UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

EXPLORANDO A UNIÃO ENTRE MEMÓRIA DE CONTEXTOS E DE INSTRUÇÕES DE UMA ARQUITETURA RECONFIGURÁVEL VISANDO A REDUÇÃO DE ÁREA E CONSUMO DE POTÊNCIA EM SISTEMAS EMBARCADOS

TRABALHO DE GRADUAÇÃO

Thiago Baldicera Biazus

Santa Maria, RS, Brasil

2014

EXPLORANDO A UNIÃO ENTRE MEMÓRIA DE CONTEXTOS E DE INSTRUÇÕES DE UMA ARQUITETURA RECONFIGURÁVEL VISANDO A REDUÇÃO DE ÁREA E CONSUMO DE POTÊNCIA EM SISTEMAS EMBARCADOS

Thiago Baldicera Biazus

Trabalho de Graduação apresentado ao Curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Graduação em Engenharia de Computação

Orientador: Profª. Dr. Mateus Beck Rutzig

Santa Maria, RS, Brasil

2014

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Engenharia de Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

EXPLORANDO A UNIÃO ENTRE MEMÓRIA DE CONTEXTOS E DE INSTRUÇÕES DE UMA ARQUITETURA RECONFIGURÁVEL VISANDO A REDUÇÃO DE ÁREA E CONSUMO DE POTÊNCIA EM SISTEMAS EMBARCADOS

elaborado por
Thiago Baldicera Biazus

como requisito parcial para obtenção do grau de
Graduação em Engenharia de Computação

COMISSÃO EXAMINADORA:

Mateus Beck Rutzig, Dr.
(Presidente/Orientador)

Everton Alceu Carara, Dr. (UFSM)

Benhur de Oliveira Stein, Dr. (UFSM)

Santa Maria, 17 de Janeiro de 2014.

AGRADECIMENTOS

À minha família, que me ajudou nos momentos de dificuldade.

Ao professor Mateus Beck Rutzig, pela orientação, esforço e dedicação para tornar esse trabalho o que ele é hoje. Obrigado pelas sugestões, ideias, conversas e correções.

À Luana Binotto, minha namorada e melhor amiga, por ter me incentivado sempre e me ajudado nos piores momentos. Obrigado por estar sempre ao meu lado.

Aos meus amigos, pelas conversas e troca de ideias que me ajudaram nesta caminhada.

RESUMO

Trabalho de Graduação
Curso de Engenharia de Computação
Universidade Federal de Santa Maria

EXPLORANDO A UNIÃO ENTRE MEMÓRIA DE CONTEXTOS E DE INSTRUÇÕES DE UMA ARQUITETURA RECONFIGURÁVEL VISANDO A REDUÇÃO DE ÁREA E CONSUMO DE POTÊNCIA EM SISTEMAS EMBARCADOS

AUTOR: THIAGO BALDICERA BIAZUS
ORIENTADOR: MATEUS BECK RUTZIG
Local da Defesa e Data: Santa Maria, 17 de Janeiro de 2014.

No presente trabalho, foi desenvolvida uma nova estrutura de memória específica para arquiteturas reconfiguráveis, soluções emergentes para sistemas embarcados. A nova memória, chamada de Memória Unificada, busca juntar os contextos de reconfiguração e as instruções em apenas uma estrutura de armazenamento. Através deste funcionamento, é quebrado o paradigma de usar uma memória para cada função. Além disso, a Memória Unificada também busca aumentar a taxa de acerto de contextos de reconfiguração, aumentando sua utilidade dentro do cenário de arquiteturas reconfiguráveis. Por fim, é mostrado que a Memória Unificada consegue reduções significativas de área e potência, enquanto o desempenho é aumentado nas análises com paridade de área e quase que mantido no caso de redução de área.

Palavras-chave: Arquitetura. Reconfiguravel. Cache. Unificada.

ABSTRACT

Undergraduate Final Work
Degree in Computer Engineering
Federal University of Santa Maria

EXPLORING THE UNION BETWEEN CONTEXT AND INSTRUCTION MEMORY OF A RECONFIGURABLE ARCHITECTURE AIMING THE REDUCTION OF AREA AND POWER CONSUMPTION IN EMBEDDED SYSTEMS

AUTHOR: THIAGO BALDICERA BIAZUS
ADVISOR: MATEUS BECK RUTZIG
Defense Place and Date: Santa Maria, January 17, 2014.

In this work, a new structure of memory for reconfigurable architectures, emerging solutions for embedded systems, has been developed. The new memory, called Unified Memory, try to merge reconfiguration contexts and instructions in only one storage structure. Through this operation, the paradigm of using one memory for each function is broken. Moreover, the Unified Memory also seeks to increase the hit rate of reconfiguration contexts, increasing its usefulness within the reconfigurable architecture scenario. Finally, it is shown that the Unified Memory achieve significant reductions in area and power, while the performance is increased in the parity area analysis and almost maintained in the reduction of area analysis.

Keywords: Reconfigurable. Architecture. Unified. Cache.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os sistemas embarcados vêm ocupando um enorme espaço no mercado atual, e isto acontece pelo fato de cada vez mais fornecerem diferentes funcionalidades em um único dispositivo, como os aparelhos celulares. Esta tendência em agregar diversas aplicações em um único dispositivo embarcados torna o projeto dos mesmos cada vez mais desafiador.

O primeiro desafio é suportado pela Figura 1.1, que mostra que a bateria desde o ano de 1990 não tem evoluído na mesma velocidade que as outras periféricos presentes em sistemas embarcados. Este problema se torna demasiadamente preocupante, uma vez que reunir diferentes aplicações, que produzem comportamentos totalmente distintos, exige muito dos dispositivos e, consequentemente, da bateria.

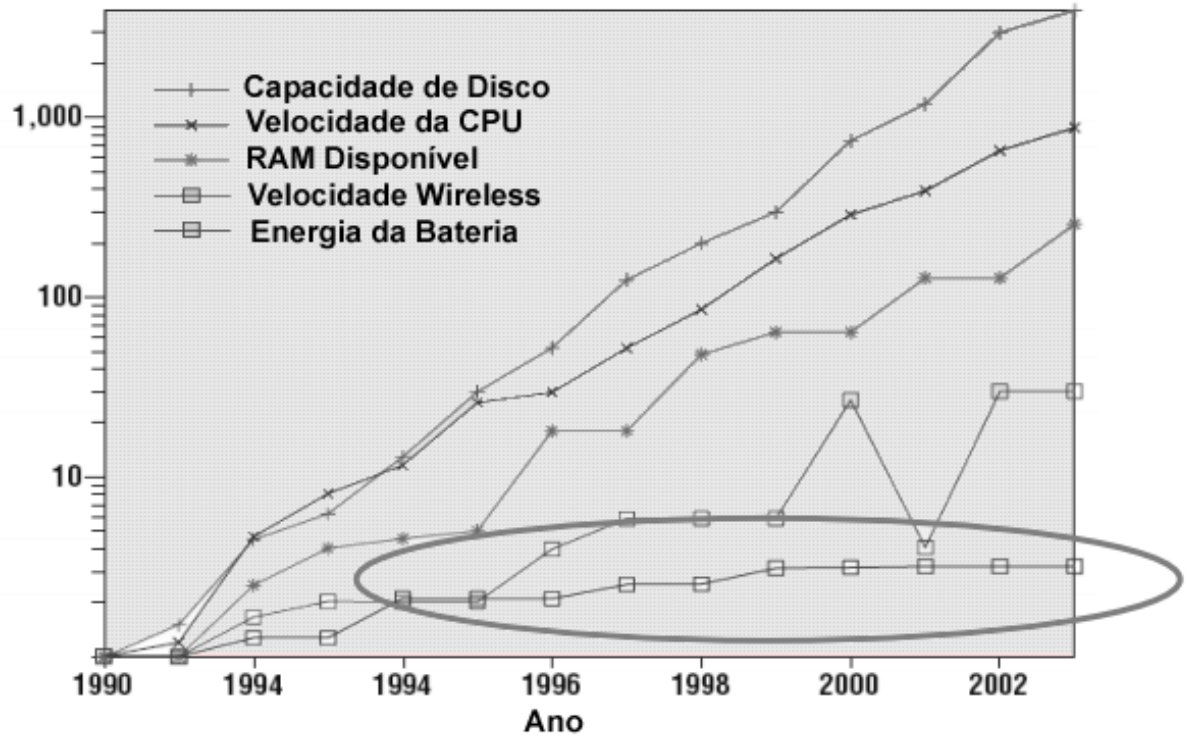

Figura 1.1 – Melhora dos recursos computacionais entre 1900 e 2002. [1]

Assim, a tendência de agregar mais aplicações em um único dispositivo torna o mesmo cada vez mais complexo. Para atender esse gradativo aumento de complexidade, os sistemas estão buscando um desempenho cada vez maior. Isto pode ser notado na Figura 1.2, que demonstra a constante busca pelo aumento da frequência de clock durantes os anos, de modo que se tente fornecer um maior desempenho aos sistemas.

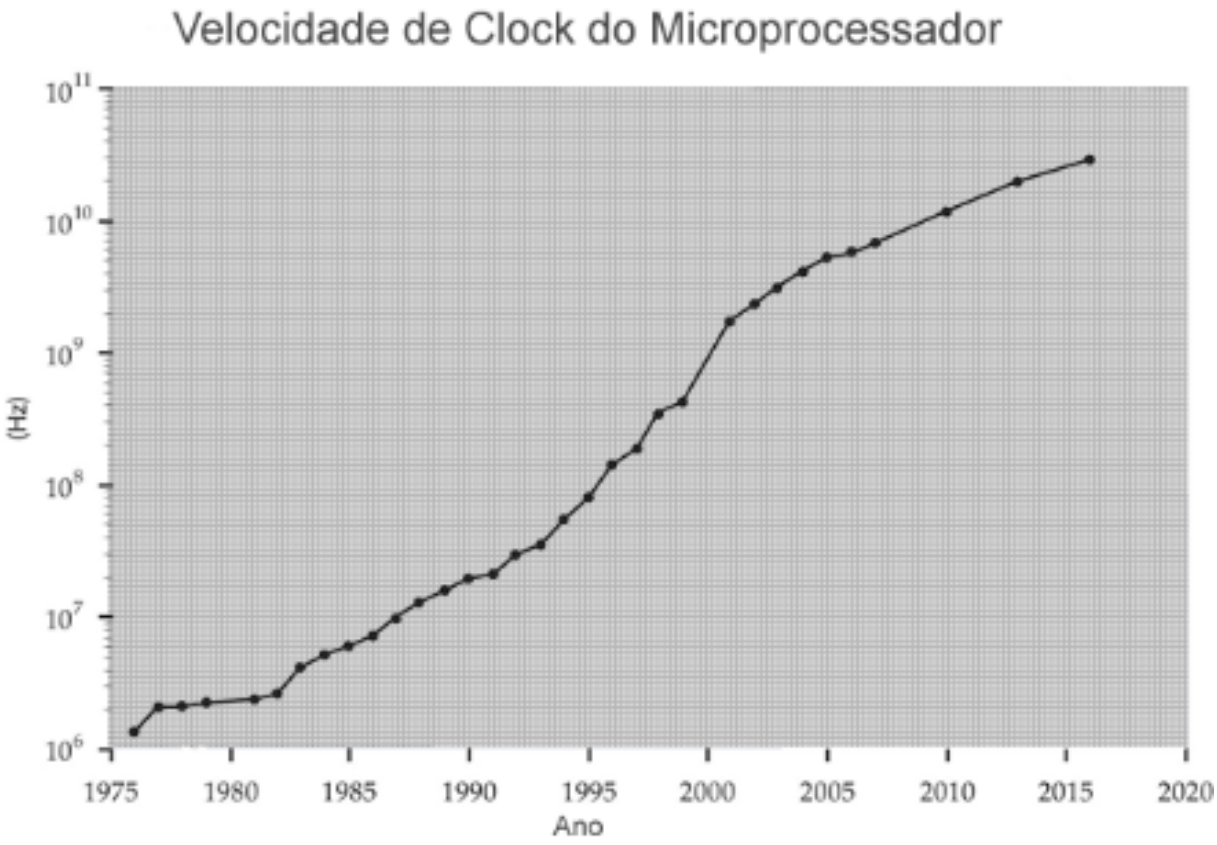

Figura 1.2 – Aumento da Velocidade de Clock dos Microprocessadores desde 1975. [2]

Assim sendo, ficam claras duas características necessárias em sistemas embarcados: o máximo de desempenho e o mínimo de energia possível. Além disso, outras restrições são muito importantes em sistemas embarcados: é desejável ocupar e consumir o mínimo de área e de potência. Todas estas restrições acabam indo uma de encontro à outra, resultando em uma crescente complexidade na construção de dispositivos embarcados. Entretanto, existe uma possível solução para tentar seguir todas estas restrições: o uso de arquiteturas reconfiguráveis.

As arquiteturas reconfiguráveis são estruturas que conseguem se adaptar a diferentes tipos de aplicação, de modo que consiga acelerá-las. São formadas por basicamente quatro blocos, todos essenciais para seu funcionamento: um processador de propósito geral, para executar as instruções nativas; um hardware tradutor, capaz de detectar partes do código que podem ser acelerados; uma RFU (Reconfigurable Function Unit), responsável por acelerar de fato as aplicações; e uma memória de contextos, responsável por salvar os contextos que serão executados na RFU.

O processador de propósito geral serve para executar as instruções nativas até que o

hardware tradutor consiga detectar quais instruções podem ser executas na RFU. Uma vez que o hardware tradutor consiga detectar e agrupar um conjunto de instruções, forma-se um contexto de reconfiguração que é armazenado na memória de contextos. Quando o sistema reconfigurável verifica que o início de um contexto de reconfiguração já detectado será executado no processador de propósito geral, a execução então é desviada para a RFU, acarretando uma aceleração das instruções que compõem o contexto.

Apesar de todos os blocos citados serem necessários para o funcionamento das arquiteturas reconfiguráveis, a memória de contexto é vista como um problema para o cenário de sistemas embarcados. Isto ocorre pelo fato desta estrutura ser redundante neste sistema devido a presença da memória de instruções do processador de propósito geral.

Além disto, estas duas estruturas de memória possuem um comportamento muito particular em arquiteturas reconfiguráveis. No início da execução da aplicação, enquanto os contextos de reconfiguração ainda estão sendo detectados e as instruções nativas estão sendo executadas no processador de propósito geral, a memória de instruções acaba sendo muito acessada, enquanto a memória de contextos fica ociosa. Em um segundo momento, com os contextos de reconfiguração já detectados, ocorre a reutilização dos mesmos, onde a memória de contextos passa, então, a ser acessada com maior frequência, deixando, assim, a memória de instruções ociosa.

Sendo assim, é necessário se perguntar se de fato duplicar uma estrutura já existente é vantajoso para o cenário de sistemas embarcados. E a conclusão é que não, visto que, apesar de garantir um bom desempenho do sistema, uma estrutura de memória duplicada dentro do sistema introduz um aumento de área, potência e energia, restrições que possuem importancias equivalentes dentro deste cenário.

Logo, o objetivo deste trabalho é construir uma nova estrutura de memória, que será capaz de comportar tanto instruções nativas como contextos de reconfiguração, retirando do sistema a duplicação de estruturas de memória, vista como desnecessária.

Esta nova estrutura de memória, chamada de Memória Unificada, deverá reduzir o custo em área, potência e energia acarretado pela duplicação desta estrutura. Entretanto, deverá manter o desempenho, visto que o mesmo é uma restrição também muito importante no cenário de sistemas embarcados.

No restante destes trabalho, são apresentados, em sequência: os conceitos de arquiteturas reconfiguráveis e memórias cache, os trabalhos relacionados já existentes na literatura,

a implementação da nova estrutura de memória, os resultados extraídos das simulações e as conclusões finais.

2 CONTEXTUALIZAÇÃO

Neste capítulo, serão apresentados os conceitos envolvendo as duas áreas estudadas e desenvolvidas neste trabalho: as arquiteturas reconfiguráveis e memórias cache.

2.1 Contextualização: Arquitetura Reconfigurável

Uma arquitetura reconfigurável é composta, basicamente, por quatro básicos lógicos: uma Reconfigurable Function Unit (RFU), uma memória de contextos, um hardware tradutor e um processador de propósito geral. Sua principal função é conseguir se adaptar e acelerar diferentes tipos de aplicações através da utilização destes quatro blocos lógicos. Para isto, cada bloco lógico tem uma função dentro do sistema. As funções de cada bloco lógico se encontram descritas na Tabela 2.1.

Bloco Lógico	**Função**
Processador de Propósito Geral	Executa o código nativo
Hardware Tradutor	Estrutura que detecta instruções que podem ser executadas na RFU, de modo que se melhore o desempenho da aplicação
RFU	Unidades funcionais capazes de se moldar de acordo com a necessidade da aplicação
Memória de Contexto	Memória responsável por armazenar os contextos de reconfiguração necessários para a RFU

Tabela 2.1 – Blocos Lógicos de uma Arquitetura Reconfigurável.

