CRISTÓFERSON GUIMARÃES MAGALHÃES BUENO

SUPER-VLIW: UMA ARQUITETURA DINAMICAMENTE RECONFIGURÁVEL COM TOLERÂNCIA A FALHA

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2011

CRISTÓFERSON GUIMARÃES MAGALHÃES BUENO

SUPER-VLIW: UMA ARQUITETURA DINAMICAMENTE RECONFIGURÁVEL COM TOLERÂNCIA A FALHA

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 04 de fevereiro de 2011.

Carlos de Castro Goulart
(Co-Orientador)

Clarindo Isaías P. da Silva e Pádua

Ricardo dos Santos Ferreira
(Orientador)

Dedico essa dissertação aos meus pais
Geraldo e Norma

Aos meus irmãos
Kamila, Kelson e Kívia

A minha namorada
Isabella

Aos meus avós, em especial minha avó
Angélica que hoje se encontra
ao lado de Deus.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus que me deu força e coragem para realizar este trabalho. Agradeço por sempre colocar as pessoas certas em meu caminho; me auxiliando não só neste trabalho, como nos demais desafios da vida.

Agradeço também aos meus pais, Geraldo e Norma, e aos meus irmãos Kamila, Kelson e Kívia pelo incentivo, apoio e momentos de alegrias. Agradeço a todos que acreditaram em mim, amigos, familiares e colegas de trabalho, professores; sem dúvida não suportaria todas as adversidades sem vocês na minha vida. Aos meus pais, obrigado por toda a educação que recebi e pela oportunidade que me deram de realizar meus sonhos. Vocês são exemplos de vida para mim. Aos meus avós deixo registrado aqui os meus mais sinceros agradecimentos, vocês são o exemplo de carinho, dedicação, perseverança e fé que eu me espelho. Um obrigado especial para minha avó Angélica, que infelizmente não posso mais desfrutar da companhia.

Agradeço à Isabella meu amor, que sempre me apoiou, escutou e teve paciência para suportar minhas ausências em virtude das muitas tarefas do mestrado. Você foi e ainda é uma grande companheira nesta minha caminhada.

Agradeço em especial aos professores que marcaram minha vida como mestres e amigos, Marcos Vinícius da Silva, Marcelo Balbino e André Gustavo dos Santos.

Ao meu orientador Ricardo S. Ferreira pelas horas de explicações, auxílios e cobranças que me mantiveram com foco no objetivo de concluir com sucesso o mestrado.

A todos os colegas, do mestrado e da graduação, que me incentivaram, ajudaram e contribuíram direta ou indiretamente para a realização desse trabalho. Um agradecimento especial ao Odilon, Thiago, Alex. Zé Rui e João.

Aos amigos de república que foram obrigados a escutar todas as minhas piadas ruins ao longo destes anos de Viçosa, Dudinha, Magrelo, Denis, Fellipe, Tonucci.

Por fim mas não menos importante um muito obrigado a Universidade Federal de Viçosa.

BIOGRAFIA

CRISTÓFERSON GUIMARÃES MAGALHÃES BUENO, filho de Geraldo Magalhães Bueno e Norma Guimarães Magalhães Bueno, brasileiro nascido em 27 de julho de 1981 na cidade de Ipatinga, no estado de Minas Gerais.

No ano de 2002 ingressou no curso de graduação em Sistemas de Informação no Centro Universitário do Leste de Minas Gerais, concluído no ano de 2005.

Em 2008, foi aprovado na seleção do programa de pós-graduação do Departamento de Informática – DPI, onde, em maio de 2008, iniciou o mestrado em Ciência da Computação na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em fevereiro de 2011.

Trabalhou como Analista de Sistemas na Cientec Consultoria e Desenvolvimento de Sistemas de Março de 2006 a Fevereiro de 2009.

Em março de 2009 ingressou no programa REUNI pela Universidade Federal de Viçosa, UFV participando de atividades de docência até janeiro de 2010 quando a bolsa se encerrou.

Em fevereiro de 2010 mudou para Belo Horizonte onde ingressou na empresa 3bits trabalhando como Desenvolvedor Web, permanecendo nesta até junho, quando se transferiu para a Prodabel – PBH. Nesta última assumiu o cargo de Arquiteto de Software no projeto Geocorporativo com o objetivo de projetar a Infraestrutura de Dados Espaciais Coorporativa para a Prefeitura Municipal de Belo Horizonte.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

BUENO, Cristóferson Guimarães Magalhães, M.Sc., Universidade Federal de Viçosa, fevereiro de 2011. **Super-VLIW: uma arquitetura dinamicamente reconfigurável com tolerância a falha.** Orientador: Ricardo dos Santos Ferreira. Co-Orientadores: Carlos de Castro Goulart e Vladimir Oliveira Di Iorio.

Um novo cenário emerge devido às nanotecnologias. Estas permitirão taxas de integração elevadas, nos limites, ou mesmo além da capacidade atual do silício. Contudo, estimativas apontam para um percentual de falha entre 1% a 20%, números que podem comprometer o futuro das nanotecnologias. Este trabalho propõe uma arquitetura reconfigurável nomeada Super-VLIW capaz de tolerar as altas taxas de defeitos estimadas para as futuras tecnologias. A arquitetura consiste em uma unidade reconfigurável fortemente acoplada a um processador MIPS. A unidade reconfigurável por sua vez é composta por uma unidade de tradução binária a uma cache de configuração, um vetor de grão-grosso de unidades funcionais e uma rede de interconexão. A reconfiguração é realizada em tempo de execução, traduzindo o código binário sem a necessidade de recompilar. A rede de interconexão é composta por um arranjo de redes multiestágio. Estas redes provêm um comunicação tolerantea falha entre as unidades funcioanais da unidade reconfigurável e os registradores doprocessador MIPS. Este trabalho propõem um mecanismo dinâmico para alocação das unidades disponíveis garantindo a execução paralela das operações básicas, realizando o posicionamento e roteamento em um único passo, o que permite a interconexão correta das unidades mesmo na presença de um número muito elevado de falhas. Além disso, a arquitetura proposta pode escalonar para as futuras nanotecnologias mesmo sob um taxa de falha de 20%.

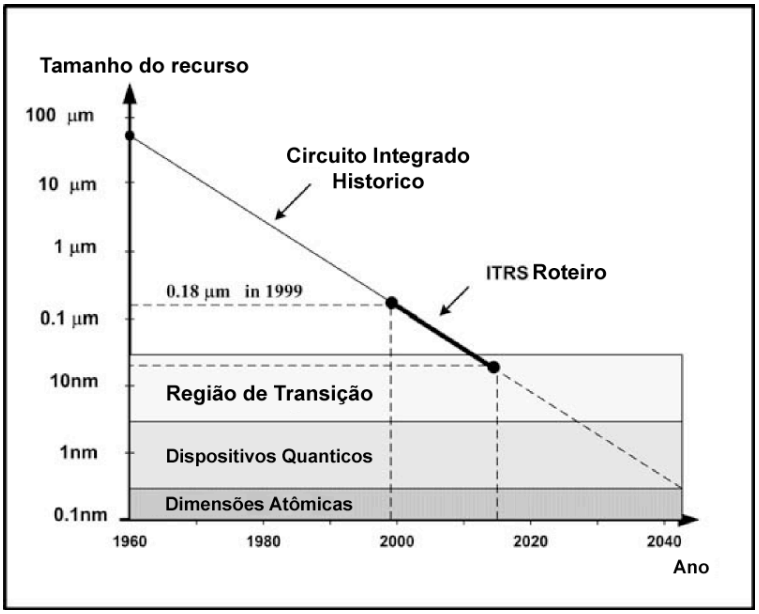
ABSTRACT

BUENO, Cristóferson Guimarães Magalhães, M.Sc., Universidade Federal de Viçosa, February, 2011. **Super-VLIW: a dynamic reconfigurable architecture fault tolerant**. Adviser: Ricardo dos Santos Ferreira. Co-Advisers: Carlos de Castro Goulart and Vladimir Oliveira Di Iorio.

A new scenario emerges due to nanotechnologies that will enable very high integration at the limits or even beyond silicon capacity. However, the fault rate, which is predicted to range from 1% up to 20% of all devices, could compromise the future of nanotechnologies. This work proposes a fault tolerant reconfigurable architecture that tolerates high fault rates expected to future technologies, named Super-VLIW. The architecture consists of a reconfigurable unit tightly coupled to a MIPS processor. The reconfigurable unit is composed of a binary translation unit, a configuration cache, a reconfigurable coarse-grained array of heterogeneous functional units and an interconnection network. Reconfiguration is done at run-time, by translating the binary code, and no recompilation is needed. The interconnection network is based on a set of multistage networks. These networks provide a fault-tolerant communication between any pair of functional unit and from/to the MIPS register file. This work proposes a mechanism to dynamically allocate the available units to ensure parallel execution of basic operations, performing the placement and routing on a single step, which allows the correct interconnection of units even at huge fault rates. Moreover, the proposed architecture could scale to the future nanotechnologies even under a 20% fault rate.

1 INTRODUÇÃO

As nanotecnologias avançaram ao longo das últimas décadas resultando em um processo de redução gradual da tecnologia de semicondutores, a Figura 1.1 apresenta uma projeção da escala da tecnologia para os próximos anos. Este dimensionamento permitiu grandes avanços na produção de circuitos integrados, possibilitando a inserção de um maior número de dispositivos na pastilha de silício, com considerável ganho de desempenho.



Fonte: Adaptado de (STOJCEV *et al.*, 2008)

Figura 1.1. Histórico e tendências da dimensão das tecnologias ao longo do tempo

A despeito dos benefícios obtidos, o atual dimensionamento desta tecnologia gera problemas de confiabilidade, com aumento da densidade de defeitos inseridos durante a fabricação. Um circuito integrado é composto por fios e áreas de contatos. A redução do diâmetro do fio torna este mais frágil e suscetível a quebras, como consequência as áreas de contato também precisam sofrer redução tornando o processo de impressão do circuito ainda mais complexo.

Em outras palavras, a taxa de falhas geradas no processo de fabricação em escala nanométrica é elevada, segundo (DEHON *et al.*, 2005) ela pode atingir até

20% dos fios e conexões, enquanto que nas atuais tecnologias não superam a taxa de 0.1%.

Deste novo cenário surgiu uma busca por novos modelos de arquiteturas que se adaptem melhor a estas novas tecnologias de fabricação de semicondutores.

Exemplos da literatura (GOLDSTEIN *et al*., 2000; HARTENSTEIN, 2001; MEI *et al.*, 2003; COMPTON *et al.,* 2008; TANIGAWA *et al.*, 2008) comprovam que arquiteturas compostas por Arranjos Reconfiguráveis de Grão-Grosso (*Coarse-Grained Reconfigurable Arrays* ou CGRA) atendem satisfatoriamente aos requisitos de desempenho e consumo energético dos novas tecnologias. Entretanto, apenas uma pequena parcela destes estudos ponderou a relação destes requisitos com a presença de falhas.

Este trabalho vem propor uma arquitetura que esteja em conformidade com os requisitos de consumo energético e desempenho do mercado e ainda leve em consideração este requisito emergente que é tolerância a falhas. Também temos como objetivo garantir que a arquitetura não rompa com os paradigmas da indústria de *software* exigindo que ferramentas externas sejam necessárias para modificar os binários a fim de tornar as aplicações aptas a serem executadas nas arquitetura reconfiguráveis. Mecanismos de tradução binária são uma das ferramentas que podem ser utilizadas para garantir a compatibilidade de *software,*respeitando assim o padrão da indústria.Um mecanismo de tradução binária será apresentado em detalhe nos próximos capítulos.

1.1 Motivação

Este trabalho busca apresentar soluções para dois desafios inerentes a arquiteturas reconfiguráveis;prover tolerância a falhas e o segundo garantir a compatibilidade de *software*.

Trabalhos recentes (BECK, 2008), (LAURE, 2010) e (FERREIRA, 2010) apresentaram soluções que podem ser empregadas para compor uma arquitetura dinamicamente reconfigurável que contemple os requisitos supracitados.

