABALHO

No início da pesquisa foram utilizadas cargas de trabalho sintéticas, mas para dar uma maior credibilidade aos resultados, optou-se por utilizar cargas de trabalho reais.

As cargas de trabalho utilizadas nas simulações foram obtidas no *BYU Trace Distribution Center* [BYU 2007]. Neste site podem ser encontradas várias cargas de trabalhos reais de benchmarks. As cargas que foram utilizadas nessa pesquisa são as cargas do SPECfp (Standard Performance Evaluation Corporation Floating Point) [SPEC 2008]. O SPEC é um benchmark muito utilizado para testes de desempenho computacional de um computador que trabalha com valores inteiros e de ponto flutuante. Esse benchmark é utilizado principalmente para avaliação de módulos de processadores e memória, pois trabalham muito com operações com esses tipos de valores [Henning 2000].

As cargas obtidas através do *BYU Trace Distribution Center* contém campos a mais do que os necessários para a simulação e por isso foi feita uma redução nos dados das cargas para se adaptarem ao simulador. No simulador só serão utilizadas as informações de endereço de acesso e os acessos à dados, pelo fato de desejarmos apenas simulações de *cache* de dados (para facilitar nas avaliações e verificações dos resultados), logo um conversor de *memory trace* foi desenvolvido para eliminar as informações desnecessárias.

Foram escolhidas as cargas do SPEC pelo fato de ele já ser um *benchmark* famoso e muito utilizado em pesquisas de memórias [SPEC 2008] [Carvalho 2004]. Na **Tabela 1** estão as principais características dos traces utilizados na pesquisa. O número de acessos descrito na **Tabela 1** representa o número total após a eliminação das informações desnecessárias para nossas simulações.

Tabela 1 – Características dos Traces Utilizados

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

Como é possível observar na **Tabela 1,** foi mantido um número de acessos mais ou menos semelhante para todos os traces a fim de tornar as comparações mais fáceis.

4.3 VERIFICAÇÃO DOS PRINCIPAIS MÓDULOS DA ARQUITETURA PROPOSTA

Encontram-se nessa seção os métodos, técnicas e resultados obtidos na verificação e validação dos principais módulos da arquitetura proposta.

4.3.1 Módulo de Determinação da Função Objetivo

Em nossas verificações, o Módulo de Determinação da Função Objetivo utilizou Planejamento de Experimentos para encontrar as funções objetivo de cada memory trace do SPEC.

Para realização do Planejamento de Experimentos foram utilizados alguns parâmetros fixos e alguns parâmetros que poderiam ter seus valores variados. A **Tabela 2** mostra os parâmetros e valores escolhidos para não variar nos experimentos e a **Tabela 3** mostra os parâmetros variáveis.

Tabela 2 – Parâmetros fixos da *cache*

Parâmetro	Valor
Política de Substituição	FIFO
Tempo de Acesso à memória *cache*	5 ns
Tempo de acesso à memória principal	100 ns
Tipo de Acesso	Seqüencial
Política de escrita	Write Through

Foram escolhidos apenas três parâmetros variáveis para reduzir o número de simulações necessárias para encontrar uma função objetivo que caracterizasse cada carga de trabalho. Com três parâmetros variáveis são necessárias apenas oito (2^3) simulações para achar cada função objetivo.

Tabela 3 – Valores mínimos e máximos dos parâmetros da *cache*

Parâmetro	Valor Mínimo	Valores Intermediários	Valor Máximo
Grau de Associatividade (x_1)	1	2 ... 4 ... 8 ... 16 ... 32	64
Número de *Slots*(x_2)	1	2 ... 4 ... 8 ... 16 ... 32	64
Tamanho do Bloco (palavras) (x_3)	1	2 ... 4 ... 8 ... 16 ... 32	64

Os parâmetros variáveis foram escolhidos pelo fato de que uma mudança em seu valor causa um impacto maior na taxa de acerto, o que torna a otimização mais atraente. Os valores dos parâmetros fixos foram escolhidos de uma maneira que facilitasse o entendimento e a comparação dos resultados obtidos a partir das simulações.

Na **Tabela 4** estão as funções objetivo produzidas pelo módulo 1, que caracterizam cada uma das cargas de trabalho utilizadas, baseadas na taxa de acerto. Nas funções objetivo, x_1 representa o grau de associatividade, x_2 representa o número de *slots* e x_3 representa o tamanho do bloco.

Tabela 4 – Funções objetivo dos traces produzidas pelo Módulo 1, utilizando o Planejamento de Experimentos 2^k

Trace	Função Objetivo
applu	$F(x) = 53,38 + 20,74x_1 + 12,64x_2 + 14,29x_3 - 6,80x_1x_2 + 0,25x_1x_3 + 5,14x_2x_3 - 5,20x_1x_2x_3$
art	$F(x) = 59,85 + 19,53x_1 + 13,75x_2 + 9,81x_3 - 4,35x_1x_2 - 0,19x_1x_3 + 2,84x_2x_3 - 4,40x_1x_2x_3$
galgel	$F(x) = 58,62 + 20,54x_1 + 13,98x_2 + 12,17x_3 - 3,32x_1x_2 - 0,60x_1x_3 + 2,22x_2x_3 - 5,95x_1x_2x_3$
mesa	$F(x) = 62,86 + 20,42x_1 + 11,70x_2 + 10,03x_3 - 7,87x_1x_2 - 1,14x_1x_3 + 2,27x_2x_3 - 2,09x_1x_2x_3$
mgrid	$F(x) = 57,48+ 20,92x_1 + 13,25x_2 + 10,88x_3 - 8,11x_1x_2 + 0,72x_1x_3 + 4,14x_2x_3 - 3,78x_1x_2x_3$
swim	$F(x) = 55,08+ 21,17x_1 + 11,97x_2 + 13,54x_3 - 4,67x_1x_2 - 0,26x_1x_3 + 3,95x_2x_3 - 5,96x_1x_2x_3$
wupwise	$F(x) = 55,01+ 19,65x_1 + 13,90x_2 + 14,04x_3 - 4,16x_1x_2 + 1,25x_1x_3 + 3,28x_2x_3 - 5,89x_1x_2x_3$
Sintético	$F(x) = 31,44+ 7,31x_1 + 5,31x_2 + 15,31x_3 - 4,06x_1x_2 + 0,69x_1x_3 + 1,19x_2x_3 - 1,19x_1x_2x_3$

As funções objetivo da **Tabela 4** nos mostram a influência que cada parâmetro tem na taxa de acerto de uma memória *cache*. É possível perceber que o grau de associatividade (x_1) de uma *cache* é um parâmetro muito importante, pois altera significantemente a taxa de acerto de uma memória *cache*, no caso dos testes que realizamos. Também é possível notar que a significância do número de *slots* e do tamanho do bloco (x_2 e x_3 respectivamente) varia de acordo com o trace. Nos traces art, galgel, mesa e mgrid o número de *slots* é um pouco mais significativo que o tamanho do bloco.

Embora o grau de associatividade tenha sido o valor mais significante nos traces do SPEC, no trace sintético o valor mais significante foi o tamanho do bloco. Isso ocorreu pelo fato do trace sintético possuir alta localidade espacial.

As funções obtidas pelo Módulo de Determinação da Função Objetivo retratam bem o comportamento dos acessos de cada memory trace permitindo que o módulo de Otimização obtenha um resultado bem próximo do ideal, como será visto a seguir.

4.3.2 Módulo de Otimização

Para a verificação do Módulo de Otimização foi implementado nos testes o algoritmo de Otimização Discreta descrito em [Ekel 2006]. O algoritmo utiliza uma abordagem que soluciona problemas de otimização com coeficientes fuzzy, em funções objetivo que possuem restrições. Este algoritmo permite obter uma solução que se aproxima da solução ótima utilizando um número pequeno de passos.