O funcionamento de todos os blocos lógicos em conjunto se encontra exemplificado na Figura 2.1. Através dela, podemos notar que o processador de propósito geral é responsável por executar as instruções nativas da aplicação. Enquanto isso, o hardware tradutor se comunica com o mesmo, detectando e traduzindo instruções que podem ser agrupadas em um contexto de reconfiguração. Uma vez que um contexto de reconfiguração é gerado, existe uma comunicação entre o hardware tradutor e a memória de contextos, de modo que a memória salve este contexto de reconfiguração para que possa ser, futuramente, reutilizado. Além disso, notamos que, uma vez que chegue o momento de reutilização de um contexto de reconfiguração, o mesmo é fornecido pela memória de contextos, a qual, através de suas informações, consegue configurar

corretamente a RFU para realizar o conjunto de instruções desejado.

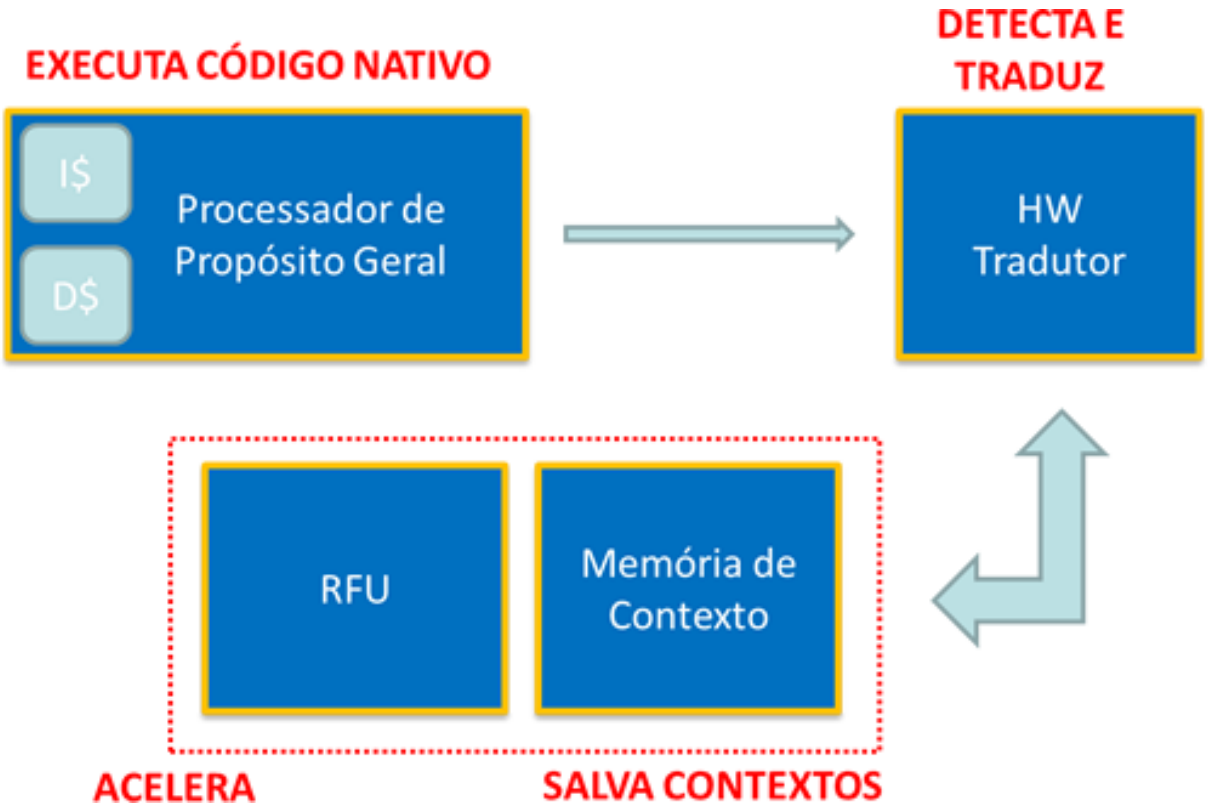

Figura 2.1 – Funcionamento de uma Arquitetura Reconfigurável.

Como seu principal objetivo é acelerar a execução de aplicações, busca-se sempre um bom compromisso entre o tempo de reconfiguração dos blocos lógicos e a execução das partes do software que lhe foram designadas. Estas partes do código são detectadas para serem executadas na arquitetura reconfigurável através da seguinte política: a soma do tempo de reconfiguração e execução deve ser menor do que o tempo que seria gasto para as instruções serem executadas apenas no processador de propósito geral. Além disso, outro fator importante no cenário de sistemas embarcados é a diminuição do consumo de energia que estas arquiteturas proporcionam. Isto se dá pela redução do tempo de execução fornecido pelas mesmas.

Uma RFU tem diversas características que a permitem ser classificada, sendo elas: acoplamento, granularidade, reconfigurabilidade e mecanismos de reconfiguração.

2.1.1 Acoplamento

O acoplamento é responsável por conectar a RFU ao processador principal, lidando com a comunicação e sincronização envolvidas neste processo. Por realizar tais tarefas, a escolha do

tipo de acoplamento acaba variando diretamente o desempenho da arquitetura reconfigurável.

Existem dois tipos de classificação de acoplamento em arquiteturas reconfiguráveis: a fortemente e a fracamente acoplada. Ambas estão descritas na Tabela 2.2, onde são mostradas suas características, desempenho e custo.

Tipo	Característica	Desempenho	Custo
Forte	A RFU é implementada como uma unidade funcional dentro do processador	Alto	Alto
Fraca	A RFU se comunica com o processador através de um barramento	Baixo	Baixo

Tabela 2.2 – Tipos de acoplamento de uma Arquitetura Reconfigurável.

2.1.2 Granularidade

A granularidade define qual será o nível de manipulação de dados em uma determinada arquitetura reconfigurável. Desta maneira, a granularidade estará diretamente ligada ao número de bits necessários para a reconfiguração do sistema e consequentemente do tamanho do contexto de reconfiguração. Ou seja, quanto maior o número de bits necessários para a reconfiguração, maior será a complexidade do controle necessário para gerenciar a arquitetura reconfigurável e maior terá de ser a memória que guardará os contextos traduzidos. Logo, é necessário escolher o tipo de granularidade de acordo com uma série de variáveis, tais como: conjunto de operações que devem ser executadas na RFU, área de armazenamento dos bits de reconfiguração, tempo de reconfiguração e desempenho.

Em arquiteturas reconfiguráveis, existem dois tipos de granularidade: de grão fino e de grão grosso. Cada tipo de granularidade, bem como suas características, estão descritos na Tabela 2.3.

Tipo	Característica
Fino	Arquiteturas reconfiguráveis onde a unidade de configuração é fina. Ex: LUTs (em FPGA)
Grosso	Arquiteturas reconfiguráveis onde a unidade de configuração é grande. Ex: ULA

Tabela 2.3 – Tipos de granularidade de uma Arquitetura Reconfigurável.

2.1.3 Reconfigurabilidade

A reconfigurabilidade define em qual momento a configuração da arquitetura irá acontecer. Quando se fala de reconfigurabilidade, existem dois tipos de classificação: configurável e reconfigurável, as quais estão descritas na Tabela 2.4.

Tipo	**Característica**
Configurável	A configuração dos blocos lógicos é realizada apenas no início da execução
Reconfigurável	A configuração dos blocos lógicos pode ser realizada durante a execução

Tabela 2.4 – Tipos de reconfigurabilidade de uma Arquitetura Reconfigurável.

2.1.4 Mecanismos de Reconfiguração

Os mecanismos de reconfiguração podem ocorrer em dois momentos diferentes: durante a compilação da aplicação ou durante a execução da mesma. Tais mecanismos estão diretamente ligados à compatibilidade de software da arquitetura reconfigurável, visto que, arquiteturas que procuram por partes do código que podem ser executadas na RFU durante o período de compilação só servirão para o hardware específico para que o compilador gera código. Ambos mecanismos de reconfiguração estão descritos na Tabela 2.5, juntos com suas vantagens e desvantagens.

Tipo	**Característica**
Durante a Compilação	Detecta durante a fase de compilação quais partes do código podem ser mais eficientes se rodadas na RFU
Durante a Execução	Detecta em tempo de execução quais partes do código podem ser mais eficientes se rodadas na RFU

Tabela 2.5 – Tipos de mecanismos de reconfiguração de uma Arquitetura Reconfigurável.

2.1.5 Memória de Contextos

As memórias de contextos armazenam as configurações previamente realizadas para serem executadas na RFU. Como são responsáveis por indicar quais as ligações corretas entre os diversos blocos lógicos da RFU, o número de bits que uma configuração está diretamente ligada ao tipo de granularidade adotada pela arquitetura reconfigurável, afetando assim o tamanho

da memória de contexto. Esta relação ocorre pelo fato de arquiteturas de granularidade fina possuírem um número maior de bits necessários para a reconfiguração visto que o bloco lógico de configuração é menor, por exemplo uma LUT de um FPGA. Ao contrário de arquiteturas de grão grosso, onde o bloco lógico é maior (por exemplo, uma ULA), fazendo com que o número de bits necessários para uma configuração seja menor que a estratégia anterior.

Por este fato, a implementação da memória de contextos pode ser feita de diferentes maneiras, sendo esta dependente do tipo de granularidade. Para arquiteturas reconfiguráveis de granularidade grossa, as quais possuem um número menor de bits por palavra de reconfiguração, pode-se usar memórias cache, as quais possuem alto desempenho mas um custo elevado. Por outro lado, devido ao alto número de bits por palavra de reconfiguração das arquiteturas reconfiguráveis de granularidade fina, é necessário utilizar memórias de maior capacidade e menor custo, tal como a memória DRAM (Dynamic Random Access Memory).

2.2 Contextualização: Memória Cache

A memória cache, implementada através da tecnologia SRAM, é uma estrutura muito utilizada por possuir uma importante característica: alto desempenho. Entretanto, tal característica traz consigo algumas desvantagens, entre as quais destaca-se o alto custo em área e potência, por este fato a impossibilidade de implementações destas estruturas com alta densidade de armazenamento.

Apesar da baixa capacidade de armazenamento que estas memórias oferecem, sua utilização é quase que obrigatória hoje em dia. Isto acontece devido à hierarquia de memória, que busca, através de múltiplos níveis com diferentes tamanhos e velocidades de acesso, dar a ilusão de que o computador possui uma memória ideal, ou seja, de alto desempenho e alta capacidade de armazenamento. O funcionamento da hierarquia de memória se encontra exemplificado na Figura 2.2.

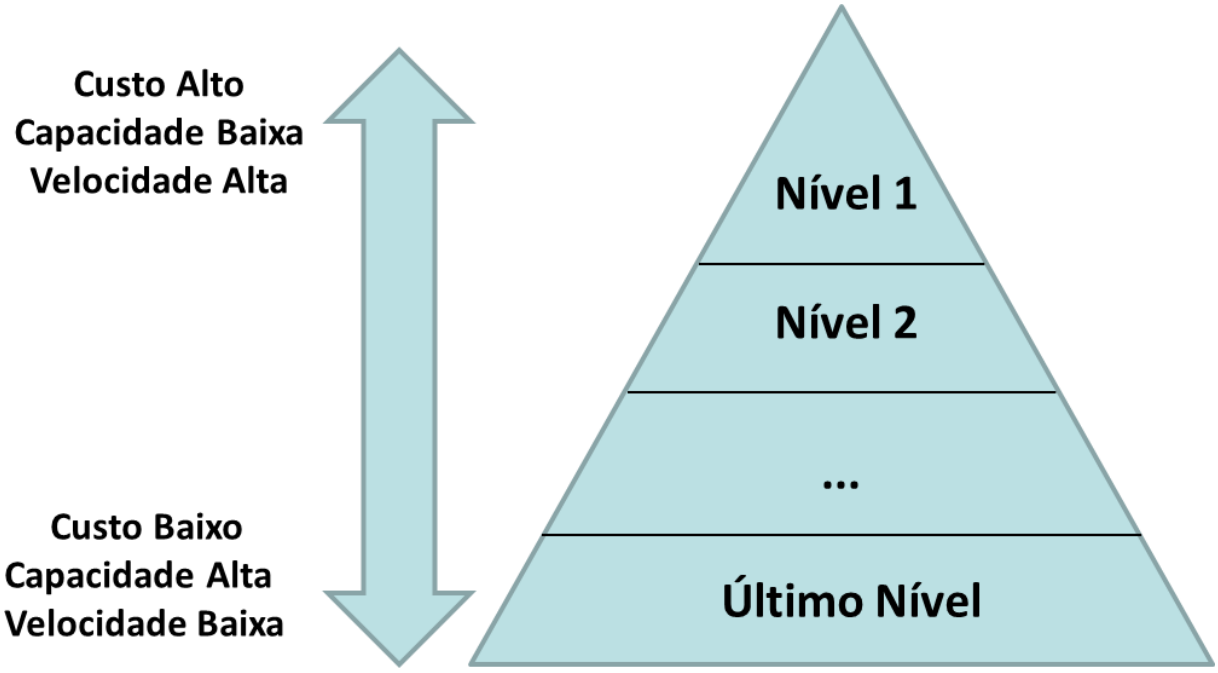

Figura 2.2 – Funcionamento da Hierarquia de Memória.
Fonte: Computer Organization and Design, 4ª Ed, D. A. Patterson e J. L. Hennessy. [3]

Estruturalmente, a memória cache é dividida em dois campos: bit de validade e rótulo. Suas posições são chamadas de índices, os quais, junto com os bits de validade e os rótulos, são responsáveis por checar se uma determinada informação se encontra disponível ou não na memória cache.

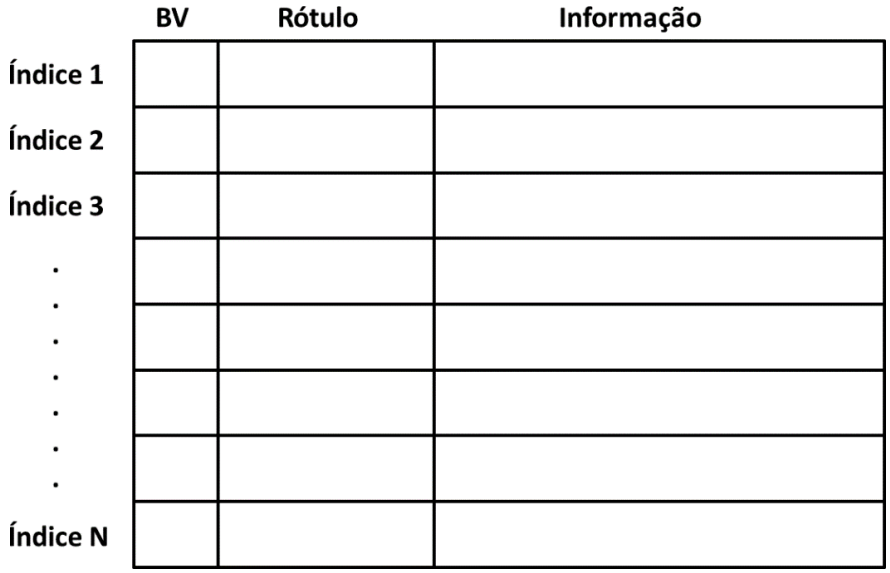

Figura 2.3 – Exemplo da estrutura de uma Memória Cache.

Quando uma informação é buscada na memória cache, o índice é encontrado, e o rótulo é comparado com o restante dos bits do endereço que está sendo buscado. Caso sejam iguais e o bit de validade seja válido, a informação está na memória cache. Este processo acaba gerando um acerto na memória. Por outro lado, se o rótulo não for igual, ou até mesmo for, mas o bit de validade não seja válido, uma falta é contabilizada na memória cache e a informação é buscada nos outros níveis de memória.

Assim sendo, o principal objetivo da memória cache é ter o menor número de faltas possível. Para isto, é utilizada uma técnica que permite explorar, de maneira eficiente, a hierarquia de memória: o princípio da localidade.

2.2.1 Localidade Espacial e Temporal

O princípio da localidade é a metodologia utilizada para tornar a hierarquia de memória eficiente. Isto é possível devido a exploração de dois conceitos de localidade: espacial e temporal.

Ambos os conceitos e suas respectivas características estão descritos na Tabela 2.6.

Tipo de Localidade	Característica
Localidade Temporal	Posições de memória, uma vez acessadas, tendem a ser acessadas novamente em um futuro próximo.
Localidade Espacial	Acessos futuros tendem a ser próximos de endereços de acessos passados.

Tabela 2.6 – Conceitos de Localidade Temporal e Espacial.

A exploração destes conceitos torna mais eficiente a memória cache, gerando, assim, menos faltas.

2.2.1.1 Exploração da Localidade Espacial pela Agregação de Múltiplas Palavras por Bloco

Para fazer uso da localidade espacial, é utilizado o conceito de múltiplas palavras por bloco. Esta técnica faz com que cada bloco tenha mais de uma palavra. Estas outras palavras do bloco serão formadas por posições próximas a palavra que foi requisitada, de forma que se tenha disponível na memória posições vizinhas à última a ser referenciada.