1.2 Objetivos

O objetivo principal é propor uma arquitetura dinamicamente reconfigurável tolerante a falha, que faça uso do mecanismo de tradução binária proposta por (BECK *et al*, 2008) garantindo assim a compatibilidade de *software* e estendê-lo para

que esteja apto a funcionar mesmo na presença de falhas. Serão analisados os impactos em termos de desempenho e escalabilidade. O foco principal será nas estruturas de interconexão entre as unidades de processamento e no impacto da presença de falhas no desempenho.

Especificamente pretende-se:

1. Modificar o algoritmo de roteamento proposto por (FERREIRA *et al.,* 2009) para lidar com unidades de *mux* defeituosas em rede de interconexão multiestágio e analisar o comportamento destas na presença de falhas;

2. Modificar o código do simulador desenvolvido por (LAURE, 2010) com base no proposto em (BECK *et al.*, 2005) para adicionar o modelo de rede com suporte ao algoritmo modificado; citado anteriormente. Modificar o simulador para adicionar suporte a falhas no tradutor binário e unidades funcionais. Gerar e injetar falhas nos componentes da simulação: unidades funcionais e rede de interconexão.

3. Avaliar o desempenho da tradução binária descrita em (BECK *et al.*, 2005) na existência de falhas nas unidades funcionais e na rede;

Com base nas análises estatísticas das simulações com falhas, propor um modelo de arranjo arquitetural que se mostre eficiente.

1.3 Organização deste documento

Este texto se encontra organizado na forma de capítulos. Neste primeiro capítulo foi apresentado uma introdução para contextualizar o trabalho, além da motivação e os objetivos que pretendemos alcançar. Nos capítulos seguintes iremos discutir em mais detalhes as tecnologias bases, referências, nossa proposta e os resultados mesurados a partir da mesma.

No Capítulo 2 apresenta o referencial teórico. Nele abordamos os principais conceitos relacionados a este trabalho, discutimos e criticamos algumas arquitetura reconfiguráveis de grão-grosso que foram propostas ao longo dos anos. Também

apresentamos uma visão das redes multiestágio e sua estrutura redundante para tolerar falhas com custo $O(N \log N)$

O capítulo 3 descreve a arquitetura base dessa dissertação que foi proposta por (BECK,2008) e suas derivações. Um dos principais aspectos diz respeito ao mecanismo de tradução binária esua principal vantagem que é manter a compatibilidade do código binário sem a necessidade de recompilação.

O capítulo 4 apresenta modificações realizadas na arquitetura base e suas variações (LAURE, 2010) (PEREIRA *et al.*, 2009). Estas modificações dão origem a uma nova arquitetura denominada por Super-VLIW, proposta neste trabalho. Além disso, mostramos através de exemplos com trecho de código como a arquitetura se comporta quando existem falhas e como ela tira proveito das características intrínsecas da rede Omega e do algoritmo de roteamento implementado em *Hardware* (FERREIRA *et al.*, 2009). Este último foi modificado neste trabalho para incluir tolerância a falhas.

No capitulo 5 apresentamos a metodologia utilizada na realização dos testes, as análises realizadas sobre as redes de interconexão, aplicações de teste, bem como os resultados obtidos.

Os resultados demonstraram que é possível criar uma arquitetura que seja capaz de escalar elevados índices de falhas e explorar o paralelismo das aplicações garantindo uma execução com desempenho satisfatório.

Por fim, o Capítulo 6 apresenta as conclusões obtidas com base neste trabalho e sugere trabalhos futuros relacionados ao que foi proposto neste documento.

2 REFERENCIAL TEÓRICO

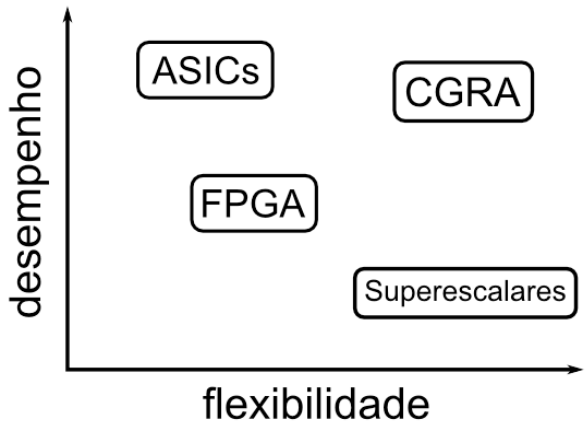
Neste capítulo iremos rever o estado da arte das arquiteturas reconfiguráveis de grão grosso com foco em arquiteturas dinâmicas. As arquiteturas fazem uso de estruturas paralelas de interconexão para possibilitar uma comunicação eficiente entre suas unidades de execução. Também serão revistas neste capítulo as estruturas de interconexão, em especial as redes de interconexão multiestágio que podem ser usadas em aplicações com tolerância a falhas.

2.1 Arquiteturas Reconfiguráveis

A computação reconfigurável surge como alternativa para atingir parcialmente a desempenho dos circuitos de aplicação específica (*Application Specific Integrated Circuits* ou ASIC) e a flexibilidade dos processadores de propósito geral. Dedicando parte da área do chip a estes dispositivos reconfiguráveis algumas aplicações ou parte delas podem executar diretamente em hardware especializado. O surgimento do Arranjo de Portas Programável em Campo (*Field Programmable Gate Array* ou FPGA) em meados dos anos 80 abriu caminho para a implementação da computação reconfigurável.

Os FPGAs são arranjos de elementos programáveis cuja funcionalidade dos elementos e suas interligações são determinados por um conjunto de bits de configuração. Arquiteturas reconfiguráveis e FPGAs foram muito pesquisadas nas duas últimas décadas (COMPTON*et al*., 2002; NAGELDINGER, 2001; HARTENSTEIN, 2001). Foram apresentados diversos modelos para caracterizar arquiteturas reconfiguráveis, contudo não existe uma classificação formal para elas, sendo assim, para este trabalho utilizaremos a mesma classificação empregada em (BECK, 2008) e (LAURE, 2010).

As arquiteturas reconfiguráveis podem ser classificadas quanto ao grau de acoplamento entre processador de propósito geral e o componente reconfigurável, granularidade do bloco, topologia de interconexão empregada e modelo de reconfiguração.



Fonte: Adaptado de (HARTENSTEIN, 2001)

Figura 2.1. Relação entre desempenho e flexibilidade dos modelos de computação

A Figura 2.1 apresenta uma representação conceitual de como os modelos de computação se posicionam quando observados os requisitos de desempenho e flexibilidade. Diferentemente das arquiteturas tradicionais (ASICs e Superescalares), os sistemas reconfiguráveis são flexíveis, podem ser reconfigurados estaticamente ou dinamicamente após a fabricação, e podem ser personalizados para atender a demanda de alto desempenho de uma aplicação especifica. Outro ponto de vantagem das arquiteturas reconfiguráveis é a sua regularidade.

Normalmente as arquiteturas reconfiguráveis são aplicadas ao domínio de aplicações com intensa carga de trabalho e manipulações de dados, atuando na aceleração de alguns trechos de código que se repetem muito ao longo da execução.

Contudo a aceleração destes trechos envolve ainda algum tipo de transformação no código binário da aplicação original, através do uso de ferramentas especificas ou novas linguagens.

2.1.1 FPGA e CGRA

Os FPGAs tem uma granularidade de reconfiguração muito pequena, cada célula implementa um função com 4 a 6 entradas e uma ou duas saídas, ou seja, uma função Booleana simples. A grande flexibilidade do FPGA aumenta a complexidade do mapeamento das aplicações para arquitetura. Uma alternativa é aumentar a granularidade das células base para unidades lógica aritméticas usadas em processadores que implementam operações em nível de palavra de 16 ou 32 bits (soma, subtração, multiplicação, lógica bit a bit, etc...) As arquiteturas reconfiguráveis de grão grosso (*Coarse Grain Reconfigurable Architecture* ou

CGRA), como são denominadas estas arquiteturas em nível de palavra, vem se destacando nos últimos anos e vários modelos de arquiteturas já foram propostas (MEI *et al.*, 2003). Como podemos observar na Figura 2.1, os CGRA podem aliar o desempenho dos ASICs a flexibilidade dos processadores de finalidade geral.

Arquiteturas Reconfiguráveis de Grão-Grosso ou CGRA são uma das arquiteturas reconfiguráveis mais promissoras. Os CGRA estão atraindo o interesse das áreas de sistemas embarcados e aplicações multimídia, que demandam plataformas flexíveis, mas muito eficientes (HARTENSTEIN, 2001).

2.1.2 Configuração

Em um sistema reconfigurável, o tempo total de execução de uma operação realizada corresponde à soma dos tempos de configuração e do tempo de execução. A forma como a configuração é manipulada depende das características e necessidades do sistema implementado, sendo que a configuração pode ser carregada apenas no início de execução ou em períodos determinados para alterar o funcionamento do sistema.

A computação reconfigurável pode ser distinta em dois modos de configuração: estático e dinâmico (SANCHEZ, 1999). A reconfiguração estática pressupõe o funcionamento permanente da arquitetura reconfigurável após esta ter sido configurada. Este modo não concede grande flexibilidade, mas permite atingir um bom desempenho através do hardware especializado para uma determinada aplicação.

Contudo algumas aplicações necessitam de uma maior flexibilidade da configuração e para tal, diversas configurações são utilizadas ao longo da execução da aplicação. Isto possibilita que mais segmentos da aplicação sejam mapeados no hardware especializado. Deste modo, a maior parte da aplicação pode ser acelerada num sistema reconfigurável resultando em um desempenho mais elevado. Este modo de operação é conhecido como reconfiguração dinâmica.

As configurações estáticas ou dinâmicas podem ser geradas por um compilador de maneira estática ou durante a execução. O fato de uma arquitetura ser dinamicamente reconfigurável não significa que sua configuração será gerada durante a execução.

Existe ainda um outro modo de reconfiguração conhecido como parcial, esta modalidade é um caso especial da reconfiguração dinâmica, pois a modificação é

aplicada também em tempo de execução, mas de forma seletiva, há apenas uma porção do sistema reconfigurável. Esta flexibilidade permite que o hardware seja mais personalizado às necessidades correntes da aplicação. Além disso, este modo apresenta uma sobrecarga de configuração menor do que no caso anterior, pois apenas os recursos selecionados precisam ser reprogramados.

2.2 Arquiteturas Reconfiguráveis

Como mencionado anteriormente, os CGRA são menos flexíveis que os FPGAs, porém as aplicações podem ser rapidamente mapeadas nestas arquiteturas reconfiguráveis. Nesta seção serão apresentadas algumas destas arquiteturas para dar uma visão geral sobre os desenvolvimentos na área de computação reconfigurável de grão-grosso. Uma linha do tempo com as arquiteturas que serão apresentadas está relacionada na Tabela 2, junto com algumas propriedades de cada arquitetura, como: estrutura de interconexão, largura da via de dados e modelo de reconfiguração. Iremos descrever algumas das principais arquiteturas CGRA, para uma revisão detalhada sugerimos consultar (HARTENSTEIN,2001).