Assim que é requisitada a otimização de uma função objetivo, o algoritmo inicia as variáveis com o menor valor possível (no nosso caso valor 1), e começa a sua execução. A cada iteração ele multiplica cada variável por dois e obtém os valores das funções objetivo para cada uma das variáveis incrementadas, ou seja, três valores de funções objetivo. As funções são comparadas com a função inicial gerando um delta da variação. O maior dos três deltas encontrados corresponde à variável que deve continuar com o novo valor (multiplicado por 2), sendo que as outras variáveis voltam a ter seus valores correspondentes aos valores da iteração anterior. O algoritmo repete esses passos até que as restrições sejam atingidas.

Para verificação da eficiência do algoritmo, foi criada uma carga de trabalho sintética com 200 acessos à memória e dela, a partir da utilização de um Planejamento de experimentos foi obtida uma função que a caracterizava. A função objetivo que caracteriza essa carga de trabalho sintética está descrita na **Tabela 4**. Na **Tabela 5** podemos ver os resultados obtidos

através da técnica de otimização e a sua comparação com o ideal para diferentes tamanhos de *cache*. Para a obtenção dos valores ideais foram realizadas todas as simulações possíveis.

Através do método de otimização podemos obter os parâmetros que mais se aproximam dos ideais. Como o método é muito simples e em poucas iterações ele já encontra uma solução, em alguns casos essa solução não é a ótima. Quanto mais iterações forem realizadas maiores são as chances de encontrar a solução ótima.

Tabela 5 - Comparação entre parâmetros obtidos através do Módulo de Otimização e os ideais

Tamanho da *Cache* (palavras)	**4096**		**8192**		**16384**	
	Obtida pelo Módulo	**Ideais**	**Obtida pelo Módulo**	**Ideais**	**Obtida pelo Módulo**	**Ideais**
Configurações (GA,NS,TB)	16,8,32	4,16,64 16,4,64	16,16,32	4,32,64 8,16,64 16,8,64 32,4,64 64,2,64	16,16,64	4,64,64 8,32,64 16,16,64 32,8,64 64,4,64
Tempo Médio (ns)	55	51	55	49	49	49
Taxa de Acerto (%)	50	54	50	56	56	56
Tempo de Acesso Total (ns)	11000	10300	11000	9800	9800	9800
Diferença em % da Ta	4		6		0	

Na **Tabela 5** as configurações encontradas representam o valor dos parâmetros (grau de associatividade, número de *slots* e tamanho do bloco respectivamente). A diferença entre a solução obtida através do Modulo de Otimização e a solução ideal, no nosso caso, chegou a ser de 6%, mas na média a diferença ficou em 3,33%. Apesar da diferença na taxa de acerto parecer ser grande, o tempo total de acesso não foi muito maior.

A grande vantagem da implementação desse método no Modulo de Otimização está na sua simplicidade e na rapidez para encontrar uma solução muito próxima do ótimo. Uma busca exaustiva pode demorar muito ou até ser inviável em casos onde o número de combinações é muito grande. Uma sugestão de trabalho futuro é o melhoramento do algoritmo de otimização e/ou a utilização de outras técnicas de otimização existentes.

A utilização do método de otimização discreta, permite a obtenção de uma solução bem próxima da ideal em um curto período de tempo o que pôde ser verificado através dos testes realizados.

4.4 RESULTADOS DE DESEMPENHO

Nas experiências de avaliação e análise de desempenho, foram realizadas comparações entre memórias *cache* de alguns processadores comerciais e os seus equivalentes reconfiguráveis, utilizando as cargas do SPEC. Para que as experiências fossem válidas, calculou-se o tamanho de cada *cache* em palavras, para que a *cache* reconfigurável não ultrapassasse o tamanho da *cache* comercial. Os resultados detalhados obtidos podem ser encontrados no fim desta dissertação. A **Tabela 6** mostra as configurações das *cache* dos processadores comercias utilizados nas comparações. Apenas as *cache* de dados foram representadas devido ao memory trace utilizado só possuir acesso à dados.

Tabela 6 – Configuração das *cache* de dados dos processadores comerciais:

Processador	Grau de Associatividade	Número de *Slots*	Tamanho do Bloco (palavras)	Tamanho da *cache* (palavras)
Pentium M	8	64	16	8192
Itanium 2	4	64	16	4096
UltraSparc III	4	512	8	16384
Pentium 4 90 nm	8	32	16	4096
AMD Hammer	2	512	16	16384

Uma primeira comparação entre as *cache* de processadores comerciais e as *cache* reconfiguráveis equivalentes foi feita para cada memory trace da **Tabela 4**, exceto a carga sintética. As **Figuras 12** a **18** nos permitem observar os resultados obtidos para cada memória *cache* simulada.

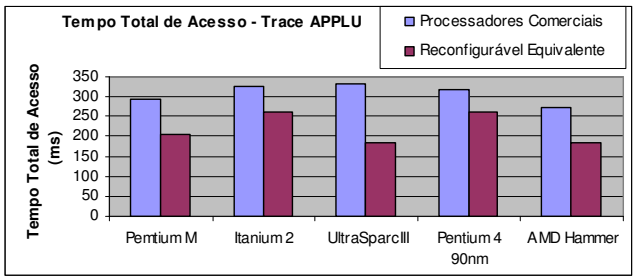

Figura 12 - Tempo Total de Acesso – Trace Applu

Tabela 7 - Speed ups para o trace Applu:

Trace APPLU	
Processador Comercial	**Speed up**
Pentium M	1,4300
Itanium 2	1,2553
UltraSparc III	1,7928
Pentium 4 90 nm	1,2234
AMD Hammer	1,4758

O primeiro trace simulado foi o Applu. É possivel notar pelo gráfico da **Figura 12** que o tempo de todas as *cache* reconfiguráveis obtiveram foi menor que as *cache* comerciais equivalentes. A maior diferença nos tempos obtidos foi a da *cache* do processador UltraSparcIII e de sua *cache* reconfigurável equivalente. A **Tabela 7** contém os speed ups das *cache* reconfiguráveis para o trace Applu. Pode-se perceber que a *cache* reconfigurável equivalente à *cache* do UltraSparcIII conseguiu obter uma melhora em seu tempo de 79,28%.

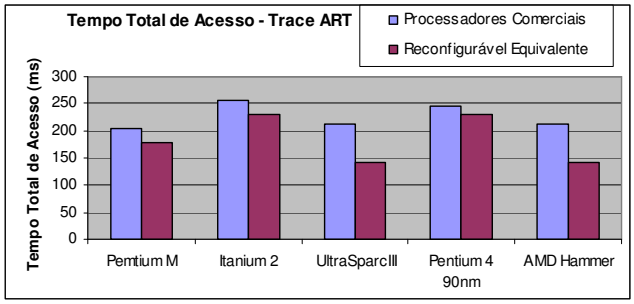

Figura 13 - Tempo Total de Acesso – Trace Art

Tabela 8 - Speed ups para o trace art:

Trace ART	
Processador Comercial	**Speed up**
Pentium M	1,1423
Itanium 2	1,1072
UltraSparc III	1,4971
Pentium 4 90 nm	1,0644
AMD Hammer	1,4881

O gráfico da **Figura 13**, assim com a **Figura 12**, também mostra uma redução de tempo para todas as *cache* reconfiguráveis. Porém, para o trace Art a *cache* reconfigurável equivalente ao Pentium 4 não obteve uma melhora tão significativa, ficando apenas 6% mais rápida que a comercial. Para o trace Art, a *cache* equivalente à do UltraSparcIII conseguiu um melhor resultado, chegando a um speed up de 49,71%, mas ainda inferior ao do trace Applu.

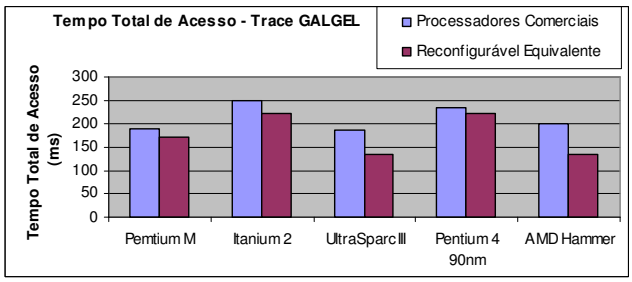

Figura 14 - Tempo Total de Acesso – Trace Galgel

Tabela 9 - Speed ups para o trace galgel:

Trace GALGEL	
Processador Comercial	**Speed up**
Pentium M	1,1036
Itanium 2	1,1214
UltraSparc III	1,3637
Pentium 4 90 nm	1,0472
AMD Hammer	1,4733

Para o trace Galgel, os resultados das *cache* reconfiguráveis ainda continuam sendo melhores, obtendo menores tempos. Entretanto, houve uma redução nos speed ups. A *cache* com maior speed up foi a *cache* reconfigurável equivalente à do AMD Hammer, com uma melhora de 47,33%.