Esta técnica funciona da seguinte maneira: no momento em que uma requisição é feita na memória principal, todos os endereços próximos ao acessado são levados, no mesmo instante, para o mesmo bloco da memória cache. Desta maneira, uma vez que um endereço próximo ao último acessado seja requisitado no futuro, o mesmo já estará disponível na memória cache.

Apesar de um bloco com um maior número de palavras explorar mais a localidade espacial, existem problemas com aumento o número de palavras por bloco. Uma falta em um bloco com muitas palavras acaba tendo um custo elevado, uma vez que a latência para preencher todas as posições do bloco acaba reduzindo o desempenho do sistema, além de agregar potência.

Além disso, na Figura 2.4, que demonstra o impacto do número de palavras por bloco na taxa de acertos de uma memória, é possível observar que: quando se aumenta muito o número de palavras por bloco, a taxa de faltas tende a aumentar. Este comportamento acontece pelo fato de que, conforme o número de palavras por bloco é aumentado, o número total de blocos dentro da memória cache é reduzido, fornecendo assim mais faltas na memória cache.

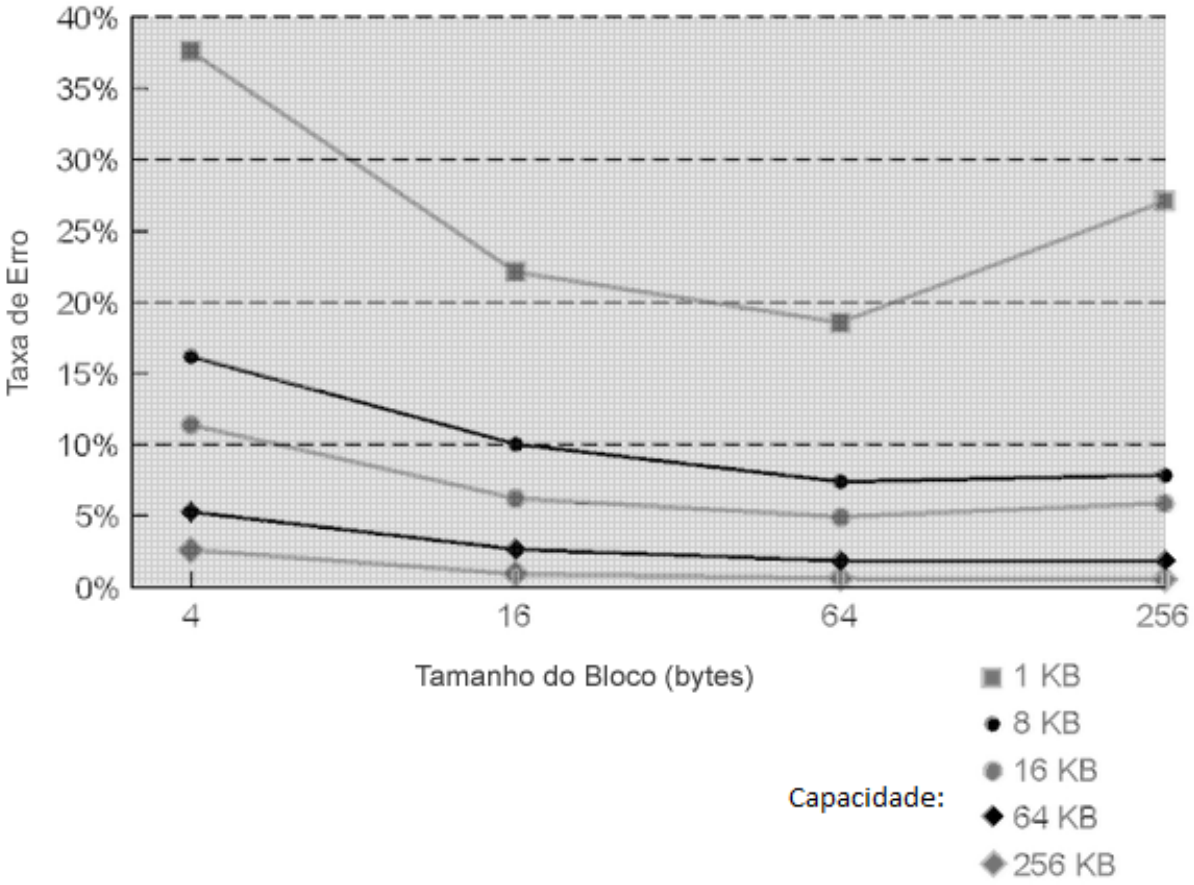

Figura 2.4 – Taxa de erro com número de palavras por bloco variando. [3]

2.2.1.2 Exploração da Localidade Temporal através da Associatividade

Por outro lado, como forma de se tirar proveito da localidade temporal, têm-se o conceito de associatividade. A associatividade busca aumentar o número de blocos disponíveis por índice, tentando, assim, conservar endereços recém atualizados na memória cache. Isto é feito através do agrupamento de blocos que geram o mesmo índice em conjuntos. Caso uma falta aconteça, apenas um bloco do conjunto é escolhido para ser substituído pela nova informação.

Quanto maior for a associatividade de uma memória cache, maior será o número de blocos disponíveis por conjunto. Desta forma, maior será o tempo que a informação ficará disponível na memória, garantindo assim que novas requisições em um curto espaço de tempo gerem acertos.

Apesar disso, a associatividade apresenta problemas quanto maior for o número de blocos por conjunto. Isso se deve ao fato de, quanto maior for o número de blocos, maior deverá ser o número de comparadores para extrair a informação correta da memória. Consequente-

mente, quanto maior for o número de comparadores, maior será a área ocupada pela estrutura de memória e a potência consumida durante a busca por uma informação.

O impacto que a associatividade gera na taxa de faltas de uma memória cache pode ser observado na Figura 2.5.

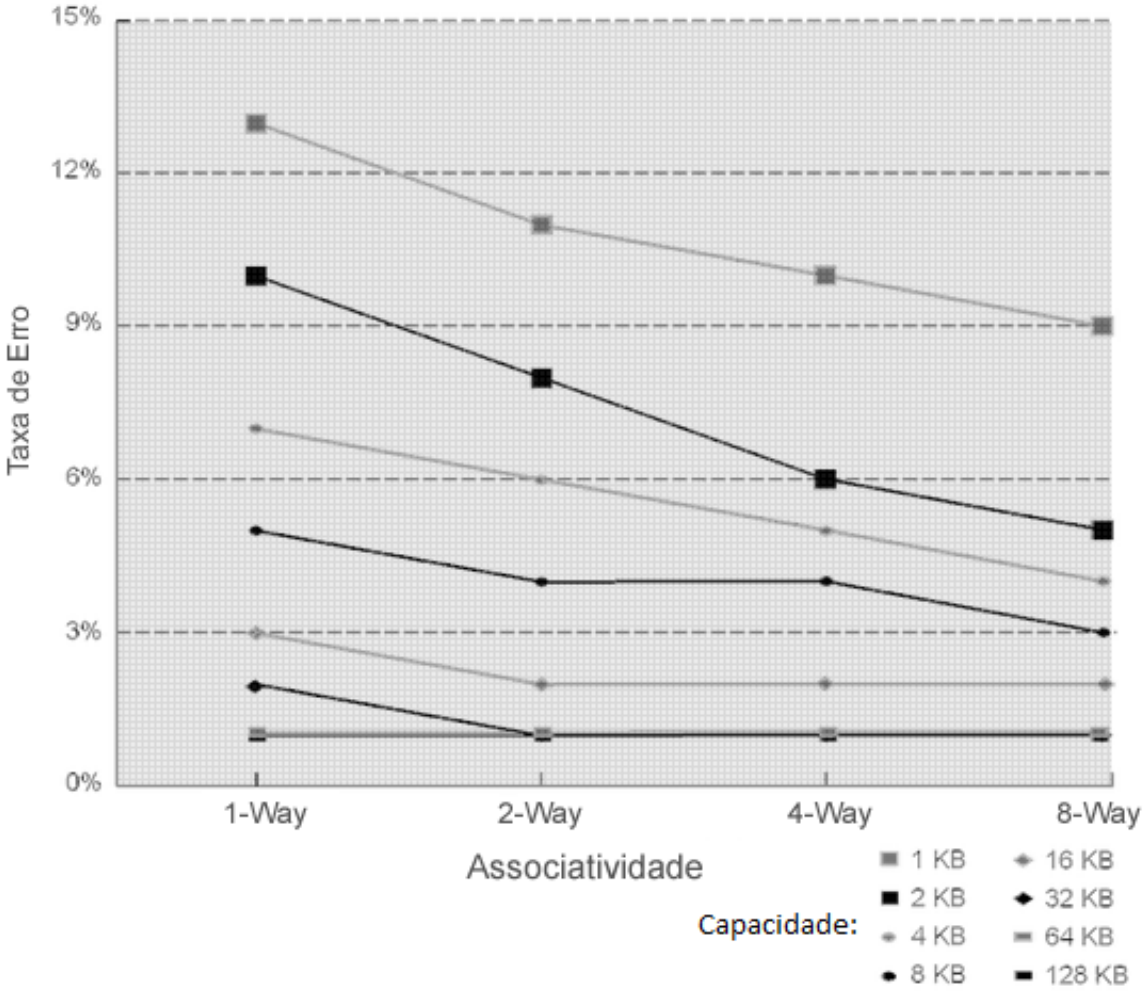

Figura 2.5 – Taxa de erro com diferentes associatividades. [3]

2.2.2 Cache Mapeada Diretamente

Uma memória cache mapeada diretamente corresponde à estrutura mais simples possível. Tal memória possui apenas um bloco por índice. A estrutura de uma memória cache mapeada diretamente é exemplificada na Figura 2.6.

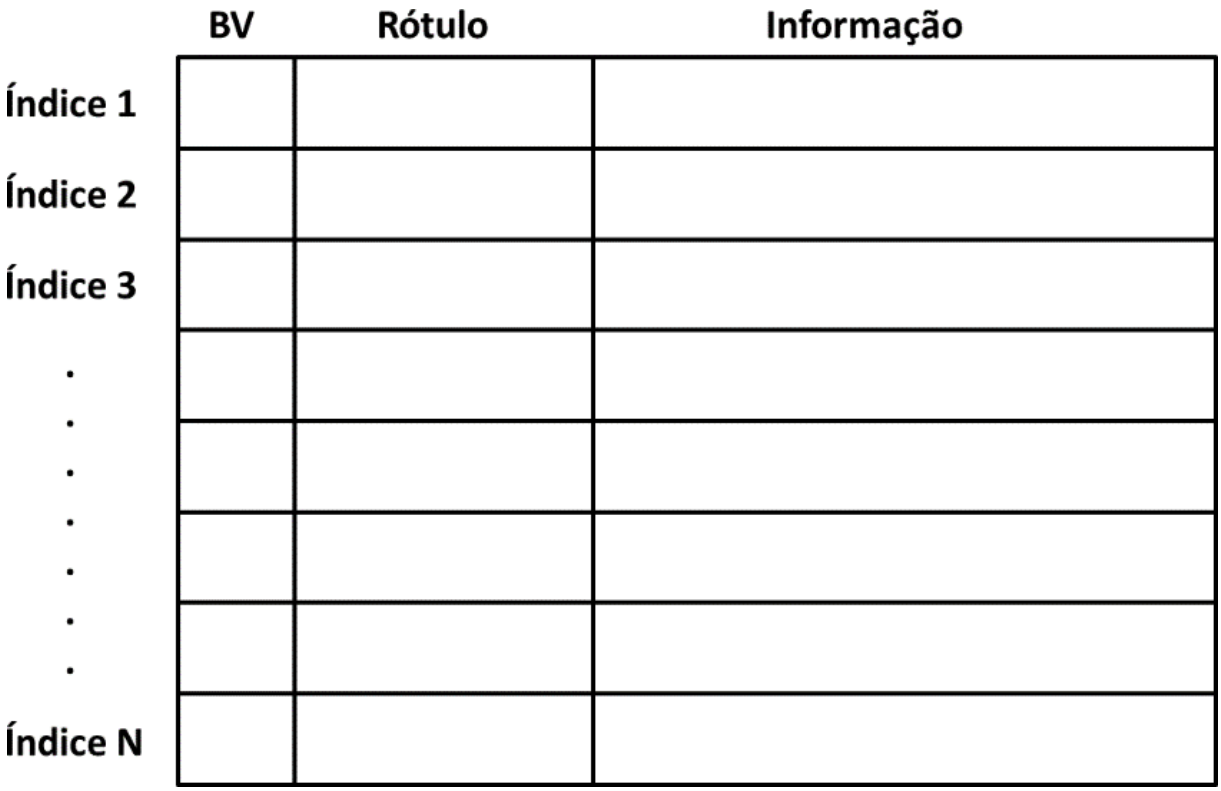

Figura 2.6 – Estrutura de uma memória cache mapeada diretamente.

Assim sendo, um endereço terá apenas uma posição possível para ser alocado. Em outras palavras, diversos endereços terão que compartilhar o mesmo índice e bloco, gerando, assim, um elevado número de conflitos de endereços e consequentemente faltas na memória cache, fato indesejável.

O processo de busca de uma informação na memória cache é feito através do cálculo do índice e comparação do rótulo e da validação do bit de validade. Tal processo mostra que, apesar de gerar um alto número de faltas, a memória cache mapeada diretamente tem algumas vantagens, tal como o menor consumo de potência que as outras organizações devido a inserção de somente um comparador de rótulo na implementação.

2.2.3 Cache Totalmente Associativa

Em uma memória cache totalmente associativa, um endereço não corresponde a um índice específico, e, sendo assim, pode ser armazenado em qualquer posição que esteja disponível. Como não existe mais o conceito de índice, todo o endereço é armazenado como rótulo, para que seja possível compará-lo posteriormente.

A Figura 2.7 exemplifica o processo de busca de uma informação em uma memória

totalmente associativa.

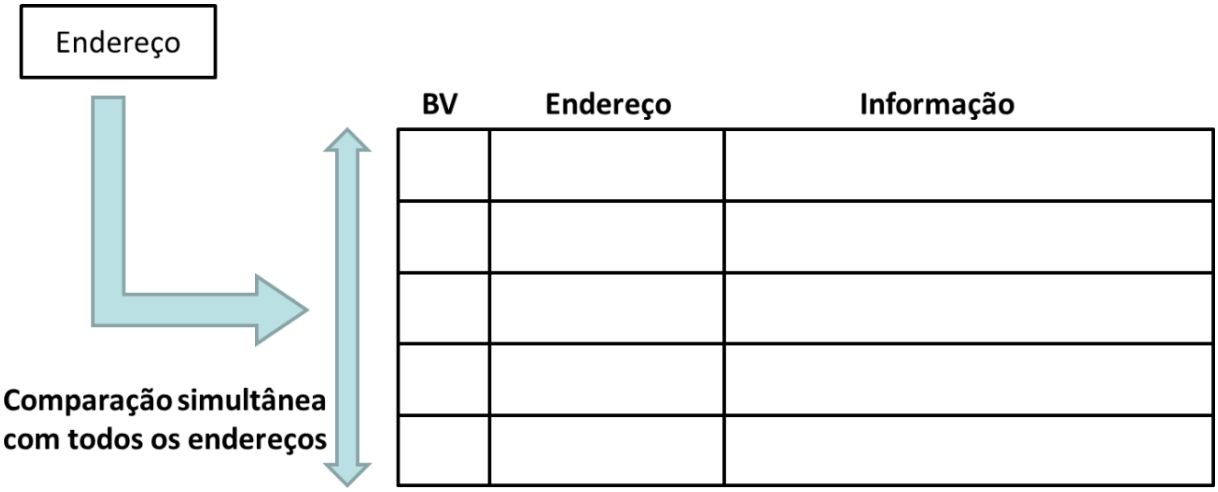

Figura 2.7 – Exemplo de busca em uma memória cache totalmente associativa.

Para buscar a linha em que o endereço deve ser armazenado, são utilizados algoritmos que buscam explorar a localidade temporal das infomações. Algoritmos como FIFO (First in – First out), LRU (Least Recently Used) e LFU (Least Frequently Used) são exemplos de mecanismos que tentam explorar diferentes conceitos, de modo que se consiga reduzir o número de faltas.

Apesar de explorar ao máximo a localidade temporal, este tipo de memória acaba tendo duas desvantagem significativas: a comparação simultânea de todos os endereços armazenados, quando ocorre uma requisição de uma informação e a busca por uma posição para ser substituída.

Caso uma informação seja buscada na memória cache, é necessário comparar de forma simultânea todas as posições para verificar se a mesma se encontra disponível, esta tarefa requer N comparadores, sendo que N é o valor do número de blocos que a memória cache contém. Este fato torna a implementação deste tipo de organização infactível para memórias que contém mais de 2048 blocos devido ao consumo de potência e área dos comparadores.