Tabela 2.1: Arquiteturas reconfiguráveis de grão-grosso organizadas cronologicamente

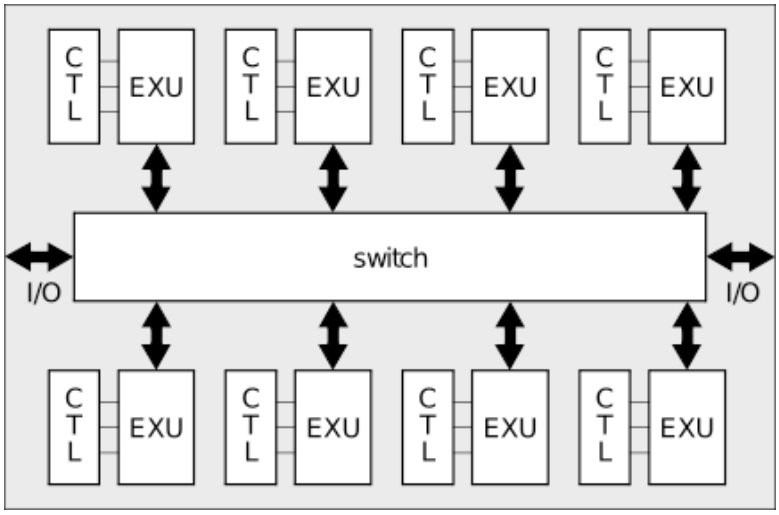
Publicação	Nome do Sistema	Estrutura de Interconexão	Modelo de Reconfiguração	Largura da Via de Dados
1990	PADDI	*Crossbar*	Estática	16 bits
1993	PADDI-2	*Crossbar*	Estática	16 bits
1994	DP-FPGA	*Mesh* (2-D)	Estática	4 bits
1995	KressArray	*Mesh* (2-D)	Estática	32 bits
1996	RaPiD	Arranjo Linear (1-D)	Estática	16 bits
1996	Matrix	*Mesh* (2-D)	Dinâmica	16 bits
1997	Garp	*Mesh* (2-D)	Estática	2 bits
1997	RAW	*Mesh* (2-D)	Estática	32 bits
1998	PipeRench	Arranjo Linear (1-D)	Dinâmica	4 bits
1998	MorphoSys	*Mesh* (2-D)	Dinâmica	16 bits

Fonte: Adaptada de (HARTENSTEIN, 2001)

2.2.1 PADDI

A arquitetura PADDI (*Programmable Arithmetic Device for Digital Signal Processing*) foi inicialmente descrita em (CHEN *et al.,* 1990). Esta arquitetura era

composta por um cluster de oito Unidades Aritméticas de Execução (EXUs), que eram conectadas através de uma rede *crossbar*. A Figura 2.2 apresenta uma visão geral da arquitetura.



Fonte: (NAGELDINGER, 2001)

Figura 2.2. Visão geral da arquitetura PADDI

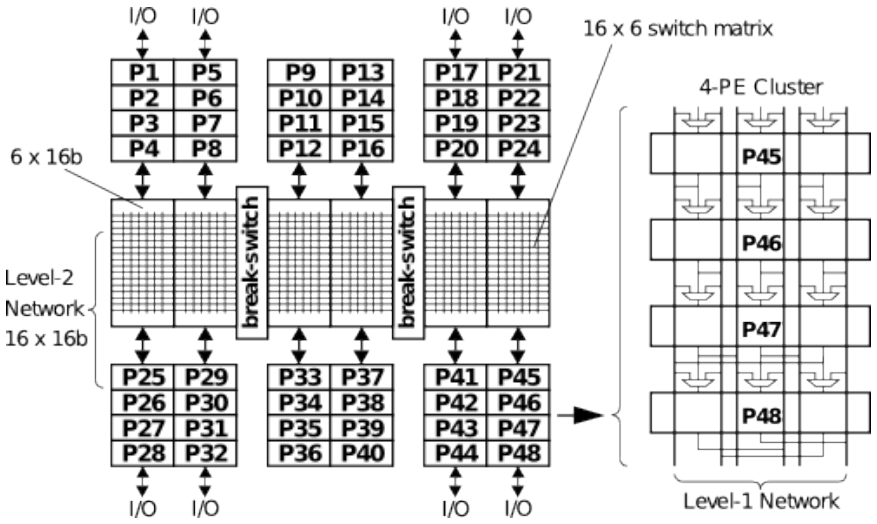
Cada EXU possui uma Unidade Lógica Aritmética (ALU) de 16 bits, que suporta operações de adição, subtração, comparação, dentre outras. Outras operações podem ser realizadas concatenando-se duas EXUs.

Esta arquitetura emprega a rede *crossbar* para rotear sinais de controle e dados. A cada ciclo os dados podem ser modificados dinamicamente em tempo de execução, contudo os sinais de controle são estáticos e precisam ser determinados em tempo de compilação.

O custo elevado da rede *crossbar* em relação ao restante da arquitetura não permite a este modelo de arquitetura ser escalável.

2.2.2 PADDI-2

Em 1993 foi proposta na Universidade de Berkley um novo modelo para suceder o PADDI, este modelo foi denominado PADDI-2 (YEUNG *et al.*, 1993). Esta arquitetura apresenta um mecanismo de execução orientado a dados. Embora os princípios básicos do PADDI tenham sidos preservados, a PADDI-2 tem várias diferenças. A Figura 2.3 apresenta a estrutura da arquitetura PADDI-2.



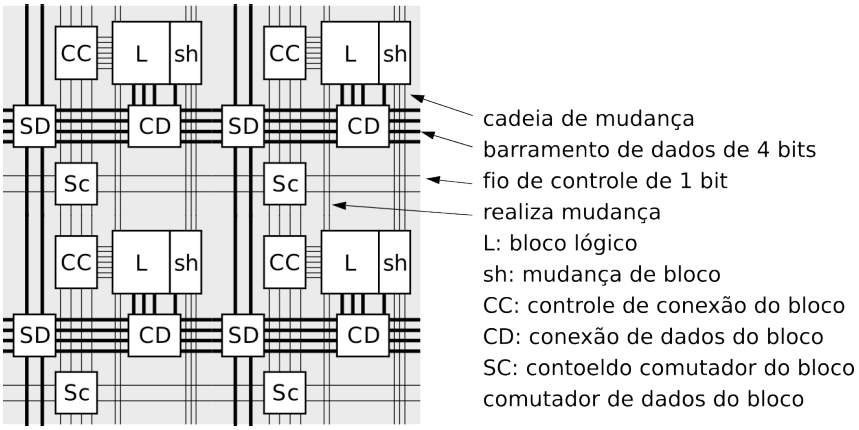
Fonte: (NAGELDINGER, 2001)

Figura 2.3. Modelo da arquitetura PADDI-2

A PADDI-2 possui 48 elementos de processamento de 16 bits cada, sendo cada uma delas capaz de processar 12 instruções distintas. Enquanto na PADDI todos os elementos de processamento eram interconectados através de uma rede *crossbar*, O PADDI-2 com 48 elementos, usa uma estrutura hierárquica de conexão, formando um cluster através de um arranjo linear de elementos de processamento.

2.2.3 DP-FPGA

A arquitetura DP-FPGA ou *Datapath* FPGA foi proposta em 1994 em (CHEREPACHA *et al*., 1996). Esta arquitetura foi uma tentativa de implementar vias de dados de forma regular semelhantes aos utilizados em processamento de sinal digital, comunicações, emulação de circuitos e processadores de propósito específico. O sistema compreende três componentes básicos: lógica de controle, via de dados e memória.



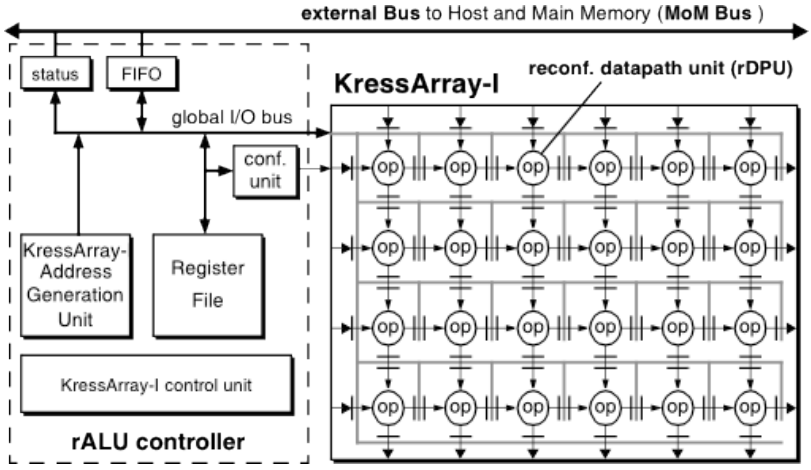
Fonte: (NAGELDINGER, 2001)

Figura 2.4. Modelo da arquitetura DP-FPGA

Na abordagem proposta nesta arquitetura fica evidente sua relação com o FPGA, ALU *bit-sliced* e uma arquitetura de roteamento bastante similar à empregada nas arquiteturas de grão-fino.

2.2.4 KressArray-I

A KressArray-I (KRESS, 1996) é uma arquitetura regular em malha simplificando o custo das conexões com uma abordagem de conexão local.A Figura 2.5 ilustra a via de dados reconfigurável utilizada pela arquitetura KressArray-I.



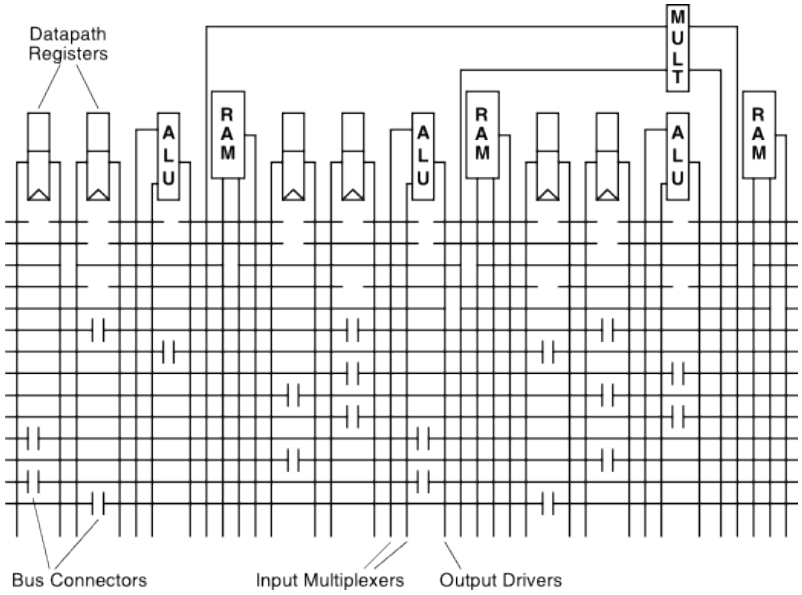
Fonte: (NAGELDINGER, 2001)

Figura 2.5. Visão geral da arquitetura KressArray-I

A arquitetura de roteamento do KressArray-I explora a conexão com o vizinho mais próximo (vizinho leste e sul) de cada rDPU ou unidade reconfigurável de via de dados (*reconfigurable DataPath Units*). Como recurso adicional para conexões longas a arquitetura faz uso do barramento global, este também é utilizado para dados de I/O e transferência dos resultados dos imediatos entre o arranjo e o registrador do controlador. Este barramento global é controlado de forma estática, sendo este controle determinado em tempo de compilação.

2.2.5 RaPiD

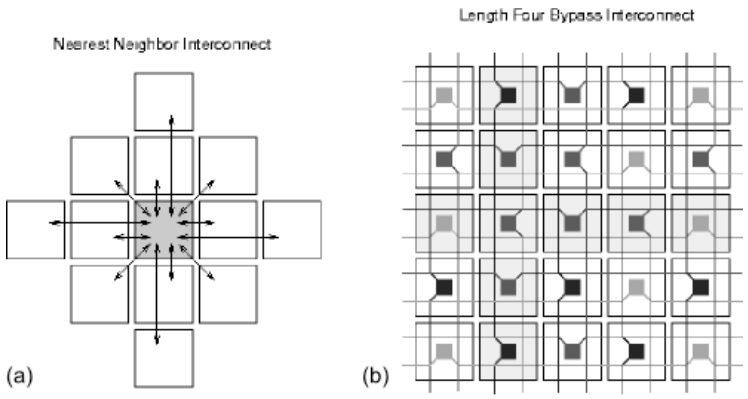
O RaPiD propõem uma arquitetura unidimensional ligada por barramentos segmentados. A estrutura da rede de conexão pode ser customizada para um conjunto de aplicações. A Figura 2.6 mostra um diagrama da arquitetura do RaPID (EBELING *et al.*, 1996).



Fonte: (NAGELDINGER, 2001)

Figura 2.6. Modelo da arquitetura RaPiD

2.2.6 Matrix



Fonte: (NAGELDINGER, 2001)

Figura 2.7. Arquitetura Matrix. (a) Conexão com o vizinho mais próximo com distância dois; (b) Interconexão com comprimento quatro

A arquitetura do MATRIX (MIRSKY*et al.*, 1996) ilustrada na Figura 2.7 é bidimensional, onde cada célula pode se comunicar com as oito células no entorno e ainda existe a possibilidade de comunicação com um salto (*1*-hop) nas direções norte, sul, leste e oeste.