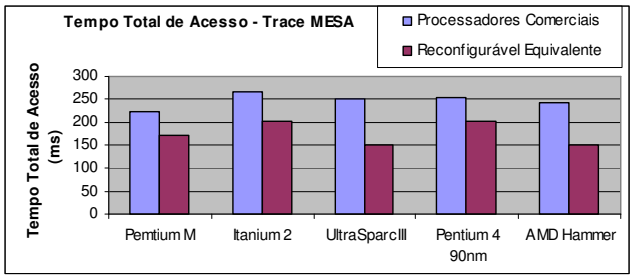

Figura 15 - Tempo Total de Acesso – Trace Mesa

Tabela 10 - Speed ups para o trace mesa:

Trace MESA	
Processador Comercial	**Speed up**
Pentium M	1,3067
Itanium 2	1,3087
UltraSparc III	1,6825
Pentium 4 90 nm	1,2495
AMD Hammer	1,4835

Apesar do trace Galgel ter sido um dos que apresentou os menores speed ups, a redução dos tempos no trace Mesa foi muito boa, estando entre as melhores dentre os traces simulados, tendo uma redução em até 2/5 do tempo no caso do UltraSparcIII. Essa redução garantiu um speed up de 68,25% quando comparada a *cache* reconfigurável proposta com a *cache* comercial.

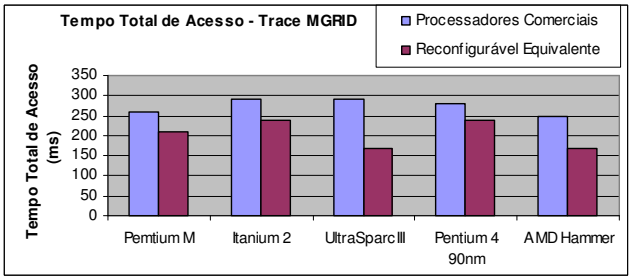

Figura 16 - Tempo Total de Acesso – Trace Mgrid

Tabela 11 - Speed ups para o trace mgrid:

Trace MGRID	
Processador Comercial	**Speed up**
Pentium M	1,2215
Itanium 2	1,2267
UltraSparc III	1,7213
Pentium 4 90 nm	1,1768
AMD Hammer	1,4835

A **Figura 16** mostra que os tempos das memórias reconfiguráveis também foram menores em todas as simulações para o trace Mgrid. Os speed ups foram maiores que dos traces anteriores e novamente a *cache* reconfigurável equivalente à *cache* do processador UltraSparcIII obteve o melhor desempenho.

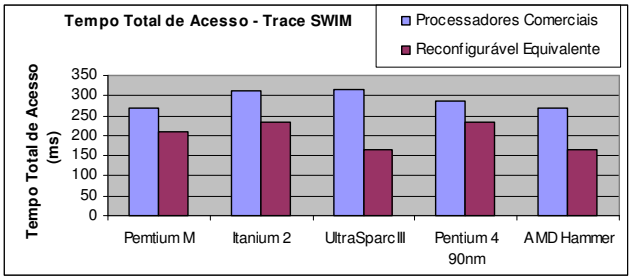

Figura 17 - Tempo Total de Acesso – Trace Swim

Tabela 12 - Speed ups para o trace swim:

Trace SWIM	
Processador Comercial	**Speed up**
Pentium M	1,2866
Itanium 2	1,3332
UltraSparc III	1,9041
Pentium 4 90 nm	1,2267
AMD Hammer	1,6166

O trace Swim, cujo gráfico de tempo total de acesso é apresentado na Figura 17, foi o que apresentou maior ganho dentre os traces simulados. Além de todas as memórias reconfiguráveis obterem tempos inferiores, novamente a *cache* reconfigurável equivalente à do UltraSparcIII obteve o maior speed up, com uma melhora de 90,41% no tempo.

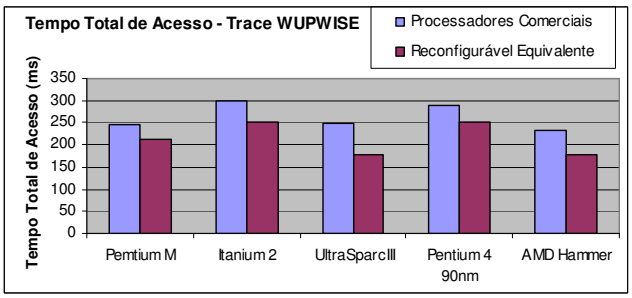

Figura 18 - Tempo Total de Acesso – Trace Wupwise

Tabela 13 - Speed ups para o trace wupwise:

Trace WUPWISE	
Processador Comercial	**Speed up**
Pentium M	1,1428
Itanium 2	1,1883
UltraSparc III	1,4062
Pentium 4 90 nm	1,1515
AMD Hammer	1,3093

O trace Wupwise (**Figura 18**) assim como os traces analisados anteriormente, também mostrou um ganho no desempenho. A *cache* equivalente à do UltraSparcIII novamente foi a que obteve maior ganho.

Apesar da *cache* do UltraSparcIII ser maior que as *cache* do Pentium M, Itanium 2 e Pentium 4, ela apresentou um tempo total maior em alguns traces (**Figuras 12, 16** e **17**). A *cache* reconfigurável equivalente à *cache* do UltraSparcIII conseguiu diminuir o tempo total, obtendo o melhor tempo total juntamente com a *cache* reconfigurável equivalente à *cache* do AMD Hammer. Isso nos leva à conclusão de que nosso modulo de otimização consegue encontrar parâmetros mais próximos dos ideais conforme o tamanho da *cache*. Quanto maior o tamanho da *cache* maior a chance de se encontrar os parâmetros próximos dos ideais.

Tabela 14 – Speed ups dos diferentes traces simulados:

Speed ups dos diferentes Traces							
Traces / Proc. Comerciais	**Applu**	**Art**	**Galgel**	**Mesa**	**Mgrid**	**Swim**	**Wupwise**
Pentium M	1,4300	1,1423	1,1036	1,3067	1,2215	1,2866	***1,1428***
Itanium 2	1,2553	1,1072	1,1214	1,3087	1,2267	1,3332	1,1883
UltraSparc III	**1,7928**	**1,4971**	1,3637	**1,6825**	**1,7213**	**1,9041**	**1,4062**
Pentium 4 90 nm	***1,2234***	***1,0644***	***1,0472***	***1,2495***	***1,1768***	***1,2267***	1,1515
AMD Hammer	1,4758	1,4881	**1,4733**	1,4835	1,4835	1,6166	1,3093

Na **Tabela 14** estão os speed ups de todos os traces simulados. Os resultados marcados de negrito e sublinhados foram os melhores resultados obtidos pela arquitetura proposta. Os resultados em itálico, são os piores resultados obtidos. Como é possível observar, a *cache* reconfigurável equivalente à *cache* do processador UltraSparcIII obteve maior speed up para quase todos os traces, exceto para o Galgel. Os speed ups variaram entre 36% e 90%.

Para todos os traces, exceto para o Wupwise, a *cache* reconfigurável equivalente a *cache* do Pentium 4 obteve speed ups inferiores aos demais. Para o trace Wupwise, a *cache* reconfigurável equivalente a do Pentium M obteve o pior speed up.