Além disto, para realizar a busca por uma posição a ser substituída, é necessário um hardware que implemente o algoritmo de substituição. Tal hardware acabará elevando a área necessária para o sistema, além de também gerar um gasto adicional de energia.

2.2.4 Cache Associativa por Conjunto

As caches associativas por conjunto buscam unir a exploração do conceito de localidade temporal com uma busca por informação mais rápida e eficiente. Isto é feito pois o número de conjuntos desejado é agrupado em um mesmo índice, como mostra a Figura 2.8.

Figura 2.8 – Estrutura de uma possível configuração de uma memória cache associativa por conjunto.

Esta organização surge como forma de unir as duas organizações de memórias mostradas anteriormente: a totalmente associativa e a mapeada diretamente. Isto acontece pois, nessa organização de memória, o conceito de índice, utilizado em memórias mapeadas diretamente, é novamente explorado, ao contrário do que acontecia em uma memória cache totalmente associativa. E, além disso, os blocos dentro de um índice acabam por formar um conjunto totalmente associativo, explorando, parcialmente, o conceito que é explorado em memórias totalmente associativas. Neste tipo de memória, quando uma falta acontece, é necessário escolher qual bloco do conjunto será substituído. Para tal tarefa, utilizam-se diferentes algoritmos, de acordo com o que é desejado explorar.

2.2.5 Cache Associativa com Múltiplas Palavras por Bloco

Para se utilizar o conceito de localidade espacial e temporal ao mesmo tempo, são utilizadas memórias cache associativas com múltiplas palavras por bloco. Desta maneira, cada bloco do conjunto terá tantas palavras quanto for determinado, conseguindo, assim, utilizar o

conceito de localidade espacial. A estrutura de uma memória cache associativa com múltiplas palavras por bloco se encontra exemplificada na Figura 2.9.

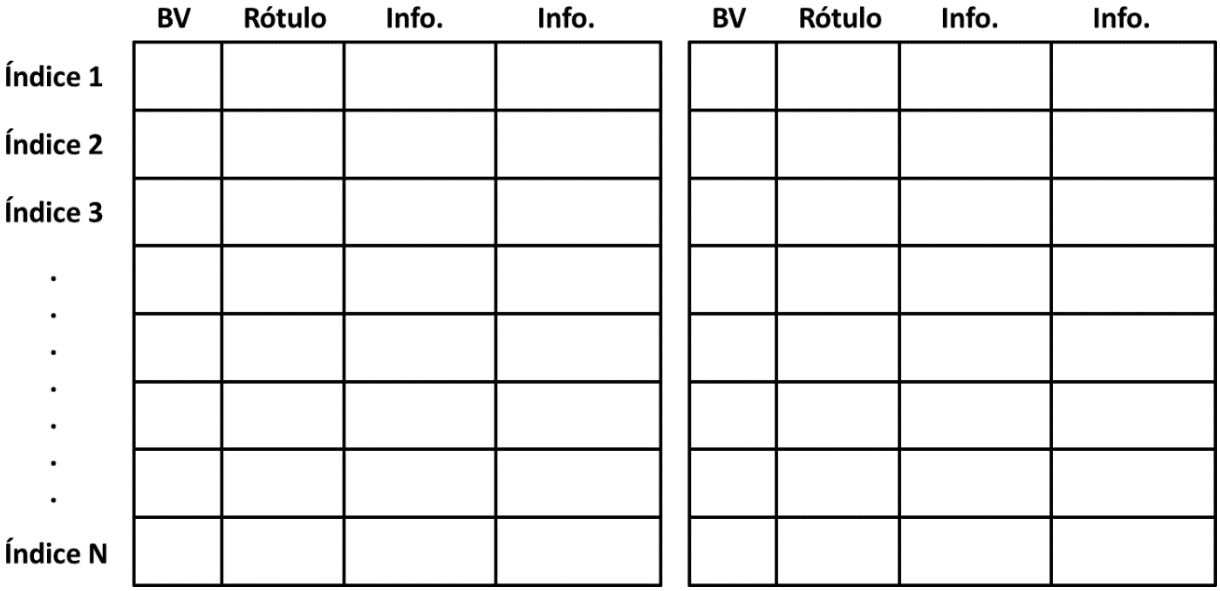

Figura 2.9 – Estrutura de uma possível configuração de uma memória cache associativa com múltiplas palavras por bloco.

Este tipo de memória utiliza os mesmos algoritmos já citados para substituição de blocos quando uma falta acontece. Porém, o novo bloco terá disponível, além do último endereço a ser referenciado, os endereços próximos ao mesmo, tentando, assim, gerar um número ainda menor de faltas do que o de uma cache associativa por conjunto sem a exploração da localidade espacial.

Logo, diante deste contexto, é notável que cada caso de implementação de memória cache exige um estudo para que se consiga chegar a uma configuração ideal para o que é desejado. Memórias cache totalmente associativas, apesar de oferecerem o melhor desempenho, não são indicadas devido ao seu alto custo em hardware. Por outro lado, memórias mapeadas diretamente também não são indicadas, uma vez que seu desempenho é muito baixo.

Devido a essas características, memórias mapeadas por conjunto surgem como alternativa para sistemas embarcados, onde o principal objetivo é conseguir um desempenho satisfatório tendo um baixo custo em hardware.

Além disso, é necessário definir qual a associatividade e o número de palavras por bloco que deverá ser utilizado. Como explicado anteriormente, uma alta associatividade, apesar de aumentar o desempenho do sistema, exige o uso de um alto número de comparadores, fato que nem sempre é aceitável no sistema. Quanto ao número de palavras por bloco, é necessário achar

um ponto ideal onde se consiga o máximo possível de desempenho para o sistema. Através da Figura 2.4, é possível notar que este número de palavras por bloco ideal não deverá ser um valor muito baixo, para garantir que exista uma melhora de desempenho. Entretanto, este número também não poderá ser muito alto, para garantir que a melhora de desempenho não seja reduzida.

3 TRABALHOS RELACIONADOS

As arquiteturas reconfiguráveis vem sendo utilizadas atualmente como forma de aumentar o desempenho e reduzir o consumo de energia de sistemas computacionais. Exemplos como [4, 5, 6] mostram arquiteturas reconfiguráveis com diferentes características que buscam extrair o maior desempenho possível economizando o máximo de energia.

Em [4], tem-se uma arquitetura reconfigurável fortemente acoplada, a qual se encontra limitada a execução de lógica combinacional. Na verdade, a RFU desta arquitetura funciona como uma outra unidade funcional (FU) qualquer do sistema, compartilhando todos os seus recursos com as demais. Já em [5], tem-se um sistema compatível com o processados MIPS e fracamente acoplado, onde a comunicação funciona através de instruções específicas.

No caso de [6], a arquitetura reconfigurável é proposta com um acoplamento forte, granularidade grossa e reconfiguração da RFU em tempo de execução. Esta arquitetura permite a detecção, em tempo de execução, de partes da aplicação com potencial de otimização, acarretando um melhor desempenho geral do sistema pelo fato de serem executadas na RFU. Tal mecanismo, por trabalhar em tempo de execução, garante a compatibilidade de software.

Por outro lado, arquiteturas reconfiguráveis como as propostas em [7, 8] utilizam-se de mudanças feitas no código durante o tempo de compilação para explorar a conceito de reconfiguração. Assim sendo, tais arquiteturas acabam por não garantir compatibilidade de software, sendo obrigatoriamente necessário recompilar a aplicação sempre que se deseja explorar o conceito de reconfiguração.

Um dos principais objetivos das arquiteturas reconfiguráveis é a otimização de aplicações pela inclusão de uma RFU, e, além disso, muitas levam em conta o impacto em área e potência que esta estrutura acarreta ao sistema. Entretanto, nenhuma das arquiteturas reconfiguráveis citadas acima busca levar em consideração o impacto que a memória de contexto traz ao sistema. A única pesquisa encontrada na literatura, feita por [14], busca estudar o impacto que existe em relação ao modo como a memória de contexto é acessada. Neste estudo, é possível observar que as memórias de contexto podem variar seu tamanho de acordo com o número de blocos lógicos de reconfiguração. Quanto maior o número de blocos lógicos, maior o número de informações que devem ser armazenadas na memória. Isto pode ser visto de acordo com a Tabela 3.1, que mostram o impacto do número de bloco lógicos de reconfiguração no tamanho da palavra de reconfiguração nas arquiteturas reconfiguráveis DIM, GARP e PipeRench:

	Conf. 1 - DIM	**Conf. 2 - DIM**	**Conf. 3 - DIM**	**Conf. 4 - DIM**	**GARP**	**PipeRench**
Níveis Totais	8	16	32	64	N/A	N/A
Colunas Totais	24	48	96	192	N/A	N/A
ULA / nível	8	8	12	12	N/A	N/A
Multip. / nível	1	2	2	2	N/A	N/A
LD ou ST / nível	2	6	6	6	N/A	N/A
Bytes de controle	124	142	165	165	N/A	N/A
Bytes necessários para uma configura-ção	820	1.838	4.261	8.357	6.144	21.504

Tabela 3.1 – Impacto do número de blocos lógicos no número de bits necessários para uma palavra de reconfiguração.

Por outo lado, quanto maior o número de bits necessário para salvar um contexto de reconfiguração, menor será o desempenho e maior será energia consumida pela memória de contextos.

O estudo mostrado em [13] revela que em média 43% do consumo total de energia da arquitetura reconfigurável MorphoSys [15] está na memória de contexto. Assim sendo, é necessário entender o impacto que a memória de contexto causa às arquiteturas reconfiguráveis e tentar otimizá-la ao máximo para garantir que o desempenho e redução de energia gerados com o uso do conceito de reconfiguração não seja perdido em parte pelo sistema de memória de contexto.

Estudos sobre otimização da hierarquia de memória são aplicados a sistemas multiprocessados, tais como [9, 10, 11, 12]. A maioria destes trabalhos busca, em tempo de execução, fornecer a melhor distribuição de espaço da memória levando em conta a demanda de cada processador. Entretanto, nenhum destes estudos pode ser aplicado diretamente à arquiteturas reconfiguráveis, uma vez que elas utilizam uma memória de contexto de reconfiguração, que não necessariamente irá atuar de uma forma idêntica às memórias de instrução e de dados.

Assim, este trabalho busca inserir os trabalhos de otimização de memórias em processa-

dores de propósito geral no cenário de arquiteturas reconfiguráveis. Assim, será desenvolvida uma técnica de memória cache reconfigurável para arquiteturas de granularidade grossa que será capaz de unificar em uma mesma estrutura a memória de instrução e a memória de contexto. Esta reconfiguração da memória cache será realizada em tempo de execução e levará em conta o comportamento do sistema, procurando sempre garantir o melhor desempenho possível. Além disso, uma vez que tais estruturas sejam unificadas, será possível reduzir consideravelmente a área ocupada e a energia consumida pelo sistema, características importantes para arquiteturas reconfiguráveis.

4 PROPOSTA DESTE TRABALHO

Neste capítulo, serão mostradas as tarefas realizadas para a implementação da técnica de Memória Unificada, bem como os passos tomados para validar o seu correto funcionamento. No próximo capítulo, a técnica proposta será aplicada na arquitetura reconfigurável DIM como um estudo de caso.

A Memória Unificada irá possuir, além dos campos tradicionais de uma memória cache, campos que possibilitem diferenciar as instruções nativas de contextos de reconfiguração. Tais campos estão expostos na Tabela 4.1.

Componente	**Função**
Campo Tipo de Informação	Responsável por diferenciar Instrução Nativa de Contexto de Reconfiguração
Algoritmo Alocador	Responsável por identificar se é necessário abrir mais slots para reconfiguração ou instrução nativa
Algoritmos de Substituição	Além de se comportar de acordo com o algoritmo escolhido (LRU ou Random), deverá utilizar o algoritmo somente no tipo de informação desejado

Tabela 4.1 – Componentes da Memória Unificada.

O seu funcionamento também será um tanto quanto diferente do que normalmente é observado em memórias cache tradicionais, como mostra o diagrama de blocos da Figura 4.1.

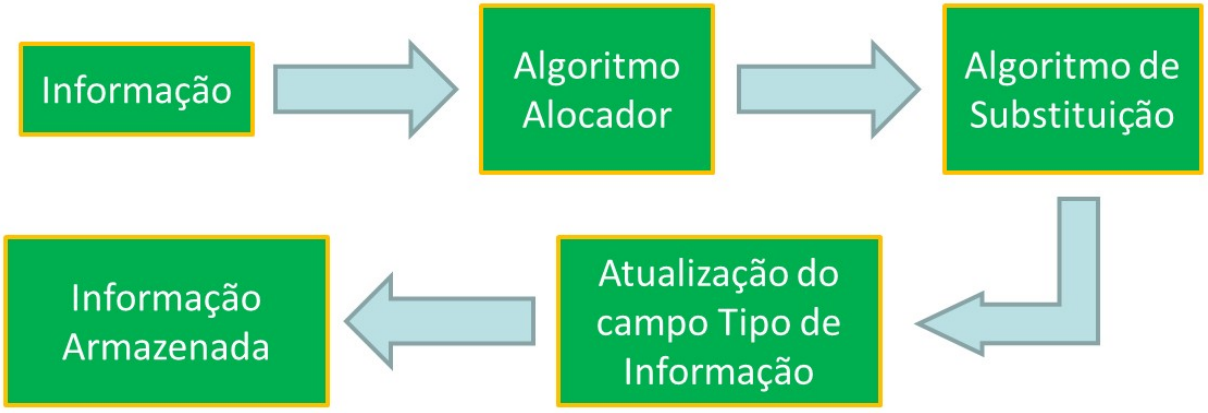

Figura 4.1 – Funcionamento da Memória Unificada.

Através da Figura 4.1, é possível observar que, uma vez que uma informação chega para ser armazenada na Memória Unificada, o primeiro passo é utilizar o hardware alocador

para decidir se é necessário alocar mais um bloco, dentro do conjunto referente ao endereço da informação que está sendo armazenada, para o tipo de informação que está sendo inserida ou apenas realizar a troca por um já existente.

Após esta etapa, o algoritmo de substituição escolhe o slot que deverá dar lugar ao novo através da técnica desejada (LRU ou Random). Após isto, o campo tipo de informação é atualizado, caso seja necessário, e a informação é armazenada na Memória Unificada.

4.1 A Estrutura da Memória Unificada

Antes de realizar a implementação de fato da Memória Unificada, é necessário definir qual será sua estrutura. A nova estrutura de memória deverá ter, além dos campos normalmente existentes em memórias cache tradicionais, um novo campo que deverá ser capaz de diferenciar o tipo de informação que está sendo armazenada, chamado de Tipo de Instrução. Este campo é capaz de identificar se a informação que está sendo armazenada é um contexto de reconfiguração ou uma instrução nativa.

Assim sendo, um bloco da Memória Unificada irá conter seu respectivo rótulo, bit de validade, informação e tipo de informação. Além disso, irá corresponder a um índice específico, como pode ser visto na Figura 4.2.

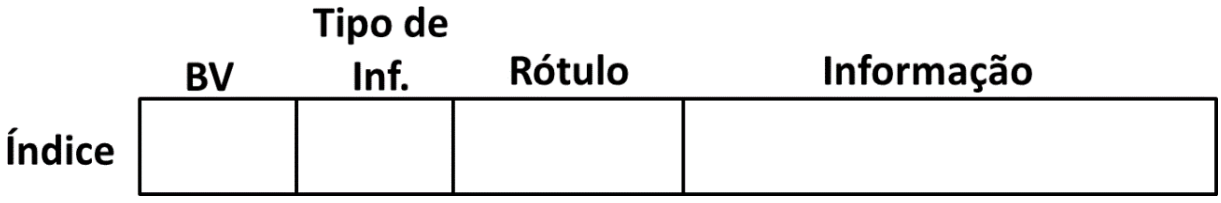

Figura 4.2 – A estrutura da Memória Unificada.

A existência do índice se deve ao fato de que não será implementada a técnica de memória cache totalmente associativa, devido ao alto consumo de potência exigido por essa técnica na hora de buscar por uma informação, fato indesejado para o cenário de sistemas embarcados. Assim sendo, qualquer outra possível estrutura de memória cache utiliza o conceito de índice, e, assim sendo, o mesmo é obrigatoriamente utilizado na estrutura da Memória Unificada.

Além disso, a Memória Unificada também deverá ter um número de palavras por bloco que consiga comportar um contexto de reconfiguração da arquitetura reconfigurável. Ou seja, o tamanho do contexto de reconfiguração regrará o tamanho do bloco da memória cache. Como

o número de palavras por bloco poderá ser variado de acordo com o que é desejado durante o projeto da memória, será apenas necessário definir qual o tamanho desejado para o contexto de reconfiguração, sendo que palavras maiores levarão a uma maior aceleração da aplicação, mas, ao mesmo tempo, acabarão reduzindo o número de blocos dentro da Memória Unificada. Assim sendo, durante o capítulo de resultados, serão apresentadas as escolhas feitas para o número de palavras por bloco e o motivo das mesmas.