2.2.7 Garp

O conceito da arquitetura Garp foi originalmente publicado em 1997 por Hauser e Wawrzynek (HAUSER *et al.*, 1997) e consiste em um processador MIPS-II associado a uma arquitetura reconfigurável, sendo esta última um mecanismo para

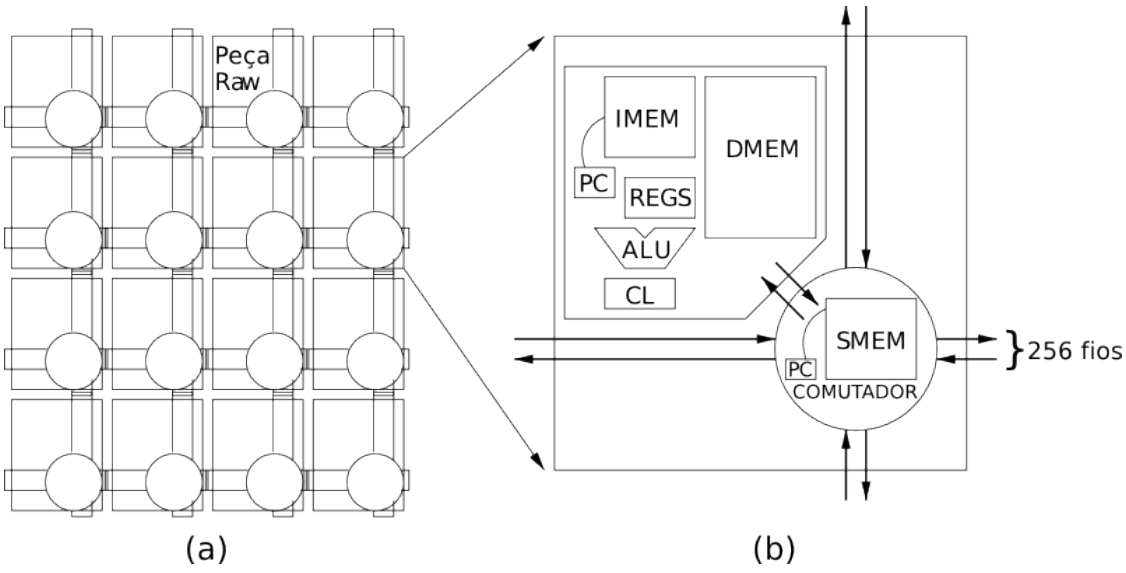
promover aceleração. A proposta do Garp é acelerar os loops das aplicações, reconfigurando o arranjo reconfigurável apenas nas entradas e saídas de cada loop. Sendo assim o arranjo Garp executa de forma independente todo o loop.

Um dos aspectos mais importantes na arquitetura Garp é que ela recebe como entrada código descrito em ANSI C padrão, sem a necessidade de inserção de nenhum indicador ou diretiva específica no código fonte. O compilador Garp (CALLAHAN *et al.*, 2000) identifica as seções de código que podem ser aceleradas através do arranjo Garp.

O uso de uma granularidade fina nesta arquitetura elevou os custos de reconfiguração, mapeamento e síntese.

2.2.8 RAW

A arquitetura RAW ou *Reconfigurable Architecture Workstation* foi concebida em 1997 pelo grupo de arquitetura do MIT e sua primeira publicação foi (WAINGOLD *et al.*, 1997). A ideia do RAW é proporcionar uma arquitetura simples para computação altamente paralela composta por várias peças repetidas que se interconectam através do vizinho mais próximo. Cada peça é composta por um processador RISC e todo o processador RAW é composto por um conjunto de 16 peças, sendo que cada uma delas possui seu banco de memória. Esta estrutura está ilustrada na Figura 2.8.



Fonte: (NAGELDINGER, 2001)

Figura 2.8. RAW: (a) Arquitetura Reconfigurável (b) Microprocessador

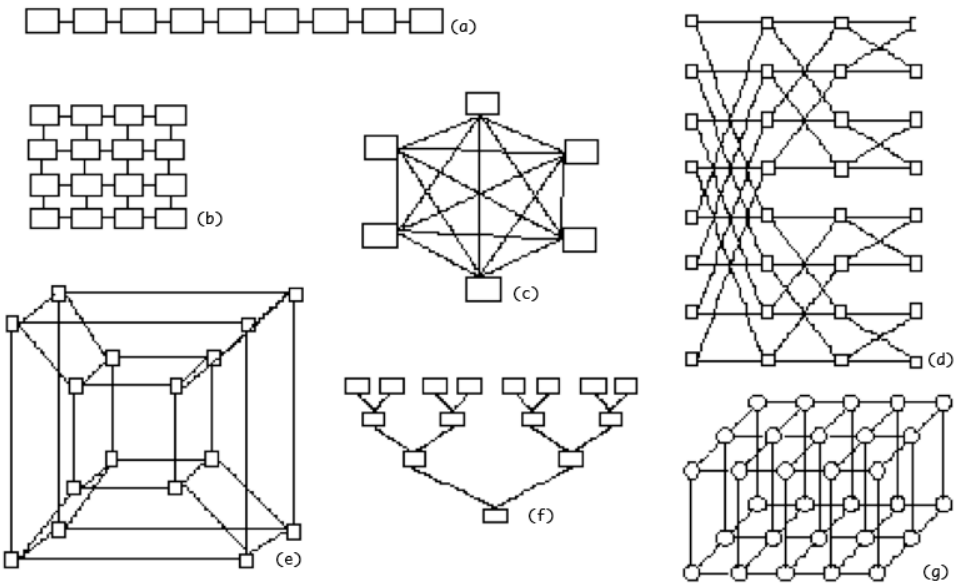
A RAW trabalha com duas estruturas de rede uma estática e outra dinâmica. A rede estática é interligada ao processador de cada peça e controla a transferência de dados e esta é determinada por um escalonamento estático em tempo de compilação. Para os casos onde o escalonamento estático não é possível, uma rede dinâmica capaz de realizar a transferência dos dados é fornecida. Esta rede dinâmica utiliza o algoritmo de roteamento *wormhole* para o encaminhamento dos dados e o roteamento utiliza um cabeçalho que é informado antes do dado ser transferido.

Apesar de promissoras todas as arquiteturas reconfiguráveis propostas anteriormente não se popularizaram. Um dos principais fatores é o custo de produção dos circuitos que precisam ser vendidos em larga escala para serem viáveis e a necessidade de recompilar os códigos, pois todas as abordagens eram dependentes de compilador. Poucas arquiteturas reconfiguráveis geram sua configuração dinamicamente, sem depender de compilação. Os trabalhos de (GSCHWIND*et al*., 2000b)e (BECK *et al.*, 2005) foram pioneiros. Neste trabalho iremos nos basear na arquitetura proposta em (BECK *et al.*, 2008) baseada na abordagem de tradução binária. Este tópico será apresentado no próximo capítulo.

Outro ponto importante é o custo da estrutura de interconexão que é um componente fundamental nas arquiteturas reconfiguráveis e será abordado na próxima seção.

2.3 Rede de Interconexão

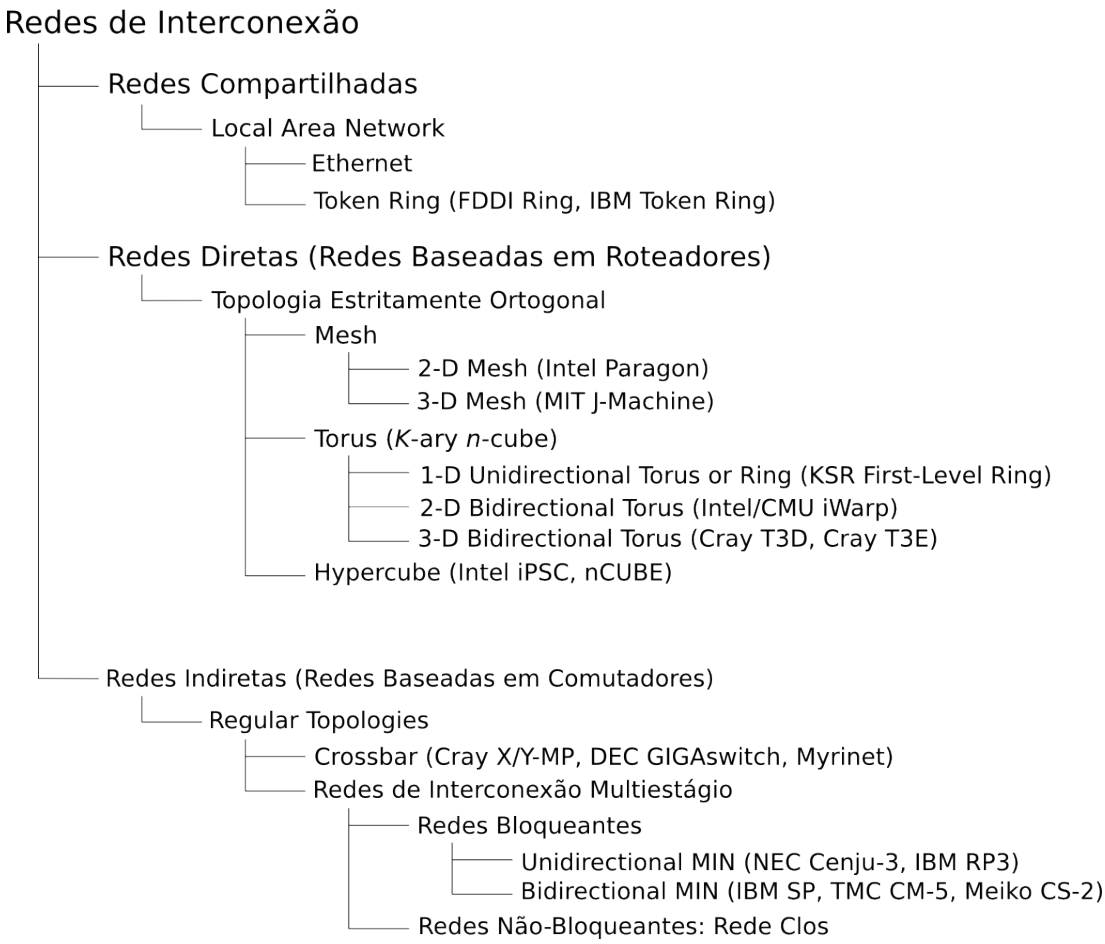
A eficiência e o custo de uma arquitetura paralela está diretamente relacionada àestrutura usada para conectar suas unidades. As características e o custo das redes de interconexão dependem da aplicação e da tecnologia. Até o momento não existe uma solução geral que atenda satisfatoriamente a estes dois principais requisitos para todas as arquiteturas. A Figura 2.9 apresenta algumas topologias de redes de interconexão mais populares e estudadas ao longo das duas últimas décadas na construção de redes de computadores e arquiteturas paralelas. Alguns dos pontos principais são: distância máxima entre dois pontos da rede, custo do comutador, número de interconexões em função do número de nós da rede, algoritmo de roteamento, regularidade para construção física da rede, dentre outros.



Fonte: (METAXAS, 2006)

Figura 2.9. Alguns Modelos de Redes de Interconexão: (a) arranjo linear, (b) arranjo 2D, (c) completa, (d) butterfly, (e) hypercube, (f) árvore, (g) arranjo 3D

Um esquema com a classificação das redes de interconexão é apresentado na Figura 2.10, onde as redes de interconexão mais conhecidas estão categorizadas em quatro classes principais, com base em suas respectivas topologias: redes compartilhadas, redes diretas, redes indiretas e redes híbridas. Para cada classe a Figura 2.10apresenta de forma hierárquica as subclasses e também indica algumas implementações reais destas redes.



Fonte: Adaptado de (DUATO et al., 2003)

Figura 2.10. Classificação das redes de interconexões

Nesta dissertação vamos focar nas redes da classe indireta, também denominadas redes dinâmicas. Estas redes são conhecidas como redes de interconexão multiestágio (*Multistage Interconnection Networks* ou MINs). A seção a seguir apresenta em detalhes esta categoria de rede.