Os testes com cargas de trabalho compostas (mais de um trace) foram realizados utilizando uma arquitetura extrínseca, como da **Figura 9**, para que a simulação fosse facilitada. Os módulos 1 e 2 trabalharam *offline* e por isso já possuíam os valores necessários para configurar o módulo de armazenamento reconfigurável (módulo 4). O módulo de armazenamento se reconfigura a cada nova carga de trabalho, pois já possui as informações resultantes dos primeiros módulos, armazenados no módulo 3. Os resultados desses testes estão apresentados nas **Figuras 19, 20** e **21** e nas **tabelas 15, 16** e **17**.

As **tabelas 15, 16** e **17** mostram os *speed ups* obtidos a partir da comparação das *cache* reconfiguráveis com as respectivas *cache* dos processadores comerciais.

Nos primeiros testes foram utilizadas apenas duas cargas de trabalho diferentes (applu e wupwise). Na **Figura 19** podemos ver que a *cache* reconfigurável equivalente conseguiu reduzir o tempo total de acesso em até aproximadamente 218 milisegundos (no caso do UltraSparc III), o que significa um speed up de 60,35%.

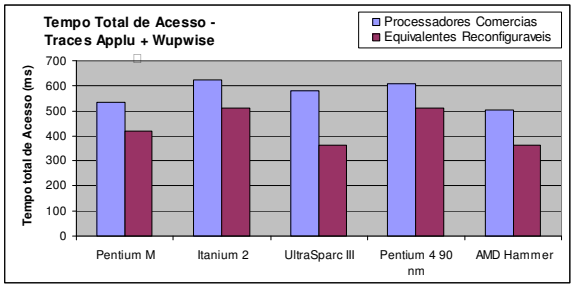

Figura 19 - Tempo de Acesso total para a carga de trabalho applu+wupwise

Tabela 15 – Speed up das *cache* reconfiguráveis (traces applu+wupwise):

Processador Comercial	Speed up
Pentium M	1,2836
Itanium 2	1,2224
UltraSparc III	1,6035
Pentium 4 90 nm	1,1882
AMD Hammer	1,3943

Em testes realizados com uma carga composta por quatro memory traces diferentes (art, mesa, mgrid e swim) foram encontrados diferentes resultados que estão destacados na **Figura 20**. O tempo total das *cache* reconfiguráveis ainda continuou menor que os tempos dos processadores comerciais. Os *speed ups* dos processadores Pentium M e Pentium 4 na **Tabela 16** foram pouco menores que os da **Tabela 15**. Isso pode ser explicado pelo fato de que os processadores reconfiguráveis equivalentes à eles possuíram pior desempenho nos traces Art e Mgrid como podemos ver nas **figuras 13** e **16**.

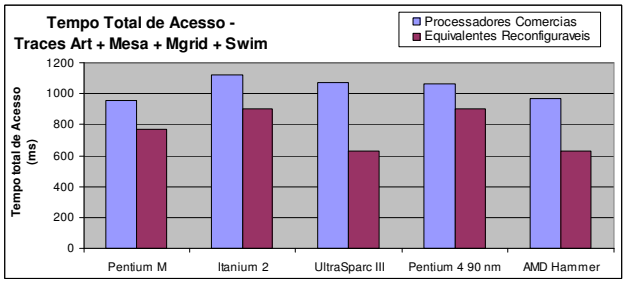

Figura 20 - Tempo de Acesso total para a carga de trabalho art+mesa+mgrid+swim

Tabela 16 – Speed up das *cache* reconfiguráveis (traces art+mesa+mgrid+swim):

Processador Comercial	Speed up
Pentium M	1,2398
Itanium 2	1,2422
UltraSparc III	1,7095
Pentium 4 90 nm	1,1774
AMD Hammer	1,5491

Testes realizados utilizando uma carga composta por seis memory traces, mas apenas 3 diferentes entre si (art, applu, galgel) também geraram resultados satisfatórios (**Figura 21**). Novamente o tempo total de acesso à *cache* dos processadores comerciais foi maior que o das *cache* reconfiguráveis. E também novamente foi encontrado um *speed up* maior que 56% no caso do processador UltraSparc III.

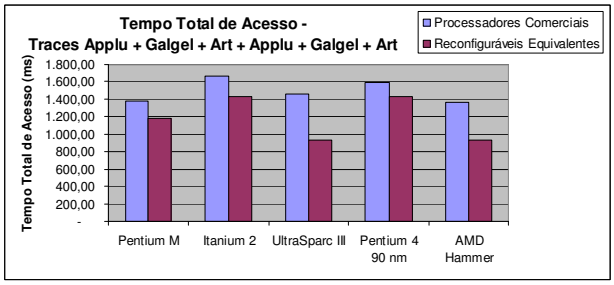

Figura 21 - Tempo de Acesso total para a carga de trabalho applu/galgel/art/applu/galgel/art

Tabela 17 – Speed up das *cache* reconfiguráveis (traces applu/galgel/art/applu/galgel/art):

Processador Comercial	Speed up
Pentium M	1,1658
Itanium 2	1,1627
UltraSparc III	1,5644
Pentium 4 90 nm	1,1142
AMD Hammer	1,4683

Para verificar as vantagens de se utilizar uma *cache* que possua dois módulos de armazenamento reconfigurável (arquitetura da **Figura 11** do **capítulo 3**) foram realizados testes com um carga de trabalho composto de seis *traces* sem que nenhum deles fosse repetido. A cada mudança de trace, o novo trace é enviado para um Módulo de Armazenamento Reconfigurável diferente. A **Figura 22** ilustra como seria a distribuição da carga de trabalho entre os dois módulos.

Para uma melhor comparação foi utilizada uma *cache* com tamanho igual à de um Pentium 4 e à de um UltraSparcIII, sendo que cada módulo de armazenamento reconfigurável possui metade do tamanho total. A **Figura 23** mostra o tempo total obtido para o Pentium 4.

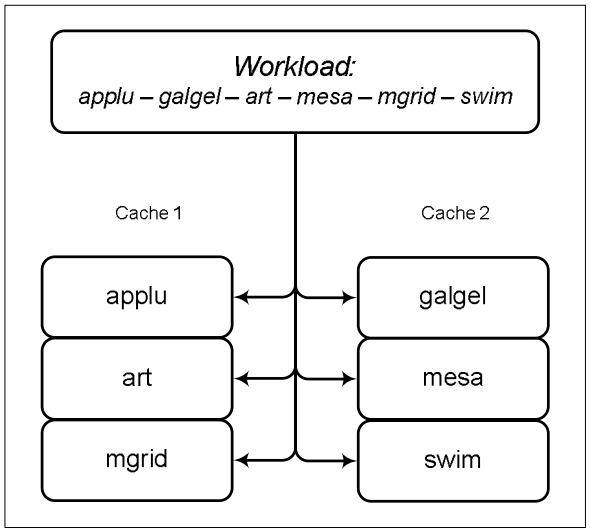

Figura 22 - Distribuição do carga de trabalho

O *speed up* obtido pela *cache* com dois módulos de armazenamento chega a ser de 1,6250 em relação à *cache* reconfigurável com um único módulo de armazenamento e de 1,8984 com relação à *cache* do Pentium 4.

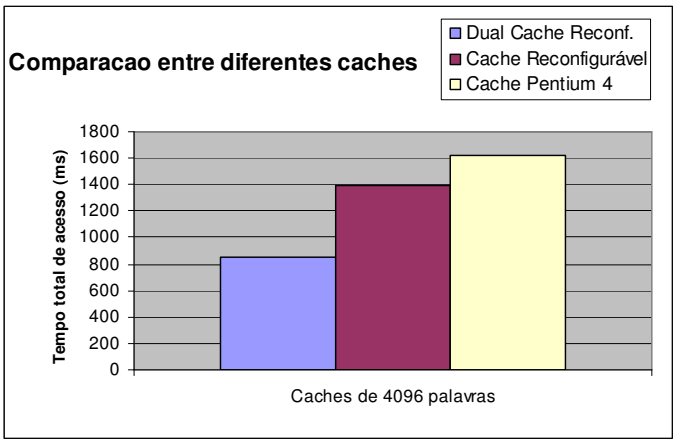

Figura 23 - Comparações do Pentium 4 (90nm) com uma *cache* com dois módulos de armazenamento reconfigurável.