4.1.1 Hardware Alocador

A técnica de Memória Unificada necessita um hardware inteligente, que seja capaz de gerenciar os blocos disponíveis na nova estrutura de memória desenvolvida. Este hardware será o principal responsável por garantir que o desempenho não seja comprometido de maneira muito significativa, uma vez que o tamanho da Memória Unificada disponível será reduzido se comparado ao tamanho que as Memórias Separadas disponibilizavam.

O hardware alocador deverá seguir um algoritmo, o qual deverá ser capaz de informar se devem ser abertos mais blocos para instruções nativas ou contextos de reconfiguração. Ele deverá ser capaz de atuar de maneira individual nos índices, não tendo que alterar toda a estrutura da Memória Unificada, mas sim os índices que estão gerando o maior número de faltas em um determinado tipo de informação. Esta atuação pode ser exemplificada na Figura 4.3.

Figura 4.3 – Exemplo de funcionamento do Hardware Alocador.

Além disso, este algoritmo deve buscar obter o máximo de desempenho sendo o menos complexo possível. Isto acontece pelo fato de a complexidade introduzir ao sistema características cujo foco do trabalho é eliminá-las, sendo elas: área, potência e energia.

4.1.2 Algoritmo de Alocação: O Algoritmo de Limiar

O algoritmo de limiar escolhido para o hardware alocador tem um funcionamento bem simples: é um contador, que será incrementado cada vez que uma falta ocorrer quando uma instrução nativa for buscada e decrementado cada vez que uma busca por um contexto de reconfiguração acabe gerando uma falta.

Esse contador será armazenado para ser comparado com um limiar. Caso o contador se encontre acima do limiar, o hardware alocador irá permitir que mais blocos sejam abertos para instruções nativas caso seja necessário. Por outro lado, caso esteja abaixo do limiar, ele irá permitir, então, que mais blocos sejam disponibilizados para contextos de reconfiguração.

O valor do limiar que será utilizado deve priorizar ao máximo os contextos de reconfiguração, uma vez que eles são o maior propósito das arquiteturas reconfiguráveis e um acerto neste tipo de informação fornecerá aceleração à aplicação. O limiar deverá ter um limite máximo e mínimo, de modo que, caso aconteça muitas faltas em sequência de um determinado tipo de informação, o outro tipo não seja tão desprezado.

O fato de o hardware alocador seguir um algoritmo tão simples é extremamente desejável no cenário de sistemas embarcados, uma vez que conseguirá economizar o máximo de área, potência e energia, mas, ainda assim, tentará garantir o mesmo desempenho de memórias cache implementadas de forma separadas.

4.1.3 Algoritmos de Substituição

Uma vez definidas as características básicas da técnica de Memória Unificada e o hardware alocador, é necessário definir as funções que serão responsáveis por definir qual bloco deverá ser substituído uma vez que seja necessário.

Estas funções deverão ser capazes de, além de definir qual bloco do mesmo tipo de informação dará lugar ao que está sendo referenciado no momento, qual bloco do outro tipo de informação pode ser substituído caso seja necessário. É claro que, caso existam blocos livres dentro dos índices, os mesmo irão ser selecionados para serem atualizados, não sendo necessário substituir nenhum bloco.

No caso de uma busca de bloco de um outro tipo de informação, o algoritmo irá seguir as ordens que serão definidos pelo hardware alocador. Assim sendo, foram escolhidos dois algoritmos de substituição para serem implementados: o LRU (Least Recently Used) e o Random (Randômico).

4.1.3.1 Algoritmo LRU

O algoritmo de substituição LRU, atualmente, é visto como o que possui o melhor desempenho em memórias cache. Por isto, foi escolhido para ser implementado na Memória Unificada.

Este algoritmo funciona da seguinte maneira: dentro do índice correspondente ao endereço que está sendo adicionado, é escolhido o bloco que foi referenciado há mais tempo para ser substituído.

Seu funcionamento é complexo: é necessário um vetor de contadores onde seja possível atualizar os endereços conforme eles são referenciados, de modo que se tenha sempre um maior contador no endereço que foi referenciado há mais tempo.

Entretanto, como esta técnica de Memória Unificada irá tratar de dois tipos de informação na mesma estrutura de memória, é necessário que o algoritmo LRU seja capaz de atualizar e verificar apenas endereços do mesmo tipo de instrução. Assim sendo, seu funcionamento, por mais que seja parecido com o algoritmo LRU normalmente utilizado em memórias cache, não é totalmente igual. E, por isso, necessita que seus dados de área, potência e energia sejam extraídos, para que seja possível compará-lo ao algoritmo LRU normalmente utilizado em memórias cache.

4.1.3.2 Algoritmo Random

Apesar do algoritmo LRU possuir o melhor desempenho, a alta complexidade envolvida no seu funcionamento introduz um gasto adicional de área, potência e energia, sendo tais características prejudiciais aos sistemas embarcados.

Logo, o algoritmo randômico também foi escolhido para ser implementado, uma vez que possui um desempenho próximo ao do algoritmo LRU, porém não é tão complexo quanto o mesmo. O seu funcionamento é simples: ele seleciona, de maneira randômica, qual será o bloco que deverá ser substituído para que o novo endereço seja inserido na Memória Unificada.

Entretanto, o mesmo problema existente no algoritmo LRU acontece neste caso: o con-

flito causado pelo campo tipo de informação. Por conta deste campo, o algoritmo randômico precisa, além de gerar o número aleatório, verificar se a posição escolhida possuí o campo tipo de instrução igual ao que está sendo requisitado. Como no algoritmo LRU, o algoritmo Random utilizado na técnica de Memória Unificada terá seus dados de área, potência e energia extraídos, de modo que seja possível compará-lo ao algoritmo Random normalmente utilizado em memórias cache.

5 ESTUDO DE CASO

Neste trabalho, a arquitetura reconfigurável DIM será utilizada para avaliar a técnica de Memória Unificada implementada.

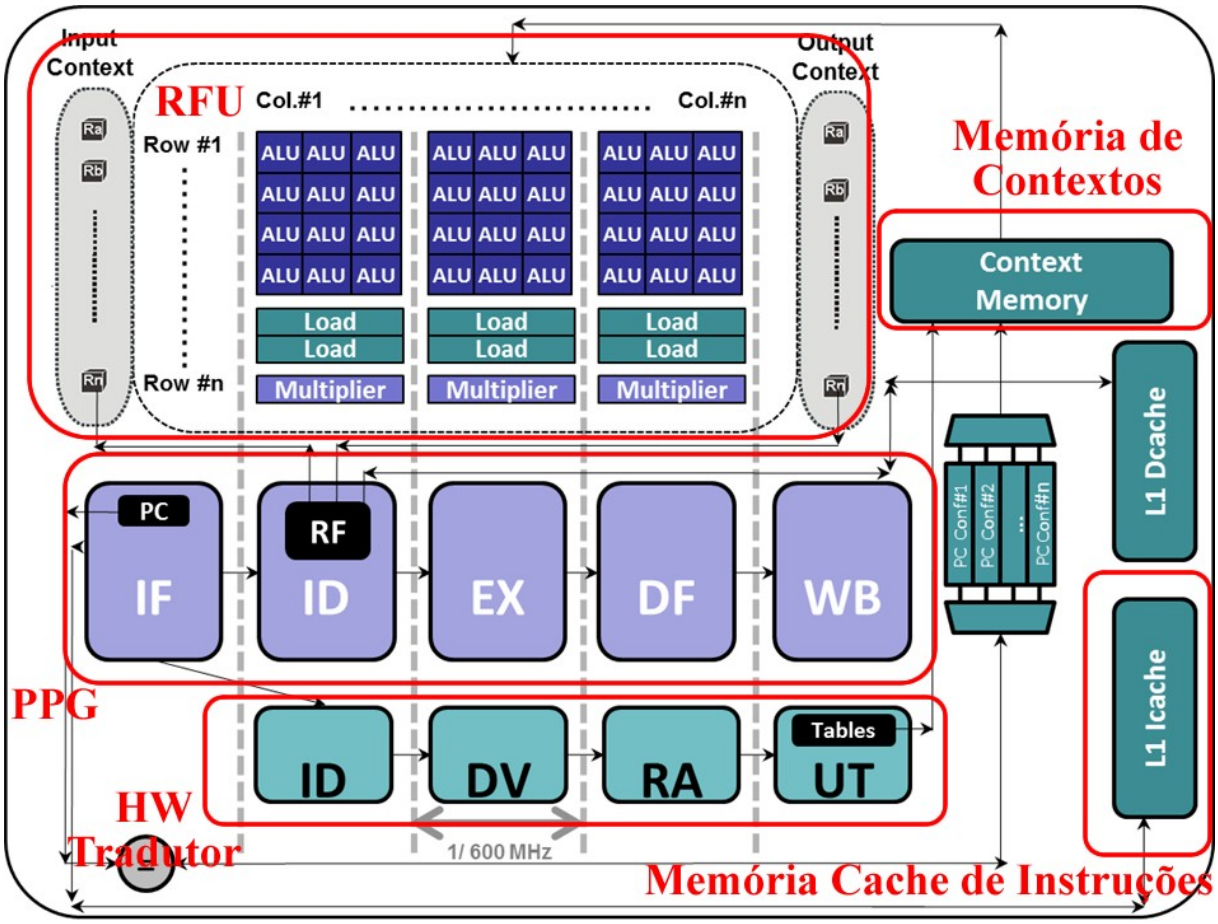

Figura 5.1 – A Arquitetura Reconfigurável DIM.

Através da Figura 5.1, é possível identificar o funcionamento da arquitetura reconfigurável DIM. O processador de propósito geral (PPG) se encarrega de rodar as instruções nativas da aplicação, utilizando a memória cache de instruções para isto. Enquanto isso, o hardware tradutor (HW Tradutor) se encarrega de identificar os pontos da aplicação que podem ser executados na RFU e, consequentemente, acelerados. Uma vez que o hardware tradutor tenha acabado de identificar um contexto de reconfiguração, o mesmo é salvo na memória de contextos, para que seja apenas executado quando necessário. É importante ressaltar que os contextos de reconfiguração são indexados na memória de contexto pelo endereço de memória da primeira instrução presente no contexto.

Dentro da RFU, existe uma divisão em linhas e colunas. Cada linha, conforme mostra a imagem, corresponde a um determinado bloco lógico (ou mais, no caso das ULAs, onde há três unidades), e cada coluna representa um conjunto de blocos lógicos. Instruções que necessitam ser executadas em série são armazenadas utilizando-se o conceito de linhas, e instruções que podem ser executadas em paralelo utilizam o conceito de colunas.

Diante de tal funcionamento, fica visível o uso das duas estruturas de memória (memória cache de instruções e memória de contextos) de forma separada. Apesar de esta ser uma possível abordagem, o fato de a mesma introduzir ao sistema um consumo extra de área e potência diante da proposta deste trabalho acaba tornando tal técnica desnecessária. Assim sendo, a Memória Unificada será acoplada a arquitetura reconfigurável DIM no lugar da memória cache de instruções e da memória de contextos, de modo que seja possível validar o correto funcionamento da mesma.

5.1 Implemetação da Proposta na Arquitetura Reconfigurável DIM

5.1.1 Implementação em Alto Nível da Memória Unificada

A técnica da Memória Unificada, para ser facilmente acoplada, foi implementada utilizando-se a mesma metodologia que a arquitetura reconfigurável DIM. Assim, a descrição do comportamento da Memória Unificada foi descrita em C++ e acoplada ao simulador da arquitetura reconfigurável.

Além disso, uma vez que a implementação anterior da arquitetura reconfigurável DIM não implementava a memória cache de instruções, somente a de reconfigurações, a mesma foi implementada, de modo que seja possível extrair os seus resultados e compará-los com os resultados da técnica de Memória Unificada.

Depois de ter sido definida qual seria a estrutura da Memória Unificada, também é necessário decidir as funções básicas que estarão presentes no sistema. Funções para adicionar uma informação à Memória Unificada, verificar se uma informação se encontra na mesma e invalidar uma informação são as três funções básicas que serão implementadas, inicialmente, no simulador.

A função responsável por adicionar um elemento à Memória Unificada funcionará da seguinte maneira: inicialmente, o PC (Program Counter) da instrução será tratado de modo que se consiga definir qual será seu rótulo e seu índice. Após esta etapa, será feita uma busca,

levando-se em consideração o tipo de informação, para ver se o PC já se encontra na Memória Unificada. Caso esteja presente, um acerto é somado e a função termina. Caso contrário, uma falta é somada e a Memória Unificada atualiza seus campos de modo que a informação passe a se encontrar presente na mesma.

Já a função de busca irá apenas retornar um valor verdadeiro se a informação estiver presente na Memória Unificada e um valor falso caso não esteja. Ainda assim, esta busca será capaz de somar acertos e faltas, tornando a simulação fiel à realidade.

Por último, a função de invalidação de informações irá apenas invalidar o bit de validade do bloco desejado, de modo que a informação será vista como não mais presente na Memória Unificada.

Uma vez expostos os campos e as funções da Memória Unificada, é necessário definir como a estrutura da mesma estará definida. Para isto, é necessário definir qual será o seu tamanho, sua associatividade, o número de palavras por bloco e o algoritmo de substituição que deverá ser utilizado na memória cache.

Entretanto, definir os parâmetros diretamente no simulador, de maneira estática, não é uma opção recomendável para este caso. Isto acontece pois será necessário testar diversas configurações de Memória Unificada para que seja possível garantir seu correto funcionamento. E, caso fosse necessário alterar diretamente o código do simulador da técnica de Memória Unificada e recompilar o mesmo toda vez que fossem alterados os parâmetros desejados, o tempo de simulação acabaria sendo muito alto.

Pensando nisso, o simulador de Memória Unificada trabalha com uma série de parâmetros que podem ser definidos e alterados na hora de iniciar sua execução, sendo, assim, capazes de alterar a configuração da estrutura de memória. Na Tabela 5.1, são apresentados quais são estes parâmetros, bem como o que irão definir dentro do simulador.

Parâmetro	**Função**
-P	Define o tamanho da Memória Cache Unificada (em KB)
-A	Define a Associatividade da Memória Cache Unificada
-B	Define o número de palavras por bloco da Memória Cache Unificada
-R	Define o Algoritmo de Substituição da Memória Cache Unificada (1 – Random; 2 – LRU)

Tabela 5.1 – Parâmetros do simulador da Memória Unificada.

Com a utilização destes parâmetros, será possível analisar diferentes configurações da Memória Unificada, de modo que seja possível, através de seus resultados, comprovar o correto funcionamento da mesma.

5.1.2 Implementação em VHDL da Memória Unificada

Uma vez que a técnica de Memória Unificada foi implementada na linguagem C++, e neste tipo de descrição não é possível extrair informações de área, potência e energia, a mesma também foi implementada na linguagem VHDL. Esta linguagem permite, além de simular e validar os algoritmos, extrair informações de área, potência e energia através do fluxo de síntese lógica.

O fluxo de síntese lógica é realizado através das ferramentas Cadence, e se encontra exemplificado na Figura 5.2.

Figura 5.2 – Fluxo de Síntese Lógica.

É necessário deixar claro que, ao contrário do algoritmo de alocação, que só funcionará na Memória Unificada, os algoritmos de substituição também farão parte do sistema quando as memória cache estiverem implementadas separadamente. Ou seja, os algoritmos irão responder e funcionar de maneiras distintas dentro destes dois cenários.

Assim sendo, os algoritmos de substituição foram implementados de duas maneiras: uma pensando em sua utilização dentro da Memória Unificada e outra pensando em memórias que atuarão separadamente. Esta tarefa é necessária uma vez que seus resultados de área, potência e energia serão diferentes, e, sendo assim, deverão ser analisados individualmente para que seja possível definir quais as reais diferenças entre os dois cenários aqui estudados.

Além de ser necessário extrair as informações de área, potência e energia dos algoritmos de substituição e de alocação, é necessário definir quais as informações serão extraídas das diferentes configurações de memória cache. Diferentes tamanhos, associatividades, entre outras características, resultam em diferentes resultados de área, potência e energia em memó-

rias cache. É necessário, então, ser capaz de retirar tais informações de todas as configurações que serão testadas neste trabalho.Para isto, foi utilizado o software CACTI, disponibilizado pela HP. Este software consegue extrair e fornecer todas as informações necessárias de uma memória cache, de modo que é apenas necessário preencher campos que irão definir a configuração requisitada. Os campos que serão preenchidos e alterados neste software se encontram na Tabela 5.2.

Parâmetro	**Função**
Cache size	Definirá o tamanho da cache, em bytes
Line size	Definirá o tamanho, em bytes, de cada índice da memória cache
Associativity	Definirá qual a associatividade da memória cache
Technology Mode	Definirá qual a tecnologia em que a memória cache será analisada. Como o fluxo de síntese lógica será realizado na tecnologia de 90nm, o mesmo valor será utilizado neste caso

Tabela 5.2 – Parâmetros do software CACTI.