2.4 Redes Multiestágio

Redes de Interconexão Multiestágio conectam dispositivos de entrada a dispositivos de saída através de uma série de estágios. O número de estágios e o padrão de conexão entre eles determina a capacidade de conexão da rede.

As MINs foram amplamente estudadas nos últimos 50 anos (WU *et al.*, 1980; FENG *et al*., 1981; ADAMS *et al.,* 1987; SIEGEL *et al.*, 1989; KAMIURA *et al.*, 2000; DUATO *et al.*, 2003; ANSAR *et al.*, 2009). Os primeiros trabalhos (CLOS, 1953) e (BENES, 1965) apresentavam propostas para redes de telefonia, posteriormente foram utilizadas para computação paralela (WU *et al*., 1980) nas décadas de 70 e 80, redes ATM na década de 90, *Network on Chip* (NoC) e redes

óticas (ABED *et al.*, 2008; SHEN *et al.*, 2001) e são utilizadas até os dias de hoje em supercomputadores (TIAN, 2006). O custo e o atraso da MIN são os principais fatores que motivaram seu uso. Enquanto uma rede Crossbar tem custo $O(N^2)$, a MIN tem custo $O(N \log N)$ e atraso $O(\log N)$ para N entradas/saídas.

Na década de 70 e 80 as redes multiestágio foram utilizadas para o processamento paralelo em várias arquiteturas, mas devido à dificuldade de construção dos circuitos, boa parte dos trabalhos foi de cunho teórico ou apenas protótipos de máquinas que não se popularizaram (WU *et al.*, 1980), (DUATO *et al.*, 2003).

Nas redes multiestágio cada estágio é composto por um conjunto de comutadores e um padrão de comunicação entre os estágios. Os comutadores do estágio i são conectados por um padrão ao estágio $i + 1$. A Figura 2.11 ilustra a estrutura geral de uma rede multiestágio. A capacidade de roteamento do comutador pode variar do mais simples 2x2, até um comutador NxN formando uma rede *Crossbar* de um único estágio.

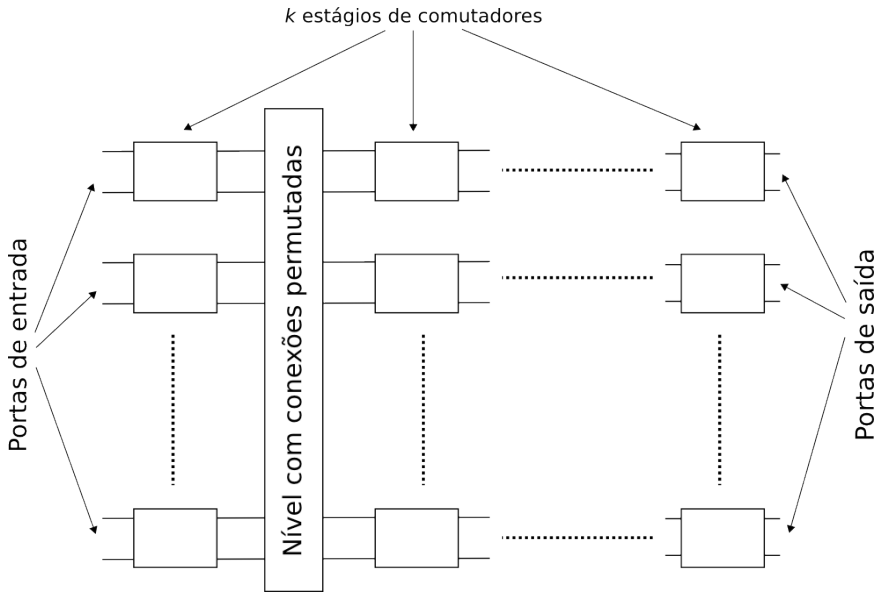


Figura 2.11 Estrutura geral de uma rede multiestágio com tamanho *N* e *k* estágios

As redes multiestágio podem ser classificadas em três categorias funcionais, sendo elas:

a) Bloqueantes: A conexão entre uma entrada e uma saída livre nem sempre é possível, pois podem existir conflitos, com conexões previamente estabelecidas.

Normalmente, existe apenas um caminho entre cada par de entrada/saída minimizando assim o número de estágios para Log N estágios para N entradas/saídas. No entanto, também é possível prover múltiplos caminhos a fim de reduzir os conflitos e aumentar a tolerância a falhas. Estas redes são denominadas caminhos múltiplos (*multipath)*.

b) Não-Bloqueantes: Qualquer entrada pode ser conectada a qualquer saída livre sem afetar ou sofrer interferência das conexões existentes. A ordem das conexões não interfere na capacidade de conexão. Para realizar todas as conexões elas exigem múltiplos caminhos entre cada entrada e saída, o que na prática gera estágios extras, linhas extras ou comutadores mais complexos, elevando seu custo.

c) Rearranjáveis: Qualquer entrada pode ser conectada a qualquer saída livre, Contudo a existência de conexões exigirá que os caminhos sejam rearranjados a fim de acomodar todas as conexões. As rearranjáveis constituem uma solução intermediária entre as redes bloqueantes e as não bloqueantes. Assim como as não-bloqueantes, estas redes também necessitam de caminhos extras entre as entradas e saídas, mas o número de caminhos e o custo é inferior, e o algoritmo de roteamento resolve os conflitos rearranjando as ligações.

A Tabela 2 relaciona as principais classes de redes MIN quanto suas classificações funcionais.

Table 2.2: Classificação funcional das principais classes de redes MIN

Classificação Funcional	**Redes MINs**
Bloqueantes	Omega Omega Invertida Butterfly Flip Baseline
Não-Bloqueantes	Clos
Rearranjáveis	Benes

Para este trabalho focaremos na rede multiestágio Omega. Esta rede é uma das várias redes de conexão utilizadas em máquinas paralelas devido a sua simplicidade e baixo custo. Elas pertencem à classe de redes Delta (PATEL, 1981) e é um dos modelos de redes bloqueantes mais utilizados. Para evitar conflitos e prover tolerância a falhas, iremos fazer uso de estágios extras, ou seja, redes bloqueantes com múltiplos caminhos. Entretanto, o custo será menor que das redes rearranjáveis

e não bloqueantes. A próxima seção detalhará a rede Omega assim como seu padrão de roteamento.

2.4.1 Rede Omega

A rede Omega foi originalmente proposta por (LAWRIE, 1975) e possui a capacidade de ser auto-roteável.

A Figura 2.13 ilustra uma rede Omega simples, com as seguintes características:

- A rede tem log2N estágios, sendo que cada um possui N/2 comutadores;
- N é o número de entradas e saídas, no exemplo em questão N é igual a 8;
- As entradas estão conectadas ao primeiro estágio através do padrão *Perfect Shuffle* e este padrão se repete entre os estágios;

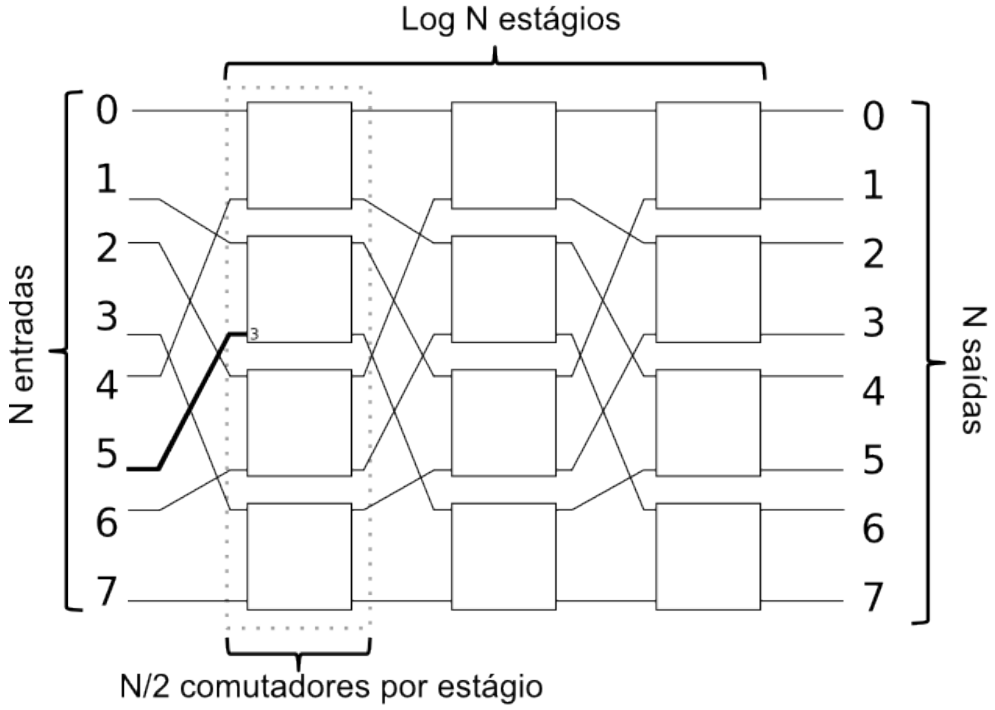


Figura 2.12. Rede Multiestágio Omega 8x8

Para uma rede com $N = 8$, cada linha pode ser representada por um código binário de $\log N = log\ 8 = 3$ bits. O padrão de ligação entre os estágios é o *Perfect Shuffle* que executa uma rotação a esquerda. Neste padrão de interconexão a linha x do estágio i será conectada na linha rotação_a_esquerda(x) do estágio i+1. Um exemplo deste comportamento está destacado na Figura 2.12 onde a entrada 5 liga-se a linha 3 no segundo comutador do primeiro estágio, a notação em binário desta ligação é **1***01 ->01***1**.

Internamente cada comutador pode assumir diferentes estados, cada estado é determinado por seus bits de configuração. A Figura 2.13 representa cada um destes estados, sendo eles:

1. Completos: ligação direta (*straight through*) ou cruzada (*exchange*);
2. *Multicast*: dispersão abaixo (*down broacast*) ou dispersão acima (*upper broadcast*);
3. Não completos: são quatro estados que derivam do completo;
4. Ainda pode existir o estado desligado.

Um conjunto de bits em cada comutador é utilizado para representar os estados, sendo um bit para completos, 2 bits para *broadcast* e de 3 a 4 bits para representar os estados incompletos e desligado.

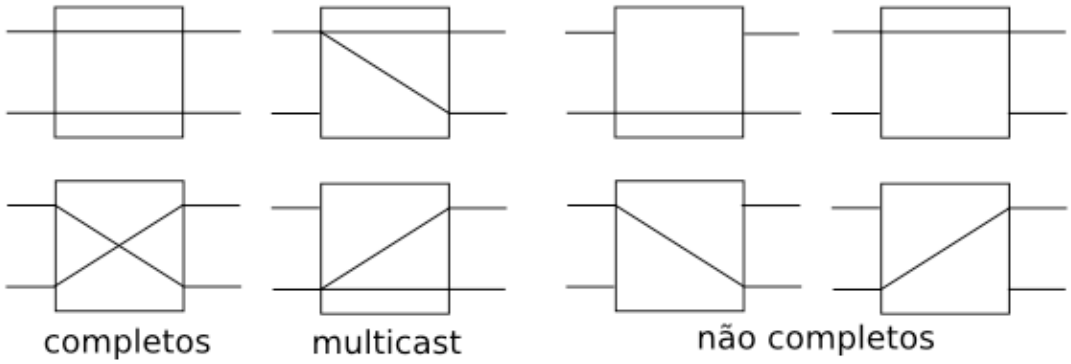


Figura 2.13. Representação dos diferentes estados que cada comutador pode assumir

O padrão de interconexão em combinação com o estado dos comutadores determina o roteamento da rede. A seção a seguir apresentará o roteamento em uma rede multiestágio Omega.

2.4.2 Busca e Alocação Dinâmica na Rede Omega

O roteamento na rede multiestágio Omega pode ser programado de duas maneiras, por pacote ou por circuito. No roteamento por pacote cada comutador verifica os bits de endereço de destino e origem para fazer o encaminhamento de cada pacote, além disso, é necessário existir um mecanismo de fila para tratar os conflitos de roteamento.