Quando comparamos uma *cache* com dois módulos de armazenamento reconfigurável com a *cache* do processador UltraSparcIII, a diferença dos tempos é ainda maior como podemos ver na **Figura 24.**

O *speed up* obtido pela *cache* com dois módulos de armazenamento reconfigurável agora chega a ser de 1,5941 em relação à *cache* reconfigurável com apenas um módulo de armazenamento reconfigurável e de 2,6717 em relação à *cache* do processador UltraSparcIII.

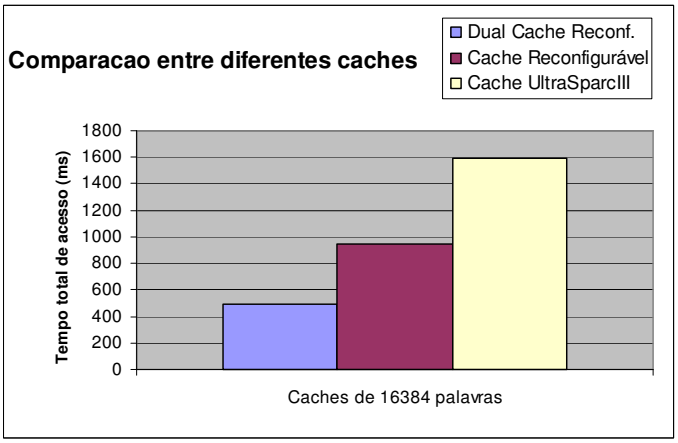

Figura 24 - Comparações do UltraSparcIII com uma *cache* com dois módulos de armazenamento reconfigurável.

Tabela 18 - Speed up da *cache* com 2 Módulos de Armazenamento Reconfigurável:

Cache	Speed up
***Cache* Reconfigurável (4096 palavras)**	1,6250
***Cache* Reconfigurável (16384 palavras)**	1,5941
Pentium 4 90 nm	1,8984
UltraSparc III	2,6717

A arquitetura de *cache* reconfigurável, com 2 módulos de armazenamento reconfigurável, obteve melhor tempo em relação à *cache* reconfigurável com apenas 1 módulo. E a *cache* reconfigurável ainda possui melhor tempo que a *cache* do processador comercial.

Tabela 19 - Speed ups das cargas de trabalho compostas:

Speed ups dos diferentes Traces				
Traces / Proc. Comerciais	**Applu + Wupwise**	**Art + Mesa + Mgrid + Swim**	**Applu + Galgel + Art + Applu + Galgel + Art**	**Applu + Galgel + Art + Mesa + Mgrid + Swim**
Pentium M	1,2836	1,2398	1,1658	-
Itanium 2	1,2224	1,2422	1,1627	-
UltraSparc III	**1,6035**	**1,7095**	**1,5644**	**2,6717**
Pentium 4 90 nm	***1,1882***	***1,1774***	***1,1142***	1,8984
AMD Hammer	1,3943	1,5491	1,4683	-
***Cache* Reconfigurável (4096 palavras)**	-	-	-	1,6250
***Cache* Reconfigurável (16384 palavras)**	-	-	-	***1,5941***

Na **Tabela 19** estão os speed ups das simulações com cargas de trabalho compostas de traces diferentes. A *cache* reconfigurável equivalente à *cache* do UltraSparcIII obteve melhor desempenho em todas as simulações. Chegando a um speed up de 2,67 no caso da *cache* dual

reconfigurável. A *cache* reconfigurável equivalente a *cache* do Pentium 4 foi a que obteve menor speed up.

4.5 CONSIDERAÇÕES FINAIS

Com os resultados obtidos podemos concluir que foram verificados e validados os módulos de Determinação da Função Objetivo e o de Otimização. Além disso, podemos também afirmar que quanto maior for a memória *cache* maior a chance do Método de Otimização encontrar parâmetros próximos do ideal.

Resultados como esses mostram que a utilização de planejamento de experimentos e técnicas de otimização em conjunto com a reconfiguração são boas opções quando se deseja aumentar o desempenho de algum dispositivo. A possibilidade de reconfiguração permite que a arquitetura proposta obtenha uma melhor adaptação às necessidades do sistema de computação. Já a utilização de técnicas de otimização e tomada de decisões propiciam à *cache* uma maior autonomia e inteligência.

Capítulo 5 - CONCLUSÕES

Neste capítulo são discutidos os principais resultados obtidos, são apresentadas as principais contribuições da pesquisa, com uma análise do alcance dos objetivos e metas e alguns trabalhos futuros.

5.1 DISCUSSÃO DOS RESULTADOS

Em nossas simulações e em nossos experimentos utilizamos *memory traces* do benchmark SPEC 200 CPU floating point. Optou-se por utilizar *traces* reais para dar maior credibilidade aos resultados, além de facilitar a comparação entre os diferentes resultados.

A arquitetura proposta nos permite observar o quanto o desempenho de arquiteturas inteligentes aumenta quando comparada com arquiteturas comuns. A possibilidade de se escolher uma configuração próxima do ótimo em tempo de execução é um dos fatores mais vantajosos da nossa arquitetura. Juntamente com a adição da inteligência, o aumento na autonomia da memória *cache* reconfigurável, torna ainda melhor o desempenho do sistema de computação. Essa autonomia permite que a memória se reconfigure com os parâmetros otimizados, sem que seja necessária a interferência humana, por exemplo. A adição de módulos que proporcionam inteligência e autonomia só trazem ganhos para a arquitetura.

Ainda existem pontos a serem estudados e melhorados, como por exemplo verificar qual o melhor momento para se realizar uma reconfiguração, verificar os impactos do *overhead* de reconfiguração, etc. Avaliações da arquitetura de maneira intrínseca não puderam ser feitos pelo fato de não ter sido possível (por limitação de tempo), a implementação da arquitetura em *hardware* reconfigurável.

Devido às limitações de tempo, a única meta que não foi concluída foi a de implementar a arquitetura em um dispositivo reconfigurável. As outras metas , assim como os objetivos foram todos atingidos. Foi possível construir um simulador de memória reconfigurável, que permite ao usuário simular memórias *cache* normais, e *cache* reconfiguráveis inteligentes e autônomas. Conseguimos projetar uma arquitetura autônoma e

inteligente que utiliza técnicas de otimização e de tomada de decisões. Além disso, o simulador juntamente com os *traces* reais nos permitiu realizar comparações de desempenho entre *cache* reconfiguráveis utilizando a arquitetura proposta e *cache* de processadores comerciais com tamanhos equivalentes.

Os resultados obtidos mostram que mesmo uma simples otimização, juntamente com a possibilidade de se possuir um hardware reconfigurável, pode aumentar e muito o desempenho de uma memória *cache*. Em todos os resultados a *cache* proposta obteve desempenho melhor, principalmente em *cache* maiores como a reconfigurável equivalente à do UltraSparcIII. Portanto, podemos dizer que o quanto maior for a memória *cache*, maior a chance do Módulo de Otimização encontrar parâmetros próximos dos ideais. No caso da utilização da dual *cache*, que possuía o mesmo tamanho em palavras da *cache* reconfigurável e da *cache* comercial, os resultados foram muito bons com a *cache* dual inteligente e autônoma, chegando a ficar quase 3 vezes mais rápida que a *cache* comercial.

A proposta de arquitetura aqui apresentada difere de todos os trabalhos correlatos que foram encontrados no estado da arte. Em nenhum deles existia a possibilidade de a arquitetura conseguir caracterizar cada *memory trace* que será utilizado. Muitos dos trabalhos correlatos podem ser utilizados ou incorporados na arquitetura proposta, podemos utilizar diferentes métodos de otimização, diferentes arquiteturas de *cache* reconfigurável e/ou diferentes tipos de parâmetros a serem reconfigurados.

5.2 PRINCIPAIS CONTRIBUIÇÕES

A principal contribuição desta dissertação, está na proposta e produção de uma arquitetura de memória cache reconfigurável autônoma e inteligente, capaz de se adaptar à diferentes cargas de trabalho, sem a interferência de agentes externos (por exemplo o homem).