Através destas tarefas implementadas e realizadas, é possível se aproximar ao máximo da realidade, uma vez que todas as estruturas desenvolvidas neste trabalho terão computadas suas informações de desempenho, área, potência e energia, e, assim sendo, poderão ser analisadas e comparadas durante o capítulo de resultados.

6 RESULTADOS

Neste capítulo, serão demostrados os resultados obtidos através da execução de diferentes aplicações em diversas configurações de Memória Unificada e Memórias Separadas.

6.1 Escolha das Aplicações e Configurações

As aplicações escolhidas possuem diferentes comportamentos permitindo avaliar o funcionamento da Memória Unificada em diferentes cenários. Assim sendo, foram escolhidas aplicações que, através de [6], mostraram diferentes comportamentos. São elas: *fft, lu, md, patricia, susan_c e susan_e*.

Após definir quais seriam as aplicações, é necessário definir quais as configurações de Memória Unificada serão testadas. Para esta tarefa, serão realizados testes extremos, ou seja, de melhor e pior caso. Desta forma, consegue-se dimiuir o número de execuções e validar de maneira mais clara a técnica proposta neste trabalho, visto que todo o restante dos resultados estarão no intervalo entre o pior e o melhor caso. Os parâmetros para definir estes casos para cada estrutura de memória estão presentes na Tabela 6.1.

Parâmetro	**Mínimo**	**Máximo**
Tamanho total	8KB	32KB
Associatividade	4	16
Palavras por Bloco	16	64
Slots de Reconfiguração (Memórias Separadas)	32	128
Contexto de Reconfiguração (Memórias Separadas)	64 bytes	256 bytes

Tabela 6.1 – Intervalo de Configurações de Memórias Cache analisados.

É necessário ressaltar que não deverá ser levado em consideração apenas os parâmetros das memórias cache, mas também o funcionamento da arquitetura reconfigurável, por exemplo: o melhor caso para a arquitetura reconfigurável é onde é possível obter o máximo de aceleração na execução da aplicação. Neste cenário, um grande número de unidades funcionais reconfiguráveis deve ser inserido na arquitetura.

As configurações de memória cache implementadas serão referenciados no restante do texto através da seguinte nomenclatura: Técnica / Tamanho Total / Palavras por Bloco, onde

SEP indicará a técnica de Memórias Separadas, UNI a técnica de Memória Unificada e o tamanho total será definido pela soma das duas memórias no caso de Memórias Separadas.

Por último, é necessário definir o limiar do hardware alocador. Visto que é necessário priorizar os contextos de reconfiguração. Esta priorização dos contextos de reconfiguração acontece pelo fato de que, uma vez que um contexto de reconfiguração é encontrado na memória cache, a arquitetura reconfigurável consegue acelerar a execução da aplicação, reduzindo, assim, o número total de ciclos gasto na execução da mesma.

Entretanto, é necessário também ressaltar que quando uma reconfiguração não é encontrada, nada acontece, ou seja, a arquitetura reconfigurável apenas deixa de acelerar a aplicação. Por outro lado, no caso de uma falta na busca por uma instrução, existe uma penalidade para que a mesma seja buscada no próximo nível da hierarquia de memória. Isto mostra que, apesar de priorizar os contextos de memória, é necessário manter uma boa taxa de acerto na busca por instruções, de modo que não se perca muitos ciclos buscando instruções em outros níveis da hierarquia de memória.

Assim sendo, a configuração do algoritmo de limiar escolhida segue os dados da Tabela 6.2.

Variável	**Valor**
Limiar	8
Máximo	10
Mínimo	0

Tabela 6.2 – Configuração do Algoritmo de Limiar.

Vale ressaltar que está configuração para o algoritmo de limiar foi escolhida através de testes, onde a mesma apresentou o melhor resultado dentro deste cenário de arquiteturas reconfiguráveis.

6.2 Metodologia para a Extração de Resultados de Simulação

Uma vez definidas as aplicações que serão executadas, as possíveis configurações que as memórias cache terão e o limiar do hardware alocador, resta definir as metodologias de teste e, então, testar a técnica de Memória Unificada em conjunto com as aplicações. Duas metodologias de comparação foram utilizadas para comparar a técnica de Memória Unificada com as Memórias Separadas. Primeiramente, para demostrar os ganhos da Memória Unificada é utili-

zada a metodologia de paridade de capacidade entre as técnicas. Após, é realizada a comparação de desempenho entre uma Memória Unificada de menor tamanho com Memórias Separadas de maior tamanho demostrando o potencial da técnica proposta. Neste caso, a Memória Unificada de 16KB será comparada com as Memórias Separadas de tamanho total igual a 32KB.

Além disso, foi utilizada a seguinte equação para definir o total de ciclos gastos com a estrutura de memória:

$$N^o \text{ Ciclos Total} = (Conf_{Hit} + Conf_{Miss}) + (Inst_{Hit} + (Inst_{Miss} * P))$$

Esta equação mostra que o número total de ciclos gasto com a técnica de memória proposta que é baseado nos acertos e faltas na memória cache. No caso de faltas durante a busca por instruções, será acrescentado uma penalidade P pela busca da instrução em um nível inferior de memória. Neste caso, foi utilizado o valor 3, que seria a penalidade causada pela busca na memória L2. Nos outros casos, apenas um ciclo será contabilizado.

Entretanto, esta equação mostra somente o total de ciclos gasto com a estrutura de memória, não contabilizando os ciclos consumidos pela execução da aplicação na arquitetura reconfigurável.

6.3 Resultados de Desempenho com Paridade de Capacidade

Para a paridade de capacidade, as configurações escolhidas para as técnicas de memória cache se encontram na Tabela 6.3.

Caso	Parâmetros Escolhidos
SEP/8KB/16PPB	Tamanho total: 8KB (4KB de Instrução + 4KB de Reconfiguração) Associatividade: 4 Palavras por bloco: 16 Slots de Reconfiguração: 64 Contexto de Reconfiguração: 64 bytes Blocos Lógicos: 12 linhas formadas por 3 ULAs, 2 LD/ST e 1 Multiplicador
SEP/16KB/16PPB	Tamanho total: 16KB (8KB de Instrução + 8KB de Reconfiguração) Associatividade: 4 Palavras por bloco: 16 Slots de Reconfiguração: 128 Contexto de Reconfiguração: 64 bytes Blocos Lógicos: 12 linhas formadas por 3 ULAs, 2 LD/ST e 1 Multiplicador
SEP/32KB/64PPB	Tamanho total: 32KB (16KB de Instrução + 16KB de Reconfiguração) Associatividade: 16 Palavras por bloco: 64 Slots de Reconfiguração: 64 Contexto de Reconfiguração: 256 bytes Blocos Lógicos: 39 linhas formadas por 4 ULAs, 3 LD/ST e 2 Multiplicadores
UNI/8KB/16PPB	Tamanho total: 8KB Associatividade: 4 Palavras por bloco: 16 Contexto de Reconfiguração: 64 bytes Blocos Lógicos: 12 linhas formadas por 3 ULAs, 2 LD/ST e 1 Multiplicador
UNI/16KB/16PPB	Tamanho total: 16KB Associatividade: 4 Palavras por bloco: 16 Contexto de Reconfiguração: 64 bytes Blocos Lógicos: 12 linhas formadas por 3 ULAs, 2 LD/ST e 1 Multiplicador
UNI/32KB/64PPB	Tamanho total: 32KB Associatividade: 16 Palavras por bloco: 64 Contexto de Reconfiguração: 256 bytes Blocos Lógicos: 39 linhas formadas por 4 ULAs, 3 LD/ST e 2 Multiplicadores

Tabela 6.3 – Configurações de Memórias Cache testadas com Paridade de Capacidade.

6.3.1 Taxa Média de Acerto de Contextos de Reconfiguração

A taxa média de acerto de contextos de reconfiguração para paridade de capacidade é demonstrada na Figura 6.1.

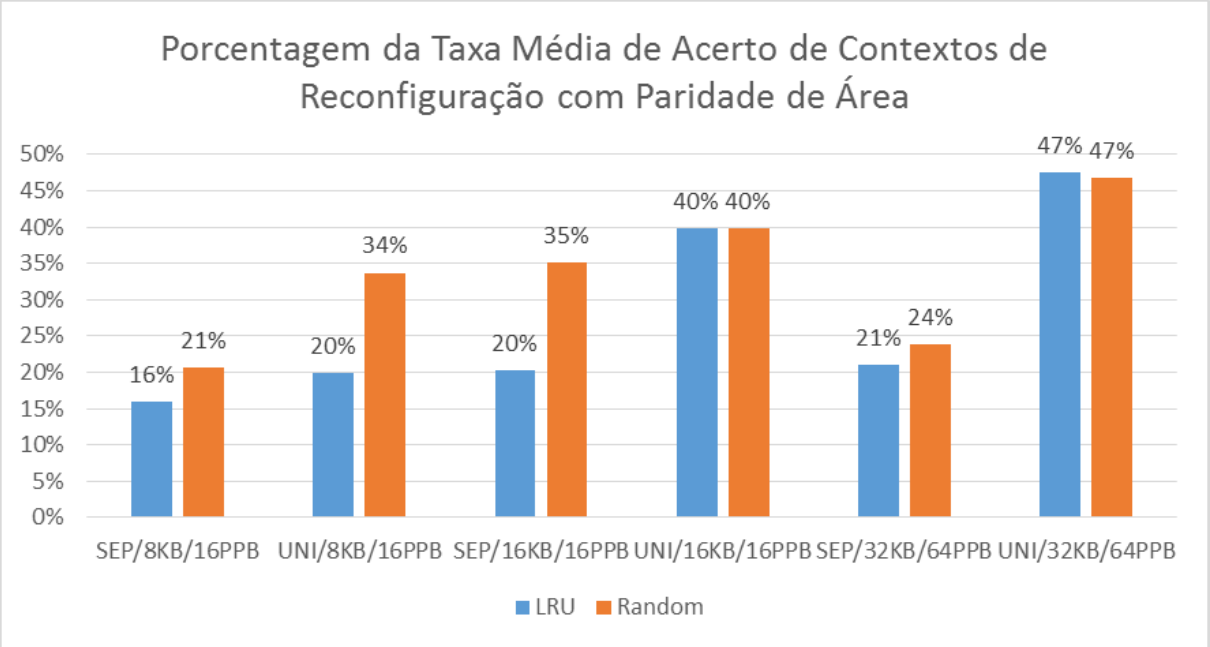

Figura 6.1 – Porcentagem da Taxa Média de Acerto de Contextos de Reconfiguração com Paridade de Capacidade.

É possível notar que, uma vez que a Memória Unificada prioriza os contextos de reconfiguração, a mesma consegue, em todos os casos com paridade de capacidade, aumentar a taxa média de acerto de contextos de reconfiguração. Desta maneira, as chances de ganho na execução da aplicação dentro da arquitetura reconfigurável acabam aumentando. Por exemplo, no caso de tamanho total de 32KB, a Memória Unificada consegue uma melhora, em relação a memória cache separada, de 26% com o algoritmo LRU e 23% com o algoritmo Random. O ganho médio, em todos os casos, chega a 16,66% no algoritmo LRU e 13,66% no algoritmo Random.

6.3.2 Desempenho Médio com Paridade de Capacidade

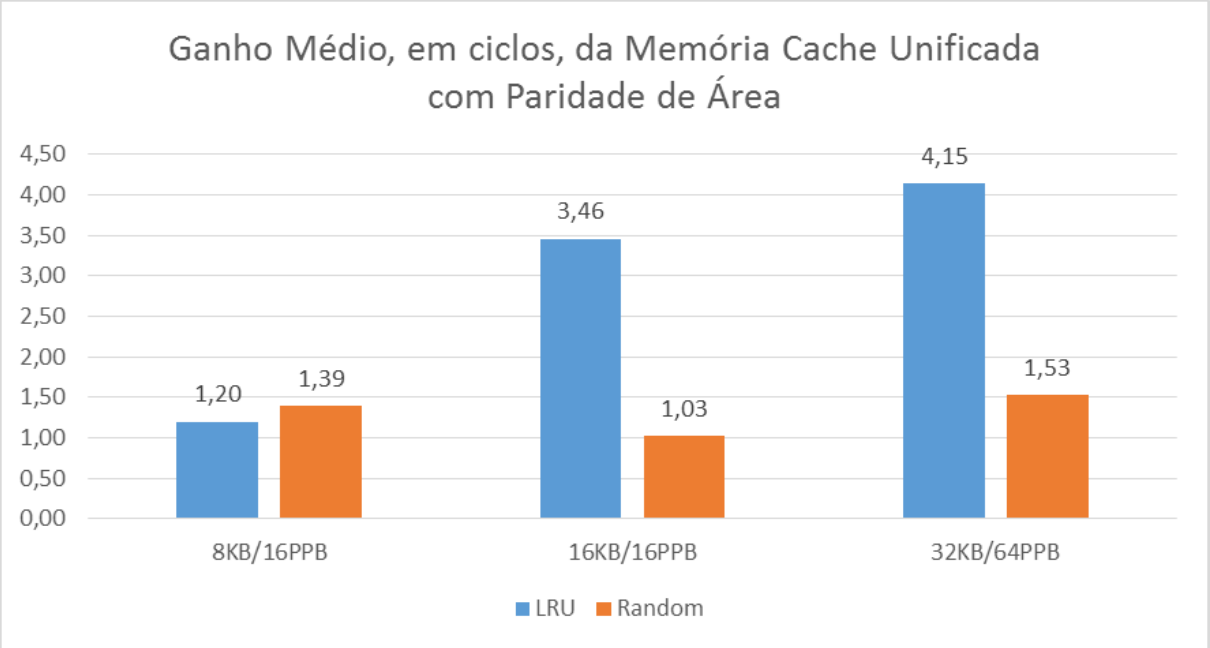

Figura 6.2 – Ganho Médio no Total de Ciclos Gastos com a Estrutura de Memória da Memória Unificada sobre a Memória Cache Separada levando em conta Paridade de Capacidade.

Através da Figura 6.2, é possível notar que a Memória Unificada tem uma melhora de desempenho expressiva no total de ciclos gastos com a estrutura de memória dentro do cenário de paridade de capacidade. A técnica de Memória Unificada sempre apresenta melhoras em relação a técnica de Memórias Separadas. No algoritmo LRU, a Memória Unificada consegue um ganho de 0.2x no caso de 8KB, 2.46x no de 16KB e 3.15x no de 32KB. Já no algoritmo Random, a Memória Unificada apresenta ganho de 0.39x no caso de 8KB, 0.03x no de 16KB e 0.53x no de 32KB. Tais dados mostram que, sendo considerada a média geral de todas as aplicações testadas, a Memória Unificada sempre apresenta ganhos se comparada a memória cache separada.

Além disso, é possível observar que o desempenho, dentro do algoritmo LRU, melhora conforme a tamanho total da memória cache aumenta. Uma vez que a tendência observada no mercado é aumentar o tamanho das memórias cache, esta característica acaba sendo muito importante e mostra que a técnica proposta escala com a tendência do mercado.

6.4 Resultados de Desempenho com Redução de Área

Para o caso de redução de área, as configurações escolhidas para as técnicas de memória cache se encontram na Tabela 6.4.

Caso	Parâmetros Escolhidos
SEP/32KB/64PPB	Tamanho total: 32KB (16KB de Instrução + 16KB de Reconfiguração) Associatividade: 16 Palavras por bloco: 64 Slots de Reconfiguração: 64 Contexto de Reconfiguração: 256 bytes Blocos Lógicos: 39 linhas formadas por 4 ULAs, 3 LD/ST e 2 Multiplicadores
UNI/16KB/64PPB	Tamanho total: 16KB Associatividade: 16 Palavras por bloco: 64 Contexto de Reconfiguração: 256 bytes Blocos Lógicos: 39 linhas formadas por 4 ULAs, 3 LD/ST e 2 Multiplicadores

Tabela 6.4 – Configurações de Memórias Cache testadas com Redução de Área.

É possível notar que, no caso da técnica de Memórias Separadas, a mesma configuração que foi utilizada no caso de paridade de capacidade com tamanho total de 32KB será executada. Entretanto, no caso da Memória Unificada, o número de palavras por bloco irá ser elevado para 64, de modo que seja possível realizar uma comparação justa.