Para um algoritmo de roteamento por circuito, primeiro é verificado se é possível adicionar uma nova conexão entre todo o caminho entrada/saída. Em uma rede Omega sem estágios extras, como o exemplo da Figura 2.14 cada par de entrada e saída terá apenas uma única rota, sendo esta determinada através de um “ou

exclusivo" ou XOR dos endereços de origem e destino. O bit mais e menos significativo resultante do XOR correspondem respectivamente ao primeiro e último estágios.

Cada bit resultante da operação XOR corresponderá ao estado de ligação do comutador de um estágio de índice correspondente, sendo 1 para ligação cruzada ou 0 para direta, todos os estados possíveis podem ser vistos na Figura 2.13.

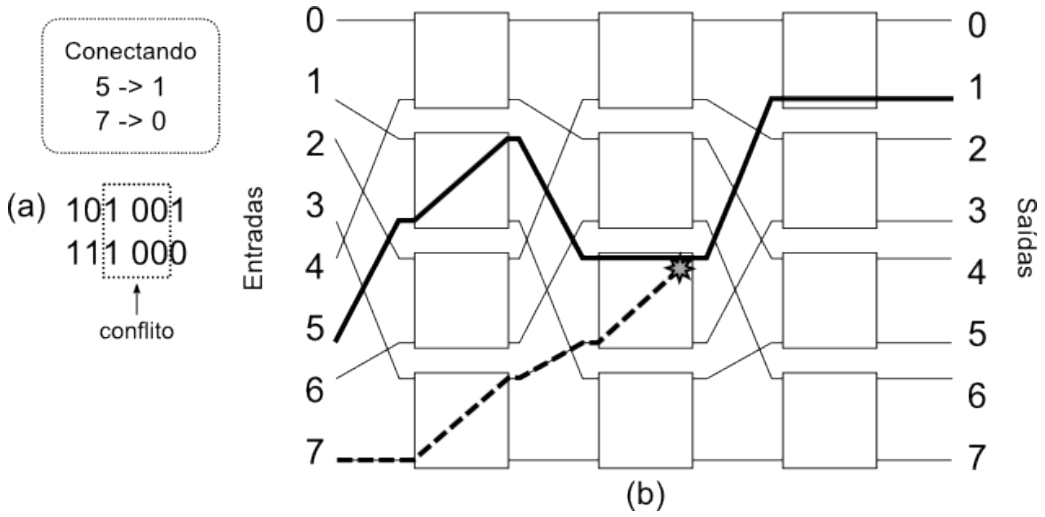


Figura 2.14. Rede Omega 8x8: (a) Representação binária da entrada 5 e saída 1 concatenada; (b) Exemplo de conflito de roteamento na rede Omega entre as conexões 5→1 e 7→0

A Figura 2.14 mostra um exemplo de roteamento entre a entrada 5 e a saída 1, ou 101 e 001 em binário, cujo XOR será 4 ou 100 em binário. O valor resultante do XOR determina que os comutadores estejam nos estados cruzado, direto e direto. Esta figura também apresenta uma situação de conflito, neste caso na tentativa de roteamento entre o par entrada e saída 7 e 0 com a conexão realizada anteriormente 5→1.

Um conflito representa a disputa por um único recurso, quando este já está alocado. A rede Omega possui característica bloqueante, portanto sempre existirão permutações de ligações que não poderão ser realizadas. Em redes multiestágio um conflito ocorre quando dois pares de entrada, saída passam pela mesma saída de um comutador em um determinado estágio.

Uma forma de minimizar os conflitos nas redes Omega sem elevar muito seu custo é inserir estágios extras na rede. Para cada K estágios extras, obtêm-se $2K$ alternativas de roteamento. A Figura 2.15 ilustra a rede Omega 8x8 com um estágio extra. Neste cenário a conexão entre 5→1 passa a ser possível através de dois caminhos.

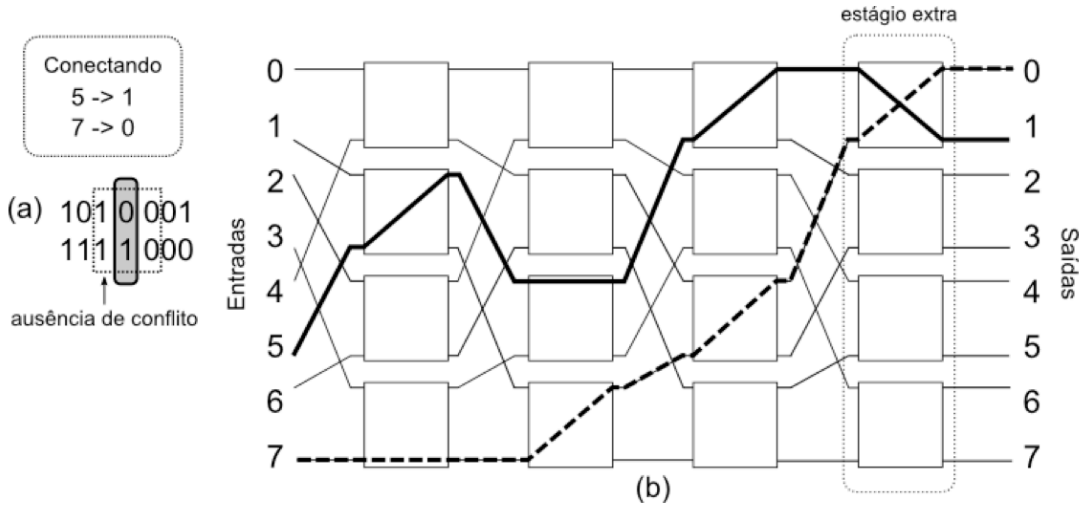
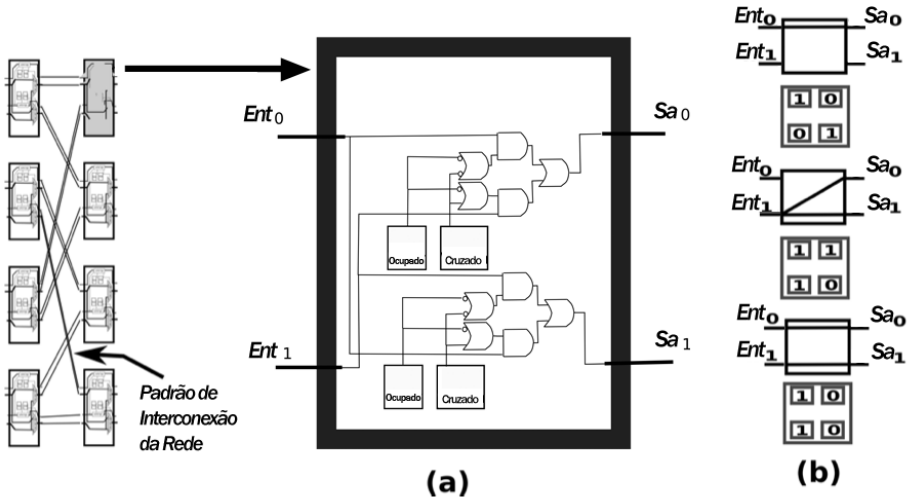


Figura 2.15. (a) Representação binária das entradas e saídas concatenadas. Os bits em destaque são em decorrência do estágio extra. (b) Resolução de conflito com adição de um estágio extra na rede Omega 8x8

É importante ressaltar que a ocorrência de conflitos depende da ordem em que os pares são roteados e do código atribuído ao bit extra.

Em alguns casos de uso, uma origem precisa atingir todos os destinos (*broadcast*) e ou subconjunto (*multicast*) dos possíveis destinos. Este é um requisito comum no cenário das arquiteturas reconfiguráveis, alvo deste trabalho.

Neste trabalho vamos abordar o roteamento em nível de circuito e não de pacote; a técnica de roteamento apresentada aqui foi originalmente proposta em (FERREIRA *et al.*, 2009). Para esta abordagem é necessário verificar dinamicamente a disponibilidade de todo o circuito antes que este seja programado; por este motivo o algoritmo proposto realiza o roteamento aplicando broadcast a fim de determinar quais as saídas estão disponíveis. Esta técnica permite verificar a possibilidade de roteamento em apenas um ciclo, mesmo na presença de estágios extras graças ao codificador de prioridade. A abordagem é baseada em uma lógica de controle por comutador. A lógica de controle é interligada seguindo o padrão da rede. Além disso, ele fez uso de comutadores capazes de executar a operação de *multicast*. A Figura 2.13 ilustra os estados dos comutadores, dentre eles o estado de *multicast*.



Fonte: Adaptado de (FERREIRA *et al.*, 2009)

Figura 2.16. (a) Comutador e lógica de controle. (b) Representação de alguns estados ocupados e cruzados e seus respectivos bits de configuração

No algoritmo proposto em (FERREIRA *et al.*, 2009) cada comutador armazena 4 bits como ilustrado na Figura 2.16 (a), os bits funcionam como uma tabela de unidades ocupadas, as saídas livres são conectadas ao decodificador de prioridade, veja Figura 2.17.

O processo de roteamento deste algoritmo pode ser dividido em 3 etapas.

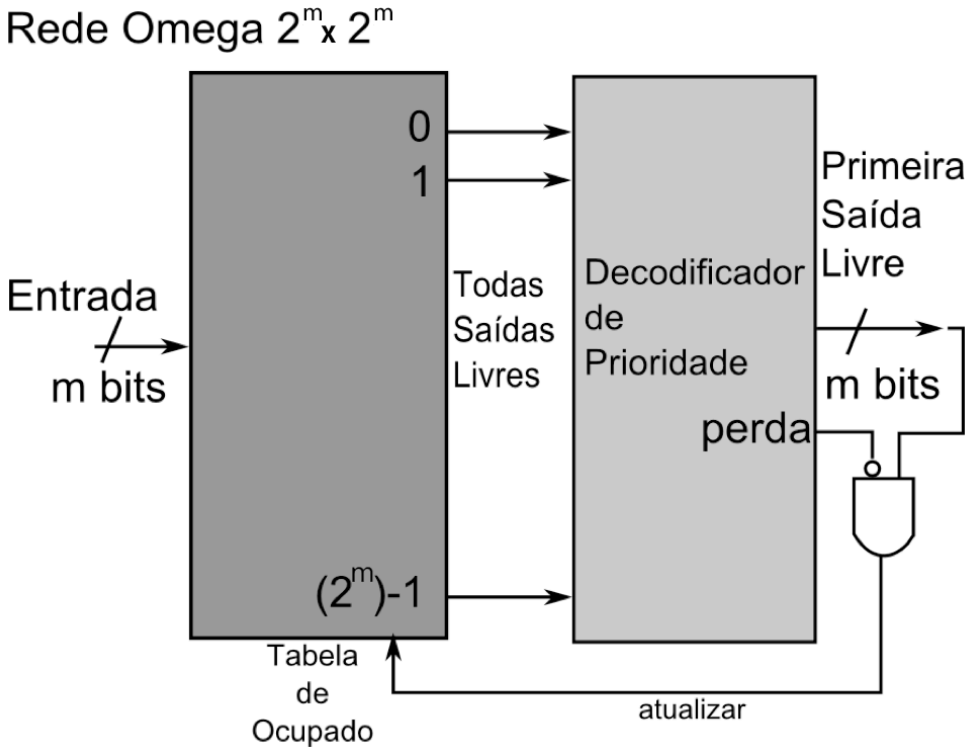


Figura 2.17. Estrutura Geral da Tabela de Reserva

Na 1º etapa um decodificador é conectado às entradas. Dado o endereço origem do roteamento apenas uma entrada correspondente ao endereço será ativada com o bit 1 (um), sendo que as demais entradas receberão o valor 0 (zero).

Na 2º etapa cada comutador propaga o valor de entrada para as saídas livres. Este comportamento é descrito pela equação a seguir:

$$Sa_0 = Ent_0(\overline{O}_0 + \overline{C}_0) + Ent_1(\overline{O}_1 + C_1)$$

A equação acima descreve que a saída Sa_0 do comutador recebe os valores da entradaEnt_0 e seus respectivos bits de configuração, ocupado O_0 e cruzado C_0 barrados*AND* entrada Ent_1, com ocupado O_1 barrado e cruzado C_1.