Esta pesquisa possui várias outras contribuições. Uma delas é a inserção de um módulo de otimização que proporciona uma maior inteligência à arquitetura. Outra contribuição é o módulo de determinação dos parâmetros para otimização, que junto com o módulo de otimização permite à *cache* se reconfigurar de uma maneira mais autônoma.

A terceira contribuição é o módulo de gerenciamento de parâmetros de reconfiguração que permite que o módulo de armazenamento reconfigurável sempre possua uma configuração otimizada para cada carga de trabalho a ser executada.

Além dessas contribuições, podemos destacar também o fato de disponibilizar para futuros pesquisadores o simulador em Java, referências e bibliografias importantes para o estudo de memórias *cache* reconfiguráveis.

5.3 TRABALHOS FUTUROS

Existe uma grande quantidade de trabalhos futuros que podem surgir a partir desta pesquisa. Como exemplo podemos citar:

- Implementar a arquitetura proposta em um dispositivo reconfigurável. A implementação da arquitetura proposta em um dispositivo reconfigurável tornaria possível realizar testes reais, e não apenas simulações o que ajudaria na comparação com outros tipos de memória *cache*.
- A utilização de técnicas de otimização mais complexas para se obter parâmetros mais próximos dos ideais. Com a utilização de técnicas de otimização mais complexas seria possível obter resultados mais próximos dos ótimos, o que poderia aumentar o desempenho.
- Realizar uma análise mais complexa e mais detalhada dos resultados obtidos no Módulo de Determinação da Função Objetivo e no Módulo de Otimização.
- Utilização de técnicas diferentes para se caracterizar cada carga de trabalho. Procurar outras formas de se identificar (diferente do Planejamento de experimentos) cada carga de trabalho que será utilizada.
- Utilização de diferentes arquiteturas de memórias *cache* reconfiguráveis. Poderíamos utilizar diferentes arquiteturas de memórias *cache* reconfiguráveis com diferentes parâmetros reconfiguráveis para tentar achar uma combinação que produza maior desempenho.

Capítulo 6 - BIBLIOGRAFIA

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

[Asaduzzaman 2006] Asaduzzaman A.; Mahgoub I. *Cache* Optimization for Embedded Systems Running H.264/AVC Video Decoder. Department of Computer Science & Engineering. Computer Systems and Applications. pp.665- 672, 2006.

[Ballard 1982] Ballard, D. H. & Brown, C.M. Computer Vision. Englewood Cliffs, New Jersey: Prentice Hall, 1982. 523 p.

[Benitez 2006] Benitez, D. Moure, J. C. Rexachs, D. I. Luque, E. Evaluation of the field-programmable *cache*: performance and energy consumption. Conference On Computing Frontiers. Ischia, Italy SESSION: Special session on *cache* optimization. pp. 361 – 372. 2006.

[Brassard 1995] Brassard G. and Bratley P. Fundamentals of Algorithmics. Prentice Hall, Nova Yorque, 1995.

[BYU 2007] BYU Trace Distribution Center. Disponível em: http://tds.cs.byu.edu/tds/. Acessado em: 07/11/2007, 2007.

[*Cache*grind 2006] Valgrind: Tool Suite. Acessado em: 27/02/2007. http://valgrind.org/info/tools.html

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

[Carvalho 2005] Carvalho, M. B. Dissertação de Mestrado: Proposta e Desenvolvimento de uma Arquitetura de Memória *Cache* Reconfigurável. Orientador: Carlos Augusto Paiva da Silva Martins. Ano de Obtenção: 2005. PPGEE da PUC Minas.

[Chang 2002] Chang C.; Sheu J.; and Chen H. Reducing *Cache* Conflicts by Multi-Level *Cache* Partitioning and Array Elements Mapping. The Journal of Supercomputing. Volume 22, Issue 2 (June 2002). pp. 197 – 219, 2002.

[Chaves 2006] Chaves R., Kuzmanov G.K., Vassiliadis S., Sousa L. Reconfigurable Memory BasedAES Co-Processor. Proceedings of the 13th Reconfigurable Architectures Workshop (RAW), 2006.

[Chidester 2002] Chidester, M. George, A. Parallel simulation of chip-multiprocessor architectures. ACM Transactions on Modeling and Computer Simulation (TOMACS). Volume 12 , Issue 3. pp. 176 – 200, 2002.

[Corrêa 2005] Corrêa, F. R. C. Trabalho de Diplomação: Proposta de aplicação de soft processors em sistemas embutidos. Orientador: Henrique Cota de Freitas. DCC, PUC Minas.

[Eiras 1996] Eiras, S; Andrade, J. C. O uso do simplex modificado como estratégia de otimização em química analítica. Química Nova, v.19, n.1, pp. 25-29, 1996.

[Ekel 2006] Ekel, P. ; Schuffner Neto, F. . Algorithms of Discrete Optimization and Their Application to Problems with Fuzzy Coefficients. Information Sciences, Oxford-New York-Toronto, v. 176, n. 19, pp. 2846-2868, 2006.

[Ghosh 2004] Ghosh A.; Givargis T. *Cache* Optimization for Embedded Processor Cores: An Analytical Approach. ACM Transactions on Design Automation of Electronic Systems, Vol. 9, No. 4, October 2004, pp. 419–440.

[Goupy 1988] GOUPY, J. La méthode des plans d'expériences. Paris: Dunod, 1988. 303p.

[Gross 1998] Gross, J. and Yellen, J. Graph theory and its applications. Nova Yorque: CRC Press, 1998.

[Hammond 1997] Hammond L., Basem A. N., Olukotun K.. A Single-Chip Multiprocessor. Computer Magazine, IEEE Computer Society, vol.30, no.9, pp. 79-85, Sept., 1997.

[Hauck 1999] Hauck, S.; Li, Z.; and Schwabe, E. Configuration Compression for the Xilinx XC6200 FPGA. IEEE Transactions on Computer aided Design of Integrated Circuits and Systems, vol. 18, no. 8, August 1999. pp. 138-146.

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

[Hennessy 2006] Hennessy, J. L., Patterson, D. A. Computer Architecture: A Quantitative Approach, 4ª Edição (Inglês), 2006.

[Hillier 2001] Hillier F.; Lieberman G. Introduction to Operations Research. McGraw-Hill International Editions, 2001.

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

[Kusse 1998] Kusse, E; Rabaey, J. M.; Low-Energy Embedded FPGA Structures. International Symposium on Low Power Electronics and Design archive Proceedings. EECS Department, University of California at Berkeley. 1998.

[Lee 1994] Lee J.; Lee M.; Choi S.; Park M. Reducing *cache* conflicts in data *cache* prefetching ACM. SIGARCH Computer Architecture News Volume 22 , Issue 4 (September 1994) Special issue on input/output in parallel computer systems. pp. 71 – 77. 1994.

[Li 2001] Li, Y.; Henkel, J. A framework for estimating and minimizing energy dissipation of embedded HW/SW systems. The Morgan Kaufmann Systems On Silicon Series Section: Analysis and estimation. pp. 259 – 264. Readings in hardware/software co-design, 2001.

[Lien 2001] Lien, F. et al. A Hardware / Software Solution for Embeddable FPGA. IEEE Conference on Custom Integrated Circuits. pp 71-74, 2001.

[Martins 2003] Martins, C. A. P. S. et al. COMPUTAÇÃO RECONFIGURÁVEL: conceitos, tendências e aplicações. XXII Jornada de Atualização em Informática (JAI), SBC2003, Vol. 2, 2003, pp. 339 - 388.

[Mendes 2006] Mendes, J. L. D.; Coutinho, L. M. N.; Martins, C. A. P. S.; "MSCSim – Multilevel and Split Cache Simulator", 36th Frontiers in Education Conference (FIE). pp: 7-12, 2006.

[Mesquita 2001] Mesquita, D. et al. RECONFIGURAÇÃO PARCIAL E REMOTA DE CORES FPGAS. Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS. Dissertação de Mestrado em Computação.

[Mirabilis 2006] VisualSim, Architecture Exploration and Performance Analysis. Acessado em: 18/05/2007. http://www.mirabilisdesign.com/WebPages/products/mdi_products.htm

[Patterson 2005] Patterson, D. A., Hennessy, J. L. Organização e Projeto de Computadores – A Interface Hardware/Software, 3a Edição, Editora Campus, 2005.