6.4.1 Taxa Média de Acerto de Contextos de Reconfiguração e Instruções

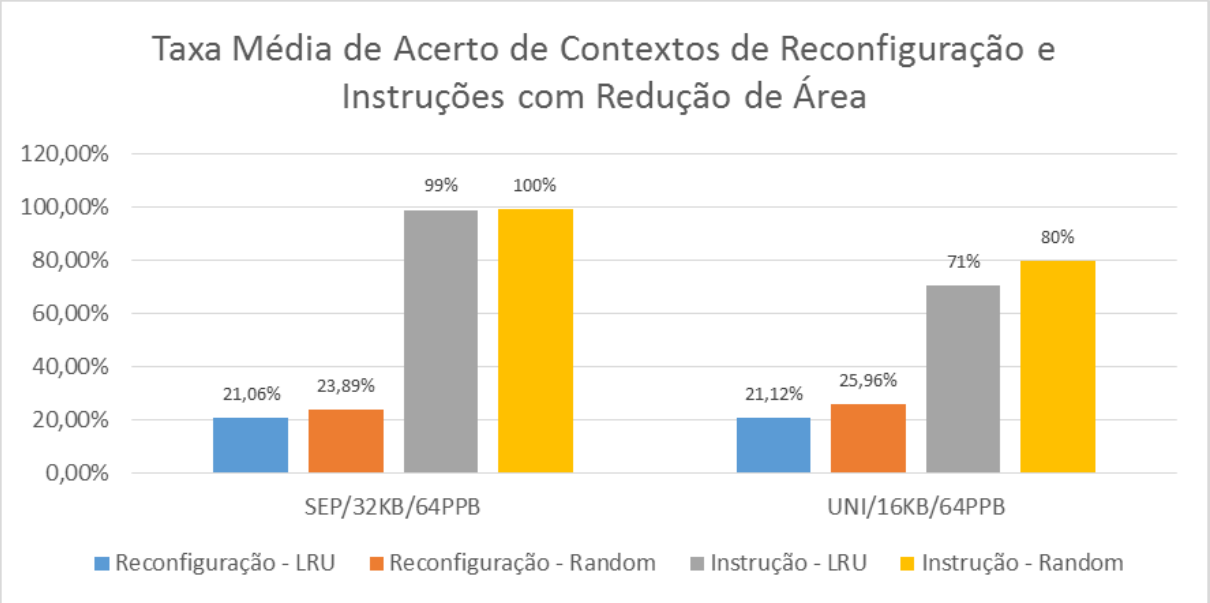

Figura 6.3 – Taxa Média de Acerto de Contextos de Reconfiguração e Instruções com Redução de Área.

Através da Figura 6.3, é possível notar que a Memória Unificada consegue aumentar a taxa de acerto de contextos de reconfiguração. Entretanto, a taxa de acertos para as instruções é reduzida. Entretanto, visto que o foco principal das arquiteturas reconfiguráveis é apresentar menor tempo de execução através dos contextos de reconfiguração executados na RFU, a perda de desempenho gerada pela queda da taxa de instruções é menor que os ganhos adquiridos pelo aumento da taxa de acerto de contextos de reconfiguração. Este comportamento poderá ser demonstrado posteriormente, na Seção 6.1.

6.4.2 Desempenho Médio com a Redução de Área

Levando-se em conta o desempenho total, em número de ciclos, e lembrando que a penalidade por uma falta de instrução é de três ciclos, é possível notar que a Memória Unificada proporciona uma perda de desempenho mínima se comparada a redução de área com a técnica de Memórias Separadas. Por exemplo, no caso do algoritmo LRU, a Memória Unificada de 16KB reduz em 17% o desempenho em relação a memória cache separada de 32KB. No caso do algoritmo Random, a diferença é ainda menor: a Memória Unificada de 16KB reduz em apenas 10% o desempenho comparada à memória cache separada de 32KB. Assim, mostra-se

que a inteligência provida pelo hardware alocador consegue atingir um desempenho geral muito próximo ao que Memórias Separadas, que somadas ocupam o dobro do tamanho da Memória Unificada. Esta redução no tamanho da memória cache irá prover redução no consumo de potência e energia, características de extrema importância para o cenário de sistemas embarcados.

6.5 Comparação de Desempenho na Aplicação susan_e

Para deixar mais evidente os ganhos da técnica proposta neste trabalho, esta seção explora em mais detalhes os resultados obtidos com a aplicação susan_e. Esta aplicação foi escolhida devido ao tempo disponível para extração de resultados desta natureza, que acaba por ser muito alto. Nesta seção, será feita tanto a comparação com paridade de capacidade como com redução de área. Assim sendo, na Figura 6.4 é explicitado o total de ciclos gastos na execução da aplicação susan_e nas configurações SEP/32KB/64PPB, UNI/16KB/64PPB e SEP/16KB/16PPB, onde o algoritmo de substituição utilizado é o Random. Ao contrário das seções anteriores, o total de ciclos gasto com a aplicação apresentados nesta seção contempla os ciclos gastos com a memória e os ciclos gastos na execução da aplicação no sistema reconfigurável.

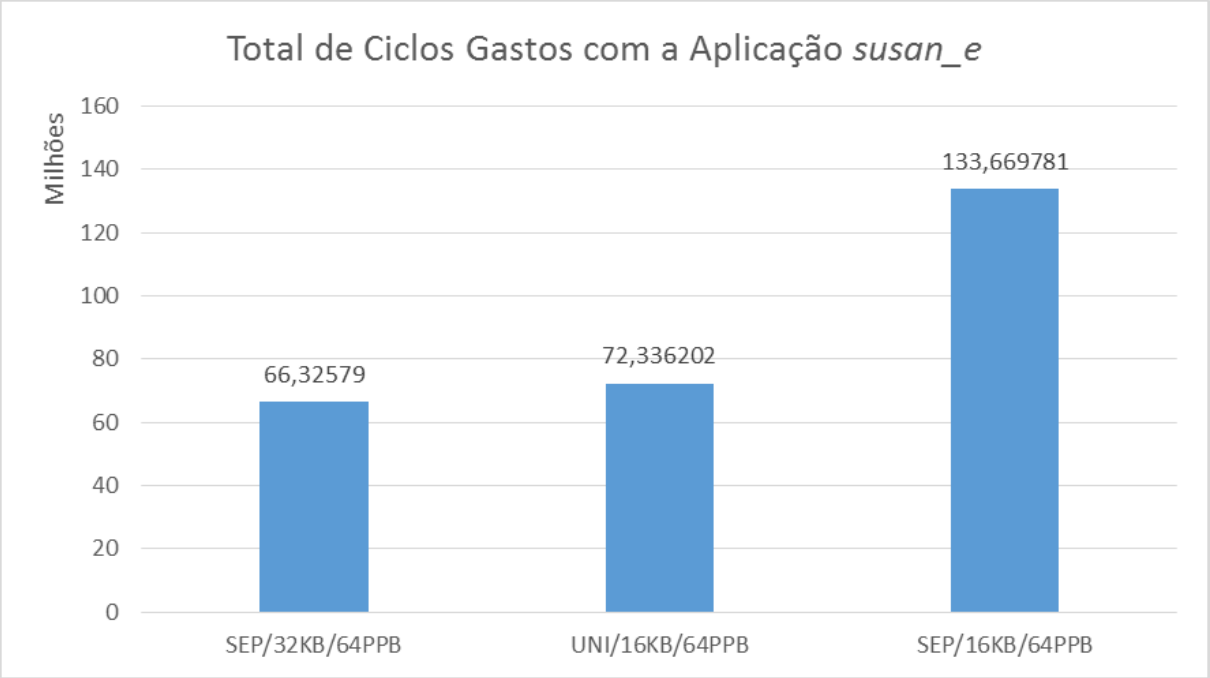

Figura 6.4 – Taxa Média de Acerto de Contextos de Reconfiguração e Instruções com Redução de Área.

Analisando a Figura 6.4 e levando-se em consideração a paridade de capacidade, onde são analisados os desempenhos das memórias UNI/16KB/64PPB e SEP/16KB/64PPB, é notável que, uma vez que a taxa de reconfigurações é maior na Memória Unificada devido a inteligência presente no hardware alocador, o ganho de desempenho através da técnica proposta neste trabalho é muito alto, chegando em 42%.

Por outro lado, analisando a redução de área dos casos SEP/32KB/64PPB e UNI/16KB/64PPB, a Memória Unificada, mesmo possuindo metade da área, apresenta uma perda de desempenho de apenas 8%. Por outro lado, se forem analisadas as Memórias Separadas SEP/32KB/64PPB e SEP/16KB/64PPB, a redução de área através da técnica de Memórias Separadas gera uma queda de desempenho de 50%, desempenho muito inferior a Memória Unificada com o mesmo tamanho.

6.6 Resultados de Área, Potência e Energia

Além dos resultados de desempenho das memórias cache analisadas, foi realizada a síntese lógica dos algoritmos de substituição e do hardware alocador para extrair os valores de área, potência e energia de cada. Os resultados estão explicitados na Tabela 6.5.

Algoritmo\Característica	**Área (um2)**	**Potência (nW)**
Algoritmo LRU para Memórias Separadas	2149,01	33955,52
Algoritmo LRU para Memória Unificada	3117,44	108310,81
Algoritmo Random para Memórias Separadas	212,13	0,001
Algoritmo Random para Memória Unificada	700,96	12437,80
Algoritmo de Limiar	4070,51	59276,84

Tabela 6.5 – Dados de Área e Potência para os Algoritmos Utilizados.

É notável que todos os algoritmos de substituição utilizados dentro da Memória Unificada acabam consumindo mais área e potência. Isto acontece pelo fato de que é necessário comparar o campo tipo de informação dos blocos que serão substituídos, aumentando, assim, o gasto com área e potência. Além disso, no caso da Memória Unificada, ainda é necessário inserir o algoritmo de limiar no sistema, o que representa um acréscimo de área e potência.

Entretanto, os valores de área e potência, tanto dos algoritmos de substituição quanto

do algoritmo de limiar, utilizado pelo hardware alocador, são negligíveis em relação à área e potência do restante do sistema.

Por fim, foram extraídas as informações de cada configuração de memória cache simulada através do software CACTI [16]. Os resultados obtidos através do mesmo se encontram na Tabela 6.6.

Tamanho	**Associatividade**	**Palavras por Bloco**	**Área (mm2)**	**Potência Total (mW)**	**Tempo de Acesso (ns)**
4KB	4	16	0,07	2,23	0,75
8KB	4	16	0,14	4,16	0,79
16KB	4	16	0,28	7,73	0,86
16B	16	64	0,28	7,73	0,86
32KB	16	64	0,49	15,36	1,01

Tabela 6.6 – Dados de Área e Potência para as configurações de Memória Cache testadas.

A Tabela 6.6 mostra os dados de área e potência de todas as configurações de memória cache testadas neste trabalho. E, através da análise da mesma, é possível notar uma importante característica: as áreas e potências consumidas pelas configurações de memória cache são muito maiores que as consumidas pelos algoritmos utilizados neste trabalho. Isto mostra que os dados extraídos dos algoritmos terão um impacto menor no total de área e potência do sistema, enquanto as informações retiradas das configurações de memória cache serão determinantes.

6.7 Comparação de Área e Potência das Configurações Simuladas

Através da síntese lógica realizada no algoritmo alocador e nos algoritmos de substituição e das informações extraídas do software CACTI, é possível fazer a comparação de área e potência dos casos simulados.

Assim sendo, a Tabela 6.7 mostra as comparações de área e potência dos casos simulados, tanto dos casos com paridade de capacidade como do caso com redução de área da Memória Unificada.

Algoritmo\Característica	Área (um2)	Potência (nW)	Algoritmo de Substituição
SEP/8KB/16PPB	0,15	4,51	LRU
SEP/8KB/16PPB	0,15	4,47	Random
UNI/8KB/16PPB	0,14	4,33	LRU
UNI/8KB/16PPB	0,14	4,23	Random
SEP/16KB/16PPB	0,29	8,35	LRU
SEP/16KB/16PPB	0,29	8,32	Random
SEP/16KB/64PPB	0,29	8,35	Random
SEP/16KB/64PPB	0,29	8,32	Random
UNI/16KB/16PPB	0,28	7,90	LRU
UNI/16KB/16PPB	0,28	7,80	Random
SEP/32KB/64PPB	0,57	15,50	LRU
SEP/32KB/64PPB	0,57	15,46	Random
UNI/32KB/64PPB	0,49	15,53	LRU
UNI/32KB/64PPB	0,49	15,43	Random
UNI/16KB/64PPB	0,28	7,90	LRU
UNI/16KB/64PPB	0,28	7,80	Random

Tabela 6.7 – Relação total de Área e Potência das Memórias Cache analisadas.

É interessante observar, através da Tabela 6.7, que mesmo nos casos com paridade de capacidade existe uma redução de área e potência quando utilizada a técnica de Memória Unificada. Isto acontece pelo fato de ser mais custoso utilizar duas memórias de 4KB para obter tamanho total de 8KB devido ao fato de alguns componentes da estrutura estarem duplicados, consumindo uma maior área e potência do que utilizar apenas uma memória de 8KB. Essa característica, aliada ao tamanho e potência negligíveis dos algoritmos de substituição e hardware alocador, faz com que a área e a potência total, mesmo nos casos de paridade de capacidade, seja menor na Memória Unificada.

Sendo assim, através da análise de todos os dados obtidos, será mostrado um quadro comparativo com a média das aplicações para as seguintes características: Ganho Médio, Ganho Médio da Taxa de Acerto de Contextos de Reconfiguração, Redução de Área e Redução de Potência. Estas características foram escolhidas pelo fato de serem os principais fatores que decidirão se a Memória Unificada apresentou ou não um desempenho satisfatório para o cenário de sistemas embarcados.

O quadro comparativo será dividido, no caso da paridade de capacidade, levando-se em consideração o tamanho total da memória cache simulada e o número de palavras por bloco da mesma. No caso de redução de área, será feita a comparação entre as estruturas de memória

SEP/32KB/64PPB e UNI/16KB/64PPB.

Logo, o quadro comparativo é exposto na Tabela 6.8.

Caso\Característica	**Ganho Médio no Total de Ciclos Gastos com a Estrutura de Memória**	**Ganho Médio da Taxa de Acerto de Contextos de Reconfiguração**	**Redução de Área**	**Redução de Potência**
Paridade de Capacidade 8KB/16PPB Algoritmo LRU	1,20	1,24	1,07	1,04
Paridade de Capacidade 8KB/16PPB Algoritmo Random	1,38	1,48	1,07	1,05
Paridade de Capacidade 16KB/16PPB Algoritmo LRU	3,46	2	1,01	1,05
Paridade de Capacidade 16KB/16PPB Algoritmo Random	1,03	1,14	1,01	1,06
Paridade de Capacidade 32KB/64PPB Algoritmo LRU	4,14	2,25	1,16	0,99
Paridade de Capacidade 32KB/64PPB Algoritmo Random	1,53	1,63	1,16	1
Redução de Área Memória Unificada Algoritmo LRU	0,82	1	2	1.96
Redução de Área Memória Unificada Algoritmo Random	0,90	1,07	2	1,98
Redução de Área Memórias Separadas Algoritmo LRU	0,79	0,90	1,96	1,85
Redução de Área Memórias Separadas Algoritmo Random	0,67	0,86	1,96	1,85

Tabela 6.8 – Quadro comparativo das Memórias Cache analisadas.

Através da Tabela 6.8, é possível notar que, levando-se em conta a paridade de capacidade, a Memória Unificada consegue atingir um ganho, para o algoritmo LRU, de 20% para tamanho total de 8KB, 246% para tamanho total de 16KB e 314% para tamanho total de 32KB.

No caso do algoritmo Random, os ganhos foram de 37,8% para 8KB, 3% para 16KB e 53,2% para 32KB. Esses ganhos acontecem pela existência do hardware alocador na Memória Unificada, que, através de sua inteligência, consegue garantir o melhor desempenho possível para o cenário de arquiteturas reconfiguráveis.

No caso da redução de área, a Memória Unificada apresentou melhoras significativas dentro do cenário de sistemas embarcados, uma vez que conseguiu apresenta uma perda mínima de desempenho reduzindo quase pela metade a área e a potência gastas. No caso do algoritmo LRU, a Memória Unificada teve uma perda de desempenho de 17,2%, enquanto a área e a potência praticamente foram reduzidas pela metade. Já no caso do algoritmo Random, enquanto a área e a potência continuam reduzidas pela metade, a perda de desempenho fica em apenas 9,6%.

7 CONCLUSÃO

Neste trabalho foi proposta uma técnica que unifica, em uma mesma estrutura da memória cache, contextos de reconfiguração e instruções nativas do processador para atingir a redução de área, potência e energia do sistema. Através dos resultados é possível identificar que a técnica de Memória Unificada atinge ganhos significativos dentro dos cenários analisados.

Na metodologia de paridade de capacidade, a Memória Unificada atinge ganhos em todos os casos, fato que por si só já leva a mesma a ser a melhor escolha para utilização em arquiteturas reconfiguráveis embarcadas. Além disso, em todos os casos, foi possível obter uma redução de área e potência, características importantes dentro do cenário estudado. Isto acontece pelo fato de o hardware alocador e o algoritmo de substituição utilizados na Memória Unificada não acabarem inserindo um consumo de potência e área consideráveis ao sistema, se comparados aos elementos utilizados na técnica de Memórias Separadas.