A Figura 2.16 (a) ilustra a lógica de controle que permite esta. Como ilustrado na figura cada comutador possui um par de bits para cada saída, sendo um para informar se ele está ocupado e outro pra informar se a propagação deve ser direta ou cruzada. Caso o comutador não esteja ocupado (1) o valor da entrada é propagado pra aquela saída, do contrário o valor da entrada será propagado apenas para aquela direção, fazendo uso da capacidade *multicast* dos comutadores. Esta propagação pela rede caracteriza a operação de *broadcast*, que se repetirá até que atinja o estágio final. A Figura 2.18 exemplifica a propagação mostrando todas as saídas alcançáveis pelo *broadcast* partindo da entrada um.

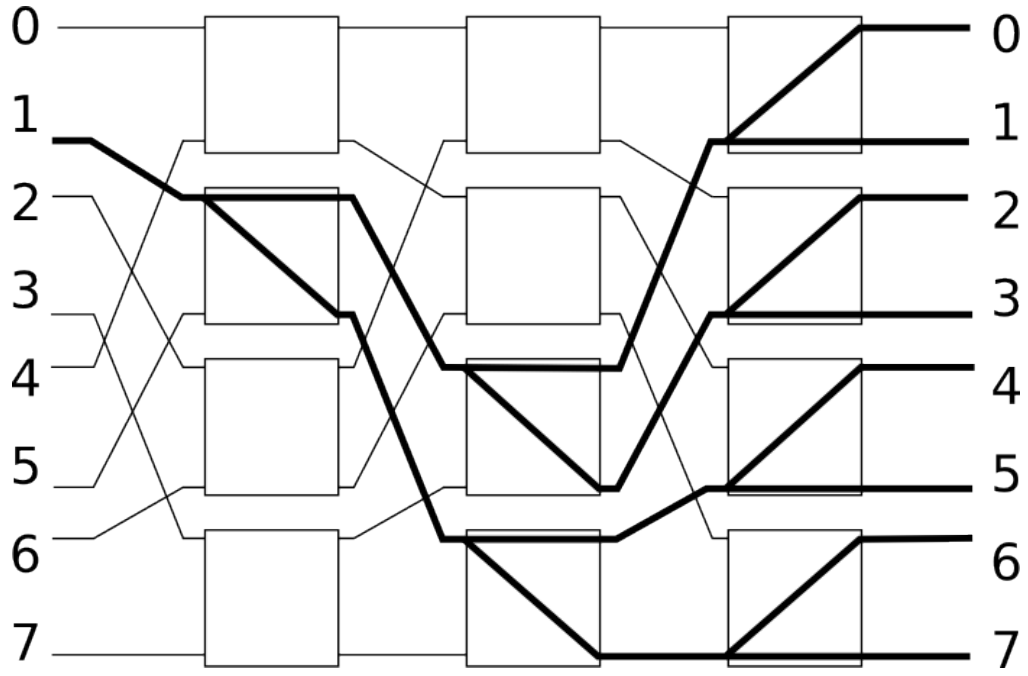


Figura 2.18. Possibilidades de roteamento para a entrada um da rede Omega 8x8

Na figura acima todas as rotas são alcançadas, pois a rede está livre.Agora vamos considerar a possibilidade da rede já possuir algumas conexões pré-estabelecidas, como ilustra a Figura 2.19.

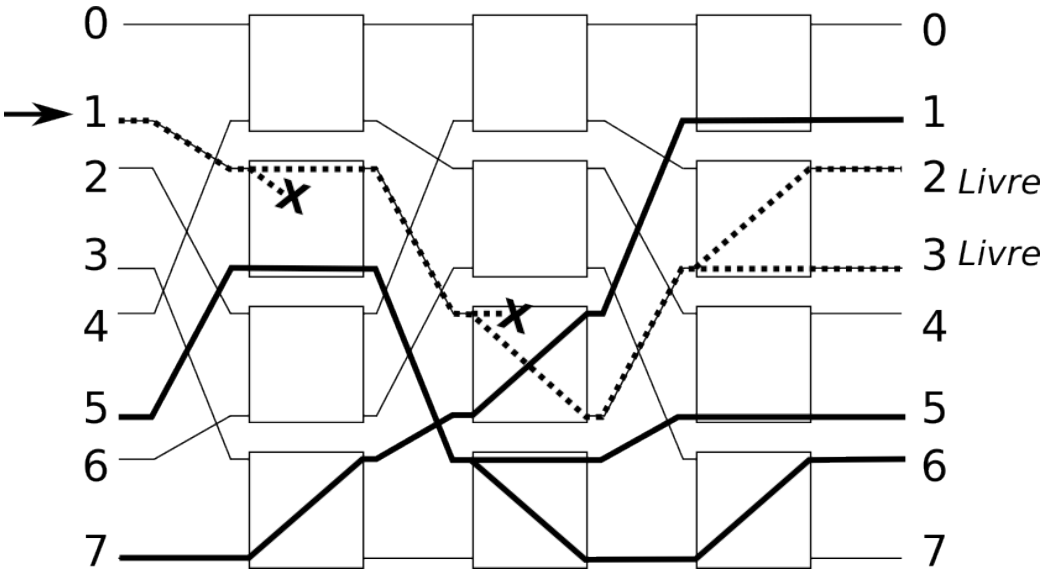


Figura 2.19. Possibilidades de roteamento para entrada 1 em uma rede parcialmente ocupada

Nesta figura as rotas ocupadas estão destacadas em negrito e as linha tracejadas representam as tentativas de roteamento para a entrada 1 e quais os caminhos livres. No exemplo em questão apenas as saídas 2 e 3 são alcançáveis e podem ser roteadas. Podemos notar que neste caso dois comutadores (um no primeiro estágio e outro no segundo) estão restringindo as possibilidades de roteamento.

Supondo que a conexão entre o par de entrada/saída 2 e 3, tenha sido estabelecida. Como ilustrado na Figura 2.20.

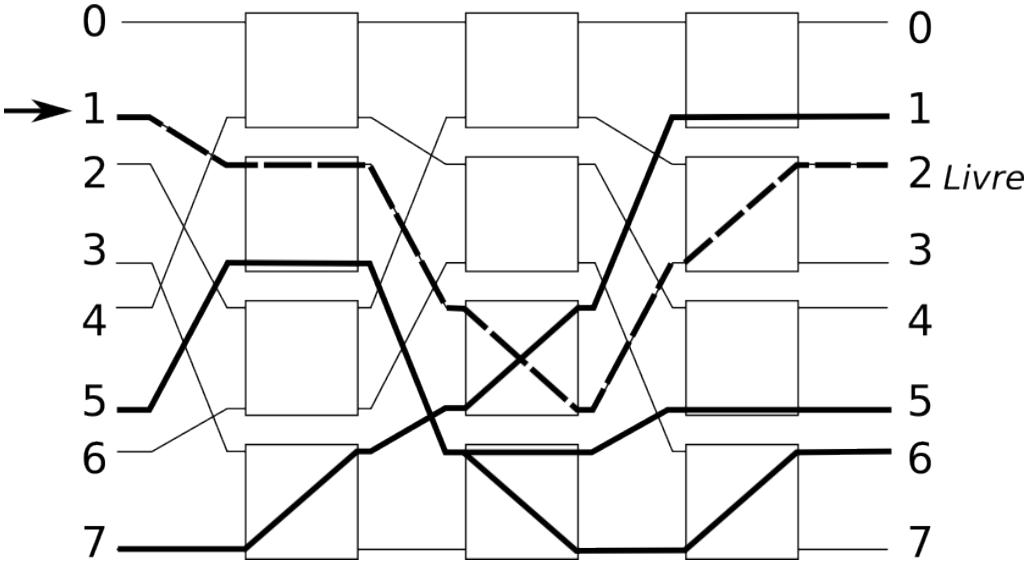


Figura 2.20. Rede Omega8x8 com conexão entre a entrada ume saída dois

Vamos verificar as possibilidades de roteamento para a entrada 6. A Figura 2.21 demonstra que todas as saídas são alcançáveis e roteadas mesmo que a rede já tenha 7 comutadores ocupados.

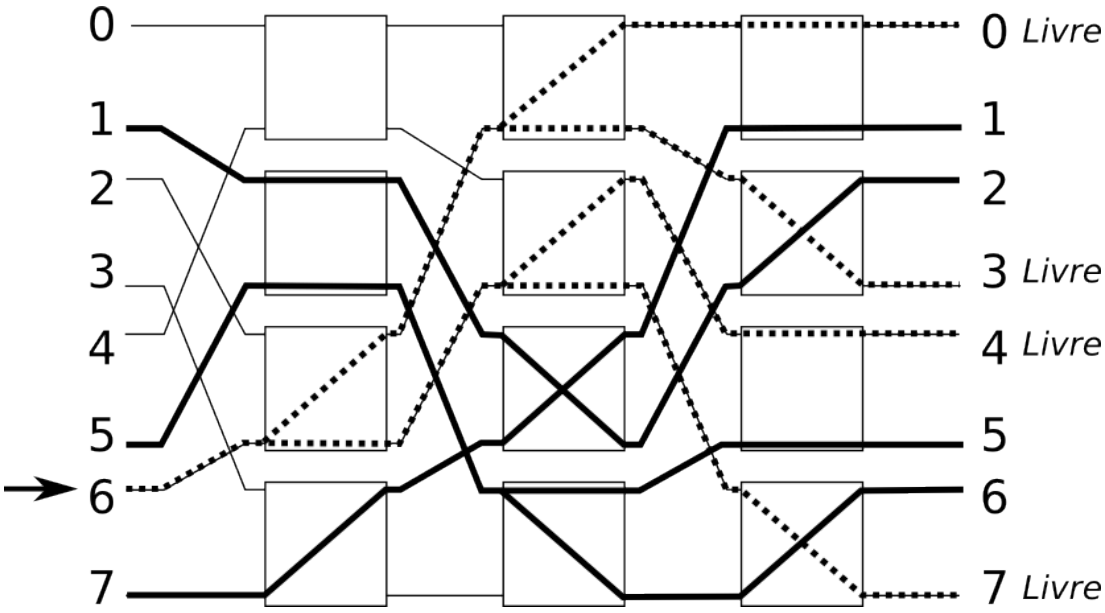


Figura 2.21. Possibilidades para a entrada seis na rede Omega 8x8

A 3ª e última etapa consiste em analisar quais as saídas foram alcançadas no último nível, selecionar uma das saídas e atualizar os comutadores usando na rota escolhida. Dada às saídas alcançáveis um decodificador de prioridade vai determinar a 1ª saída livre.

No caso específico da arquitetura apresentada neste trabalho os dados da instrução devem ser encaminhados para uma unidade funcional correspondente a operação executada, contudo podem existir falhas tanto na rede quanto nas unidades funcionais, pensando nisto o algoritmo proposto em (FERREIRA *et al.*, 2009) foi estendido para considerar falhas. Esta abordagem será apresentada em detalhes no Capítulo 4.

2.5 Tolerância a Falhas em Redes Multiestágio

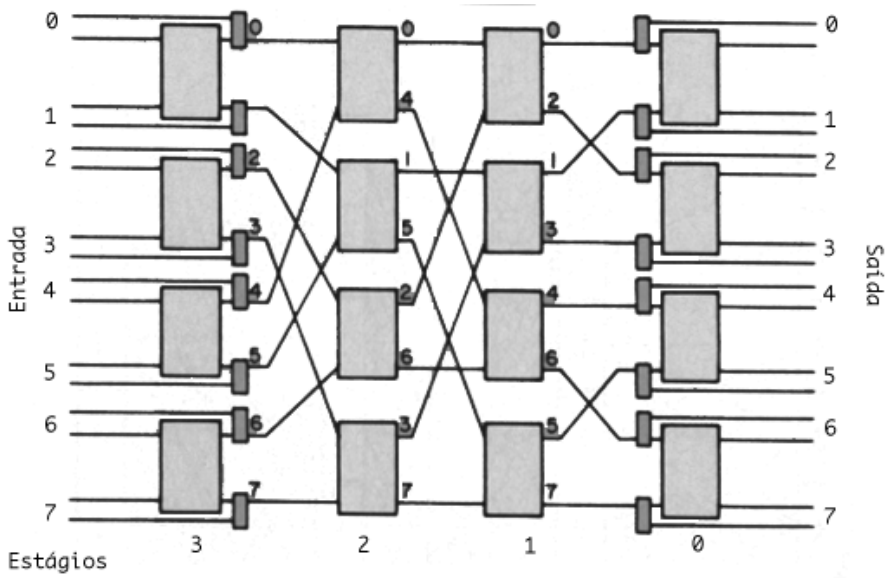
Tolerância a falhas é uma qualidade crucial em redes de interconexão para sistemas multiprocessados de larga escala. As características oferecidas pelas redes multiestágio, como uma boa relação entre custo e desempenho, fez da rede multiestágios uma das mais utilizadas em sistemas multiprocessados na década de 80 (ADAMS *et al.*, 1987). Por este motivo esta seção apresentará alguns modelos de redes multiestágio intrinsecamente tolerantes a falhas. Iremos apresentar dois

modelos, a Cubo Extra-estágio e a Dinamicamente Redundante e,por fim, abordaremos a MIN Dilatada apresentada em (KAMIURA *et al.*, 2000), Estas abordagens foram selecionadas por terem um custo menor. Outras abordagens podem ser encontradas em (ADAMS *et al.*, 1987).