[Pengyong 2006] Pengyong M.; Shuming C. MID: a Novel Coherency Protocol in Chip Multiprocessor, pp. 50, Sixth IEEE International Conference on Computer and Information Technology (CIT'06), 2006

[Press 2002] Press W.; Teukolsky S., Flannery B. e. Vetterling S. W. Numerical Recipes in C++: The Art of Scientific Computing. Cambridge University Press, 2002.

[Renovell 2001] Renovell, M. et al. IS-FPGA: A New Symmetric FPGA Architecture with Implicit SCAN. International Test Conference, pp. 924-931, 2001.

[Rochange 2005] Rochange C.; Sainrat P. A time-predictable execution mode for superscalar *pipeline* with instruction prescheduling. Proceedings of the 2nd conference on Computing frontiers. pp. 307 – 314, 2005.

[Sagahyroon 2004] Sagahyroon, A. Karunaratne, M. Impact of *cache* optimization techniques on energy management. Electrical and Computer Engineering. Volume: 4, pp. 1831- 1833, 2004.

[Sato 2000] Sato, T. Evaluating trace cache on moderate-scale processors.Computers and Digital Techniques, IEE Proceedings. Volume: 147, pp. 369–374. 2000.

[Shiue 1999] Shiue, W.; Chakrabarti, C. Memory exploration for low power embedded systems. In Proceedings of the Design Automation Conference. DAC, pp. 140-145, June, 1999.

[Smith 1982] Smith, A. J. *Cache* Memories, ACM Computing Surveys, pp. 473-530, 1982.

[SPEC 2008] SPEC – Standard Performance Evaluation Corporation. URL: http://www.spec.org. Acessado: 18/02/2008.

[Tanougast 2003] Tanougast, C. et al. Automated RTR Temporal Partitioning for Reconfigurable Embedded Real-Time System Design. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), IEEE Computer Society Press., 2003.

[Veidenbaum 1999] 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.

[Wilton 1996] Wilton, S.; Jouppi, N. CACTI: An enhanced *cache* access and cycle time model. IEEE J. Solid State Circ. 31, 5 (May), 677–688. 1996.

[Xilinx 2002] Xilinx Inc. Virtex 2.5V Field Programmable Gate Arrays (DS003). Virtex Series datasheet, 2002. Acessado: 18/05/2007. URL :www.xilinx.com/partinfo/ds003.pdf.

[Xilinx 2007] Xilinx Inc. Xilinx: The Programmable Logic Company. URL: http://www.xilinx.com.

Anexos

Resultados das Simulações

Nas tabelas a seguir estão os resultados detalhado das simulações realizadas.

Tabela 1 - Resultado da Simulação dos Traces Individuais:

TRACE APPLU										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	32 -4 -64	4 - 64 - 16	32 - 4 - 32	4 - 512 - 8	64 - 4 - 64	8 - 32 - 16	32 - 4 - 32	2 - 512 - 16	64 - 4 - 64
Taxa de Acerto (%)	76,67	85,19	73,41	79,84	73,04	87,18	74,21	79,84	78,69	87,18
Tempo Medio (ns)	28,32	19,80	31,58	25,16	31,95	17,82	30,78	25,16	26,31	17,82
Tempo Total (ns)	293181445	205028545	327005445	260501045	330828845	184527445	318707545	260501045	272323945	184527445

TRACE ART										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	32 - 4 - 64	4 - 64 - 16	32 - 4 - 32	4 - 512 - 8	32 - 8 - 64	8 - 32 - 16	32 - 4 - 32	2 - 512 - 16	32 - 8 - 64
Taxa de Acerto (%)	85,32	87,78	80,41	82,79	84,44	91,27	81,36	82,79	84,56	91,27
Tempo Medio (ns)	19,67	17,22	24,58	22,20	20,56	13,73	23,64	22,20	20,43	13,73
Tempo Total (ns)	203990775	178573775	254888075	230216875	213166675	142387475	245052575	230216875	211886875	142387475

TRACE GALGEL										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	32 - 8 -32	4 - 64 - 16	16 - 08 - 32	4 - 512 - 8	32 - 16 - 32	8 - 32 - 16	160832	2 - 512 - 16	32 - 16 - 32
Taxa de Acerto (%)	86,69	88,41	80,75	83,38	86,99	91,80	82,36	83,38	85,55	91,8
Tempo Medio (ns)	18,30	16,58	24,24	21,61	18,00	13,20	22,63	21,61	19,44	13,2
Tempo Total (ns)	188997510	171249110	250286610	223183310	185869710	136302210	233712910	223183310	200816410	136302210

TRACE MESA										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	16 -- 8 -- 64	4 - 64 - 16	16 -- 8 -- 32	4 - 512 - 8	16 - 16 - 64	8 - 32 - 16	16 -- 8 -- 32	2 - 512 - 16	16 - 16 - 64
Taxa de Acerto (%)	83,55	88,58	79,34	85,39	80,70	90,56	80,50	85,39	81,81	90,56
Tempo Medio (ns)	21,44	16,41	24,65	19,60	24,29	14,43	24,49	19,60	23,19	14,43
Tempo Total (ns)	222344160	170155160	265934460	203197860,0	251857460	149693460	253891860	203197860	240418560	149693460

TRACE MGRID										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	32 - 8 - 32	4 - 64 - 16	16 -- 8 -- 32	4 - 512 - 8	32 - 8 - 64	8 - 32 - 16	16 -- 8 -- 32	2 - 512 - 16	32 - 8 - 64
Taxa de Acerto (%)	80,14	84,66	76,83	82,04	77,00	88,73	77,98	82,04	80,87	88,73
Tempo Medio (ns)	24,85	20,34	28,16	22,95	27,99	16,26	27,01	22,95	24,12	16,26
Tempo Total (ns)	257528060	210824460	291848960	237921760	290074760	168523360	279974960	237921760	250004760	168523360

TRACE SWIM										
Cache	**Pemtium M**	**Reconfig. Equivalente**	**Itanium 2**	**Reconfig. Equivalente**	**UltraSparcIII**	**Reconfig. Equivalente**	**Pentium 4 90nm**	**Reconfig. Equivalente**	**AMD Hammer**	**Reconfig. Equivalente**
Tamanho Cache (palavras)	8192	8192	4096	4096	16384	16384	4096	4096	16384	16384
Parâmetros (GA-NS-TB)	8 - 64 - 16	32 - 8 - 32	4 - 64 - 16	32 - 4 - 32	4 - 512 - 8	32 - 8 - 64	8 - 32 - 16	32 - 4 - 32	2 - 512 - 16	32 - 8 - 64
Taxa de Acerto (%)	78,92	84,73	74,95	82,46	74,51	88,98	77,35	82,46	79,11	88,98

Tabela 2 – Resultados da simulação dos trace Applu e Wupwise:

Traces	applu+wupwise	applu+wupwise
Processador	**Pentium M**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	8192	8192
Total de acessos	20.671.696	20.671.696
Taxa de Acerto (%)	79,03	84,77
Tempo médio de acesso (ns)	25,963543	20,22667
Tempo de acesso total (ns)	536.710.467,98	418.119.573,33

Aumento na Taxa de Acerto		5,74
Reducao no tempo medio de acesso (ns)		5,736873
Reducao no tempo total de acesso (ns)		118.590.894,65
Speed Up	1,2836	28,36%

Traces	applu+wupwise	applu+wupwise
Processador	**Itanium 2**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	4096	4096
Total de acessos	20.671.696	20.671.696
Taxa de Acerto (%)	74,78	80,28
Tempo médio de acesso (ns)	30,215352	24,717163
Tempo de acesso total (ns)	624.602.571,08	510.945.679,52

Aumento na Taxa de Acerto		5,50
Reducao no tempo medio de acesso (ns)		5,498189
Reducao no tempo total de acesso (ns)		113.656.891,56
Speed Up	1,2224	22,24%