No caso de redução de área, os resultados obtidos foram ainda melhores se levado em consideração que todos os casos analisados possuem a maior aceleração proposta neste trabalho da arquitetura reconfigurável DIM. O desempenho médio foi pouco comprometido, devido à alta eficiência do hardware alocador. No caso do ganho da taxa de acerto de contextos de reconfiguração, o hardware alocador também conseguiu garantir uma melhora, mesmo que não tão expressiva. Por outro lado, a potência e a área foram quase que diminuídas pela metade, como era esperado. Se comparada a redução de área da Memória Unificada com a redução de área da técnica de Memórias Separadas, a Memória Unificada consegue atingir sempre um melhor compromisso entre desempenho, área e potência.

Logo, fica visível a contribuição da proposta deste trabalho, uma vez que a técnica de Memória Unificada consegue oferecer uma nova estrutura de memória que é capaz de obter: com paridade de capacidade, um desempenho muito superior a estruturas de memórias separadas; com redução de área, um desempenho próximo ao da técnica de Memórias Separadas, mesmo ocupando a metade da área e dissipando a metade da potência.

Como trabalhos futuros, se espera testar novos algoritmos de substituição e alocação, de modo que se busque um desempenho ainda maior da Memória Unificada. Além disso, se espera testar novas abordagens de dimensionamento da Memória Unificada, para que seja possível comparar com a abordagem utilizada neste trabalho e optar pela que apresente os melhores resultados.

REFERÊNCIAS

[1] - Joseph A. Paradiso and Thad Starner. 2005. **Energy Scavenging for Mobile and Wireless Electronics**. IEEE Pervasive Computing 4, 1 (Janeiro 2005), 18-27.

[2] - Ray Kurzweil. 2011. **The Singularity is Near**: When Humans Transcend Biology. Tantor Media.

[3] - David A. Patterson and John L. Hennessy. 2011. **Computer Organization and Design**, Revised Fourth Edition, Fourth Edition: The Hardware/Software Interface (4th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[4] - Zhi Alex Ye, Andreas Moshovos, Scott Hauck, and Prithviraj Banerjee. 2000. **CHIMAERA**: a high-performance architecture with a tightly-coupled reconfigurable functional unit. SIGARCH Comput. Archit. News 28, 2 (Maio 2000), 225-235.

[5] - Hauser, J.R.; Wawrzynek, J., "**Garp**: a MIPS processor with a reconfigurable coprocessor,"Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on , vol., no., pp.12,21, 16-18 Abril 1997.

[6] - Mateus B. Rutzig, Antonio Carlos S. Beck, and Luigi Carro. 2011. **CReAMS**: an embedded multiprocessor platform. In Proceedings of the 7th international conference on Reconfigurable computing: architectures, tools and applications (ARC'11), Andreas Koch, Ram Krishnamurthy, John McAllister, Roger Woods, and Tarek El-Ghazawi (Eds.). Springer-Verlag, Berlin, Heidelberg, 118-124.

[7] - Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, and R. Reed Taylor. 2000. **PipeRench**: A Reconfigurable Architecture and Compiler. Computer 33, 4 (Abril 2000), 70-77.

[8] - Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis. 2007. **The Molen compiler for reconfigurable processors**. ACM Trans. Embed. Comput. Syst. 6, 1, Article 6

(Fevereiro 2007).

[9] - Chuanjun Zhang, Frank Vahid, and Walid Najjar. 2003. **Energy Benefits of a Configurable Line Size Cache for Embedded Systems**. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'03) (ISVLSI '03). IEEE Computer Society, Washington, DC, USA, 87-.

[10] - Mingming Zhang, Xiaotao Chang, and Ge Zhang. 2007. **Reducing cache energy consumption by tag encoding in embedded processors**. In Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED '07). ACM, New York, NY, USA, 367-370.

[11] - Manish Verma, Lars Wehmeyer, and Peter Marwedel. 2004. **Dynamic overlay of scratchpad memory for energy minimization**. In Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '04). ACM, New York, NY, USA, 104-109.

[12] - Luca Benini, Alberto Macii, Enrico Macii, and Massimo Poncino. 1999. **Selective instruction compression for memory energy reduction in embedded systems**. In Proceedings of the 1999 international symposium on Low power electronics and design (ISLPED '99). ACM, New York, NY, USA, 206-211.

[13] - Yoonjin Kim, Ilhyun Park, Kiyoung Choi, and Yunheung Paek. 2006. **Power-conscious configuration cache structure and code mapping for coarse-grained reconfigurable architecture**. In Proceedings of the 2006 international symposium on Low power electronics and design (ISLPED '06). ACM, New York, NY, USA, 310-315.

[14] - Thiago Barcellos Ló. **Virtualização de Hardware e Exploração da Memória de Contexto em Arquiteturas Reconfiguráveis**. 2012. 96 f. Dissertação (Mestrado em Ciências da Computação) – UFRGS, Rio Grande do Sul. 2012.

[15] - Claudio Brunelli, Fabio Garzia, Davide Rossi, and Jari Nurmi. 2010. **A coarse-grain reconfigurable architecture for multimedia applications supporting subword and floating-point calculations**. J. Syst. Archit. 56, 1 (Janeiro 2010), 38-47.

[16] - Software CACTI. Disponível em: http://www.hpl.hp.com/research/cacti/.

ANEXOS

ANEXO A – Título do Anexo

Tabelas de Resultados das Aplicações

A.1 Algoritmo LRU

FFT

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	601318	5943	0,990213	121219	427272	0,221004538	722537	433215	0,625166126
UNI/8KB/16PPB	419474	14412	0,966784	133175	204777	0,394064838	552649	219189	0,716016833
SEP/16KB/16PPB	415224	1474	0,996463	134025	187608	0,416701644	549249	189082	0,743906188
UNI/16KB/16PPB	395650	5836	0,985464	135086	175580	0,434827113	530736	181416	0,745256631
SEP/32KB/64PPB	485100	379	0,999219	127313	246434	0,340639524	612413	246813	0,712749614
UNI/32KB/64PPB	370436	18388	0,952709	133002	163411	0,448705016	503438	181799	0,734691793
UNI/16KB/64PPB	426564	54977	0,885831	127552	241279	0,345827764	554116	296256	0,651615999
SEP/16KB/64PPB	1157765	5011	0,99569	82090	1143349	0,066988239	1239855	1148360	0,51915552

MD

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	491897	886	0,998202	111989	208270	0,349682601	603886	209156	0,742748837
UNI/8KB/16PPB	460558	15854	0,966722	113148	186787	0,377241736	573706	202641	0,738981409
SEP/16KB/16PPB	475438	445	0,999065	113199	194270	0,368163945	588637	194715	0,751433583
UNI/16KB/16PPB	459409	14134	0,970153	113433	181193	0,385006754	572842	195327	0,745723923
SEP/32KB/64PPB	252585	185	0,999268	104932	99079	0,514344815	357517	99264	0,782687984
UNI/32KB/64PPB	229305	20711	0,917161	105202	98419	0,516655944	334507	119130	0,737389146
UNI/16KB/64PPB	230608	22301	0,911822	104922	99124	0,514207581	335530	121425	0,734273616
SEP/16KB/64PPB	363291	371	0,99898	98111	221514	0,30695659	461402	221885	0,675268226

LU

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	255346	1470	0,994276	60438	125044	0,325842939	315784	126514	0,71396208
UNI/8KB/16PPB	250461	2762	0,989093	60906	121945	0,333090877	311367	124707	0,714023308
SEP/16KB/16PPB	251620	876	0,996531	60989	121360	0,334463035	312609	122236	0,718897538
UNI/16KB/16PPB	245558	1484	0,993993	61608	117673	0,343639315	307166	119157	0,720500653
SEP/32KB/64PPB	256172	283	0,998896	60599	113068	0,34893791	316771	113351	0,736467793
UNI/32KB/64PPB	233281	14932	0,939842	61370	110752	0,356549424	294651	125684	0,700990876
UNI/16KB/64PPB	238434	18456	0,928156	60562	113221	0,348492085	298996	131677	0,694252948
SEP/16KB/64PPB	258523	652	0,997484	60269	114254	0,345335572	318792	114906	0,735055269

Patricia

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	42749947	7196963	0,855908	2148249	92557814	0,022683331	44898196	99754777	0,310385574
UNI/8KB/16PPB	35000416	9858851	0,780227	2836929	82918155	0,033081759	37837345	92777006	0,289687501
SEP/16KB/16PPB	39741678	5206556	0,884166	2825652	82815333	0,032994156	42567330	88021889	0,325963585
UNI/16KB/16PPB	35169916	8691453	0,801843	3075430	79930638	0,037050665	38245346	88622091	0,301459121
SEP/32KB/64PPB	51101852	2653931	0,95063	1795198	97994013	0,017989901	52897050	100647944	0,344505207
UNI/32KB/64PPB	31473694	14323178	0,687245	2749885	81290812	0,032720873	34223579	95613990	0,263587645
UNI/16KB/64PPB	26299979	27460515	0,489206	1794728	98013641	0,017981739	28094707	125474156	0,182945334
SEP/16KB/64PPB	50431921	3333689	0,937996	1753448	98984262	0,017406074	52185369	102317951	0,337762121

Susan_e

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	73050145	8510	0,999884	5067632	130648773	0,037339863	78117777	130657283	0,374171977
UNI/8KB/16PPB	62751462	8798171	0,877034	5173739	127971741	0,038857789	67925201	136769912	0,331835968
SEP/16KB/16PPB	70919783	7339	0,999897	5193256	126691724	0,03937716	76113039	126699063	0,375288448
UNI/16KB/16PPB	12420060	229561	0,981852	6330074	3256351	0,660316437	18750134	3485912	0,843231481
SEP/32KB/64PPB	67768191	2942	0,999957	4809729	126099860	0,036740846	72577920	126102802	0,365299256
UNI/32KB/64PPB	14215128	534755	0,963745	5837997	1224007	0,826677102	20053125	1758762	0,919366811
UNI/16KB/64PPB	35955635	32257640	0,527106	4796914	127072291	0,036376302	40752549	159329931	0,203678748
SEP/16KB/64PPB	79780957	3733	0,999953	4246453	151654052	0,027238225	84027410	151657785	0,356523922

Susan_c

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	72332223	8071	0,999888	502123	142758601	0,003504959	72834346	142766672	0,337820047
UNI/8KB/16PPB	35482192	5066780	0,875045	1478914	72747250	0,01992443	36961106	77814030	0,322030601
SEP/16KB/16PPB	40398608	6858	0,99983	1486695	72419604	0,020115944	41885303	72426462	0,36641288
UNI/16KB/16PPB	8027636	305311	0,963361	2978259	2703425	0,524185963	11005895	3008736	0,785314647
SEP/32KB/64PPB	67744697	2912	0,999957	662403	132144637	0,00498771	68407100	132147549	0,341089575
UNI/32KB/64PPB	11326682	336498	0,971149	2802934	1395851	0,667558353	14129616	1732349	0,890785978
UNI/16KB/64PPB	35782227	34532093	0,50889	564655	138103694	0,004071982	36346882	172635787	0,173922949
SEP/16KB/64PPB	72252300	3669	0,999949	486624	142615106	0,003400546	72738924	142618775	0,337758642

A.2 Algoritmo Random

FFT

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	627409	12343	0,980707	117358	483358	0,195363533	744767	495701	0,600391949
UNI/8KB/16PPB	418854	16471	0,962164	132557	207412	0,38990908	551411	223883	0,711228257
SEP/16KB/16PPB	423327	2221	0,994781	133189	196625	0,403830644	556516	198846	0,736754033
UNI/16KB/16PPB	395650	5836	0,985464	135086	175580	0,434827113	530736	181416	0,745256631
SEP/32KB/64PPB	615385	1150	0,998135	115441	418919	0,216036006	730826	420069	0,635006669
UNI/32KB/64PPB	384127	20354	0,949679	131612	169602	0,436938522	515739	189956	0,730824223
UNI/16KB/64PPB	475815	84357	0,849409	120489	326371	0,269634785	596304	410728	0,592140071
SEP/16KB/64PPB	931270	7925	0,991562	92835	852628	0,098189987	1024105	860553	0,543390366

MD

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	497902	1485	0,997026	111310	213019	0,343200886	609212	214504	0,739589859
UNI/8KB/16PPB	461627	16362	0,965769	112964	191963	0,370462439	574591	208325	0,733911429
SEP/16KB/16PPB	476273	510	0,99893	113060	188961	0,37434483	589333	189471	0,756715425
UNI/16KB/16PPB	459409	14134	0,970153	113433	181193	0,385006754	572842	195327	0,745723923
SEP/32KB/64PPB	260583	219	0,99916	104394	102455	0,504686994	364977	102674	0,780447385
UNI/32KB/64PPB	230395	20917	0,916769	105057	98790	0,515371823	335452	119707	0,736999598
UNI/16KB/64PPB	237173	24027	0,908013	104371	102868	0,503626248	341544	126895	0,729110941
SEP/16KB/64PPB	363670	685	0,99812	98105	211448	0,316924727	461775	212133	0,685219644

LU

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	254513	1627	0,993648	60393	124927	0,325884956	314906	126554	0,713328501
UNI/8KB/16PPB	249533	2682	0,989366	60989	121191	0,334773301	310522	123873	0,714837878
SEP/16KB/16PPB	251342	1006	0,996013	60900	121693	0,333528668	312242	122699	0,717895071
UNI/16KB/16PPB	245558	1484	0,993993	61608	117673	0,343639315	307166	119157	0,720500653
SEP/32KB/64PPB	257913	603	0,997667	60244	114500	0,344755757	318157	115103	0,734332733
UNI/32KB/64PPB	238247	15635	0,938416	60773	112909	0,349909605	299020	128544	0,699357289
UNI/16KB/64PPB	240782	19059	0,926651	60175	115337	0,34285405	300957	134396	0,691294191
SEP/16KB/64PPB	260313	781	0,997009	59992	115438	0,341971157	320305	116219	0,733762634

Patricia

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	42495648	5724433	0,881285	2488944	89460499	0,027068614	44984592	95184932	0,320929905
UNI/8KB/16PPB	31010658	9578478	0,764014	3665648	74335389	0,046994862	34676306	83913867	0,292404549
SEP/16KB/16PPB	36754903	3287090	0,917909	3621264	73763792	0,046795392	40376167	77050882	0,34384043
UNI/16KB/16PPB	35169916	8691453	0,801843	3075430	79930638	0,037050665	38245346	88622091	0,301459121
SEP/32KB/64PPB	47741583	1003837	0,979407	2176791	93170443	0,022830143	49918374	94174280	0,34643247
UNI/32KB/64PPB	24888695	12556717	0,664666	3876975	68340690	0,05368458	28765670	80897407	0,262309528
UNI/16KB/64PPB	21871217	24785053	0,468773	2460723	88554590	0,027036363	24331940	113339643	0,176739015
SEP/16KB/64PPB	49888695	3213624	0,939482	1593248	100006287	0,015681647	51481943	103219911	0,332781681

Susan_e

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	21492408	15719	0,999269	6017700	18914560	0,241361994	27510108	18930279	0,59237465
UNI/8KB/16PPB	13142930	838500	0,940028	6256179	5162261	0,547901377	19399109	6000761	0,763748358
SEP/16KB/16PPB	13493883	8891	0,999342	6288052	4483199	0,58378103	19781935	4492090	0,814942516
UNI/16KB/16PPB	12420060	229561	0,981852	6330074	3256351	0,660316437	18750134	3485912	0,843231481
SEP/32KB/64PPB	28869441	6276	0,999783	5628305	23941806	0,190337635	34497746	23948082	0,590251643
UNI/32KB/64PPB	14889460	630167	0,959395	5818385	1631427	0,781010984	20707845	2261594	0,901538997
UNI/16KB/64PPB	22078479	5771447	0,792766	5613749	19699013	0,221775443	27692228	25470460	0,520895934
SEP/16KB/64PPB	48977973	14644	0,999701	5230058	71542522	0,068124036	54208031	71557166	0,431025691

Susan_c

	Hit Inst	Miss Inst	Taxa Inst	Hit Reconf	Miss Reconf.	Taxa Reconf.	Total HIT	Total MISS	Taxa Total
SEP/8KB/16PPB	15933179	14293	0,999104	2483238	20557591	0,107775549	18416417	20571884	0,472357516
UNI/8KB/16PPB	8652520	1019717	0,894573	2853150	5761882	0,331182751	11505670	6781599	0,629162835
SEP/16KB/16PPB	9395116	8726	0,999072	2880182	5002417	0,365384818	12275298	5011143	0,710111353
UNI/16KB/16PPB	8027636	305311	0,963361	2978259	2703425	0,524185963	11005895	3008736	0,785314647
SEP/32KB/64PPB	21341356	6081	0,999715	2539167	13870999	0,154731342	23880523	13877080	0,632469254
UNI/32KB/64PPB	11398690	349492	0,970251	2814709	1416728	0,665189863	14213399	1766220	0,889470456
UNI/16KB/64PPB	17163774	3331434	0,837453	2562661	10741558	0,192620176	19726435	14072992	0,583632231
SEP/16KB/64PPB	42409200	12719	0,9997	1778187	62424458	0,027696476	44187387	62437177	0,41442033