Traces	applu+wupwise	applu+wupwise
Processador	**UltraSparc III**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	16384	16384
Total de acessos	20.671.696	20.671.696
Taxa de Acerto (%)	76,95	87,51
Tempo médio de acesso (ns)	28,0442	17,4889801
Tempo de acesso total (ns)	579.721.179,03	361.526.879,98

Aumento na Taxa de Acerto		10,56
Reducao no tempo medio de acesso (ns)		10,55522
Reducao no tempo total de acesso (ns)		218.194.299,05
Speed Up	1,6035	60,35%

Traces	applu+wupwise	applu+wupwise
Processador	**Pentium 4 90 nm**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	4096	4096
Total de acessos	20.671.696	20.671.696
Taxa de Acerto (%)	75,63	80,28
Tempo médio de acesso (ns)	29,3682231	24,717163
Tempo de acesso total (ns)	607.090.979,98	510.945.679,52

Aumento na Taxa de Acerto		4,65
Reducao no tempo medio de acesso (ns)		4,6510601
Reducao no tempo total de acesso (ns)		96.145.300,46
Speed Up	1,1882	18,82%

Traces	applu+wupwise	applu+wupwise
Processador	**AMD Hammer**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	16384	16384
Total de acessos	20.671.696	20.671.696
Taxa de Acerto (%)	80,61	87,51
Tempo médio de acesso (ns)	24,38446656	17,4889801
Tempo de acesso total (ns)	504.068.279,85	361.526.879,98

Aumento na Taxa de Acerto		6,90
Reducao no tempo medio de acesso (ns)		6,89548646
Reducao no tempo total de acesso (ns)		142.541.399,87
Speed Up	1,3943	39,43%

Tabela 3 – Resultados da Simulação dos traces Art, Mesa, Mgrid, Swim:

Traces	art/mesa/mgrid/swim	art/mesa/mgrid/swim
Processador	**Pentium M**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	8192	8192
Total de acessos	41.475.182	41.475.182
Taxa de Acerto (%)	81,98	86,43
Tempo médio de acesso (ns)	23,013943	18,56316169
Tempo de acesso total (ns)	954.507.474,46	769.910.509,59

Aumento na Taxa de Acerto		4,45
Reducao no tempo medio de acesso (ns)		4,45078131
Reducao no tempo total de acesso (ns)		184.596.964,87
Speed Up	1,2398	23,98%

Traces	art/mesa/mgrid/swim	art/mesa/mgrid/swim
Processador	**Itanium 2**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	4096	4096
Total de acessos	41.475.182	41.475.182
Taxa de Acerto (%)	77,88	83,17
Tempo médio de acesso (ns)	27,1117076	21,82509554
Tempo de acesso total (ns)	1.124.463.007,04	905.199.809,69

Aumento na Taxa de Acerto		5,29
Reducao no tempo medio de acesso (ns)		5,28661206
Reducao no tempo total de acesso (ns)		219.263.197,35
Speed Up	1,2422	24,22%

Traces	art/mesa/mgrid/swim	art/mesa/mgrid/swim
Processador	**UltraSparc III**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	16384	16384
Total de acessos	41.475.182	41.475.182
Taxa de Acerto (%)	79,16	89,89
Tempo médio de acesso (ns)	25,83439199	15,1119001
Tempo de acesso total (ns)	1.071.486.109,64	626.768.807,01

Aumento na Taxa de Acerto		10,73
Reducao no tempo medio de acesso (ns)		10,72249189
Reducao no tempo total de acesso (ns)		444.717.302,63
Speed Up	1,7095	70,95%

Traces	art/mesa/mgrid/swim	art/mesa/mgrid/swim
Processador	**Pentium 4 90 nm**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	4096	4096
Total de acessos	41.475.182	41.475.182
Taxa de Acerto (%)	79,3	83,17
Tempo médio de acesso (ns)	25,697396	21,82509554
Tempo de acesso total (ns)	1.065.804.176,03	905.199.809,69

Aumento na Taxa de Acerto		3,87
Reducao no tempo medio de acesso (ns)		3,87230046
Reducao no tempo total de acesso (ns)		160.604.366,34
Speed Up	1,1774	17,74%

Traces	art/mesa/mgrid/swim	art/mesa/mgrid/swim
Processador	**AMD Hammer**	**Equivalente Reconfig.**
Tamanho da Cache em palavras	16384	16384
Total de acessos	41.475.182	41.475.182
Taxa de Acerto (%)	81,59	89,89
Tempo médio de acesso (ns)	23,4097588	15,1119001
Tempo de acesso total (ns)	970.924.006,81	626.768.807,01

Aumento na Taxa de Acerto		8,30
Reducao no tempo medio de acesso (ns)		8,2978587
Reducao no tempo total de acesso (ns)		344.155.199,79
Speed Up	1,5491	54,91%

Tabela 4 – Resultados das Simulações dos traces Applu, Galgel, Art duplicados:

Traces	applu.galgel.art.applu.galgel.art	applu.galgel.art.applu.galgel.art
Processador	Pentium M	Equivalente Reconfig.
Tamanho da Cache em palavras	8192	8192
Total de acessos	62.090.492	62.090.492
Taxa de Acerto (%)	82,89	86,04
Tempo médio de acesso (ns)	22,10	18,9586
Tempo de acesso total (ns)	1.372.339.452,63	1.177.148.801,63

Aumento na Taxa de Acerto		3,15
Reducao no tempo medio de acesso (ns)	3,14	
Reducao no tempo total de acesso (ns)	195.190.650,99	
Speed Up	1,1658	16,58%

Traces	applu.galgel.art.applu.galgel.art	applu.galgel.art.applu.galgel.art
Processador	Itanium 2	Equivalente Reconfig.
Tamanho da Cache em palavras	4096	4096
Total de acessos	62.090.492	62.090.492
Taxa de Acerto (%)	78,19	81,94
Tempo médio de acesso (ns)	26,81	23,0536
Tempo de acesso total (ns)	1.664.354.265,21	1.431.409.366,37

Aumento na Taxa de Acerto		3,75
Reducao no tempo medio de acesso (ns)	3,7517	
Reducao no tempo total de acesso (ns)	232.944.898,84	
Speed Up	1,1627	16,27%

Traces	applu.galgel.art.applu.galgel.art	applu.galgel.art.applu.galgel.art
Processador	UltraSparc III	Equivalente Reconfig.
Tamanho da Cache em palavras	16384	16384
Total de acessos	62.090.492	62.090.492
Taxa de Acerto (%)	81,49	89,97
Tempo médio de acesso (ns)	23,51	15,03
Tempo de acesso total (ns)	1.459.728.839,77	933.102.122,83

Aumento na Taxa de Acerto		8,48
Reducao no tempo medio de acesso (ns)	8,4816	
Reducao no tempo total de acesso (ns)	526.626.716,95	
Speed Up	1,5644	56,44%

Traces	applu.galgel.art.applu.galgel.art	applu.galgel.art.applu.galgel.art
Processador	Pentium 4 90 nm	Equivalente Reconfig.
Tamanho da Cache em palavras	4096	4096
Total de acessos	62.090.492	62.090.492
Taxa de Acerto (%)	79,31	81,94
Tempo médio de acesso (ns)	25,69	23,05
Tempo de acesso total (ns)	1.594.943.304,20	1.431.409.366,37

Aumento na Taxa de Acerto		2,63
Reducao no tempo medio de acesso (ns)	2,6338	
Reducao no tempo total de acesso (ns)	163.533.937,83	
Speed Up	1,1142	11,42%

Traces	applu.galgel.art.applu.galgel.art	applu.galgel.art.applu.galgel.art
Processador	AMD Hammer	Equivalente Reconfig.
Tamanho da Cache em palavras	16384	16384
Total de acessos	62.090.492	62.090.492
Taxa de Acerto (%)	82,93	89,97
Tempo médio de acesso (ns)	22,07	15,02818
Tempo de acesso total (ns)	1.370.054.025,80	933.107.090,06

Aumento na Taxa de Acerto		7,04
Reducao no tempo medio de acesso (ns)	7,03726	
Reducao no tempo total de acesso (ns)	436.946.935,73	
Speed Up	1,4683	46,83%