Antes de detalharmos as redes precisamos definir alguns conceitos. Falhas podem ser tanto permanentes quanto transientes, mas para este trabalho e nas redes descritas nesta seção vamos assumir que as falhas são permanentes.

2.5.1 Rede Cubo Extra-estágio

A rede Cubo Extra-estágio adiciona estágios extras à rede, com multiplexadores e demultiplexadores. Em adição são inseridos fios de entrada/saída e para os dispositivos a que a rede se conecta. A Figura 2.22 ilustra a estrutura da rede Cubo Extra-estágio para $N = 8$. Os multiplexadores e demultiplexadores permitem que um comutador com falha seja transposto. O estágio dobra o número de caminhos para cada par de entrada e saída.

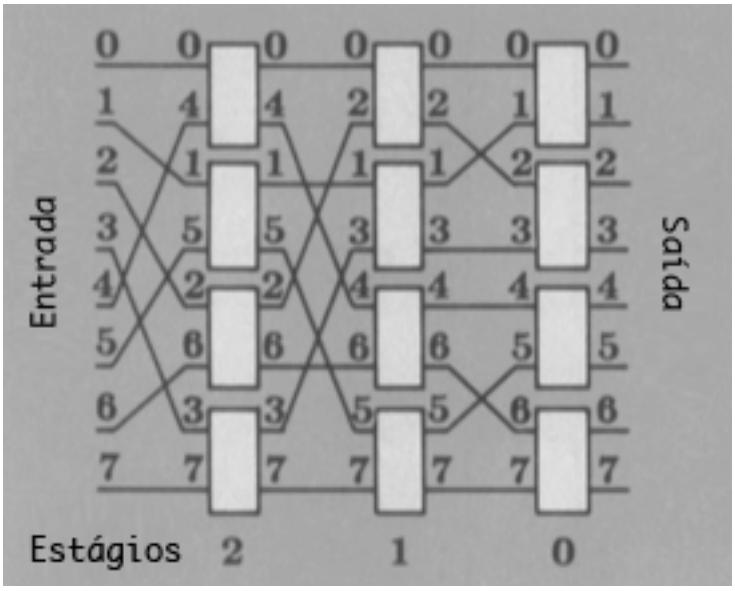


Fonte: Adaptado de (ADAMS *et al.*, 1987)

Figura 2.22. Rede Cubo Extra-estágio

Esta rede MIN apresenta uma estrutura equivalente a seis estágios e seu custo é o dobro de uma única rede Omega equivalente. Além disso, ela só provê tolerância a uma única falha.

2.5.2 Rede Dinamicamente Redundante



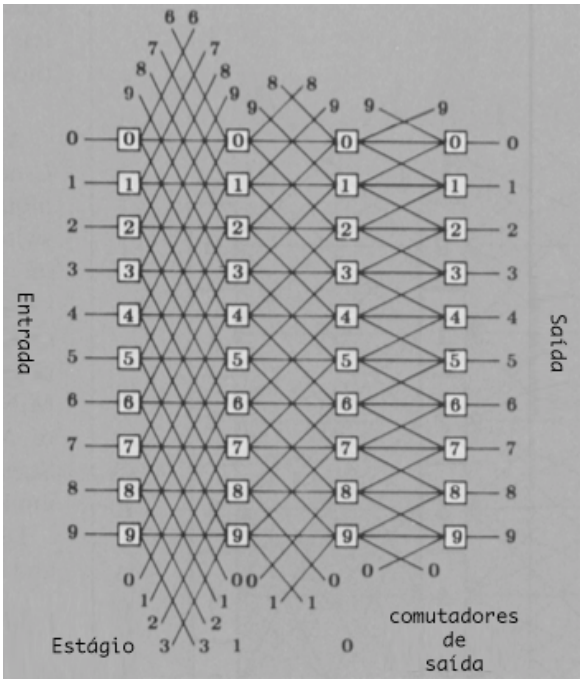
Fonte: Adaptado de (ADAMS *et al.*, 1987)

Figura 2.23. Rede Cubo generalizada $N = 8$

A rede Dinamicamente Redundante (*Dynamic Redundancy Network* ou DR) é também uma variação rede Cubo generalizada, ilustrada na Figura 2.23.

Esta rede possui $N + S$ portas de entradas e saídas e $\log_2 N$ estágios, cada estágio com $N + S$ comutadores seguidos por $3(N + S)$ fios. Cada comutador j no estágio i da rede tem três fios para o estágio $i - 1$. O primeiro fio se conecta ao comutador $((j - 2^i)$ módulo $(N + S))$, o segundo ao comutador j, e o último ao comutador $((j + 2^i)$ módulo $(N + S))$. A Figura 2.24 apresenta uma DR com $N = 8$ e $S = 2$, ondem S corresponde as portas extras disponíveis.

Quando não existe falhas na rede esta emula uma rede Cubo padrão. Se um componente da linha j for identificado como falho, a rede será reconfigurada e um comutador fisicamente numerado p será logicamente renumerado para $t(p)$, onde $t(p) = (p - j - S)$ *módulo* $(N + S)$. Contanto que exista N linhas adjacentes restando a DR vai poder representar um sub-grafo da rede Cubo permitindo o roteamento e assim se comportando como um Cubo tolerante a falhas. A reconfiguração ocorre na após a fase de fabricação, onde as falhas são detectadas através de testes tradicionais e então a rede é reconfigurada formando um sub-grafo da rede Cubo.



Fonte: Adaptado de (ADAMS et al., 1987)

Figura 2.24. Rede Dinâmicamente Redundante

2.5.3 Rede MIN Dilatada

Esta rede foi apresentada em (KAMIURA *et al.*, 2000) e é baseada em grandes comutadores radix Figura 2.25. Esta abordagem dobra o número de conexões entre estágios e utiliza comutadores 4 x 4.

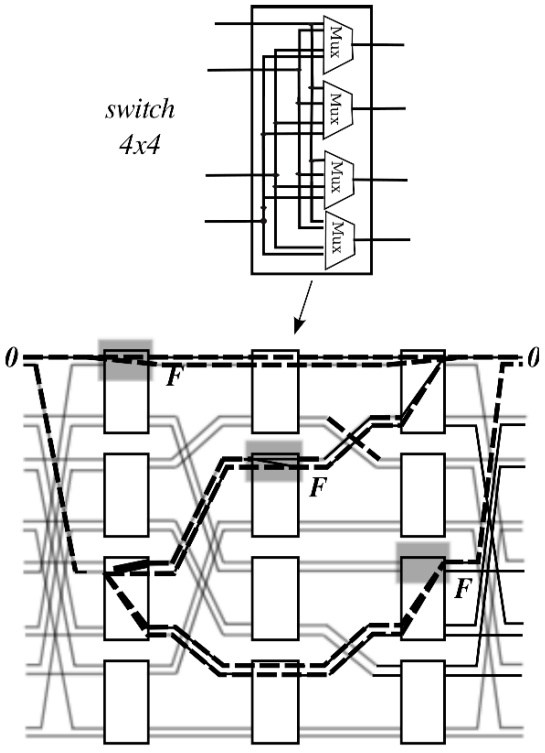


Figura 2.25Rede MIN Dilatada

Ela proporciona *n* múltiplos caminhos entre os pares de entrada/saída, como apresentado na Figura 2.25. Contudo, os múltiplos caminhos usam um conjunto não distinto de comutadores. Se considerarmos $n = 8$, mesmo havendo 8 caminhos distintos, apenas 3 falhas são suficientes para desconectar um par de entra/saída, como apresentado na figura acima. O custo de implementação pode ser até seis vezes mais caro quando comparado ao de uma rede multiestágio Omega simples. A versão 2-dilatada pode ser classificada como de redundância *N*-modular uma vez que é baseada em comutadores 4 x 4.

2.5.4 Relevância

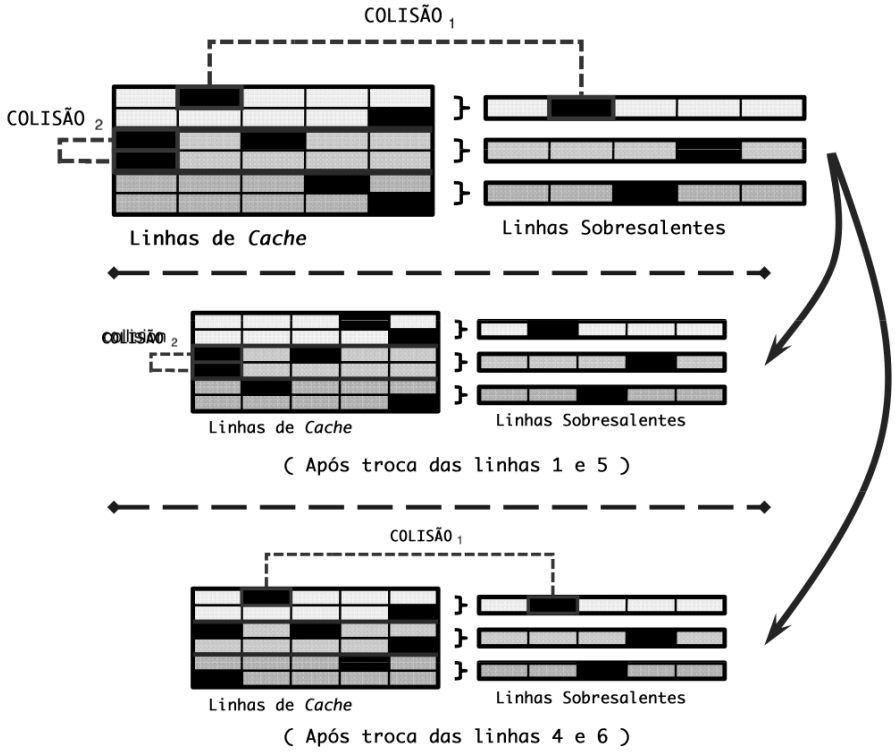
A maior parte das abordagens de tolerância das falhas consiste em aumentar o número de estágios ou comutadores, fios ou modelos diferentes e mais complexos de comutadores. As redes de interconexão apresentadas nesta seção não fugiram destas abordagens e por isso sofreram com as contraposições inerentes às mesmas, como aumento do custo e complexidade de roteamento sem obter em contrapartida ganho em desempenho ou tolerância a falhas proporcional aos recursos gastos.

A maioria dos artigos apresentados em (ADAMS *et al.*, 1987) ficaram no campo teórico, não existindo implementação prática ou viável em *hardware*. A solução de tolerância em rede MIN apresentada neste trabalho não adiciona grande complexidade ao mecanismo de roteamento além de ter um custo em área inferior aos trabalhos anteriores.

A despeito do fato da grande maioria dos estudos sobre tolerância a falhas em redes multiestágio se concentrarem nas décadas de 70 e 80; este ainda continua sendo um tópico relevante. Uma confirmação disto reside em uma proposta recente de arquitetura para cache em nano tecnologia apresentada em (ANSARI *et al.*, 2009). A seção a seguir destaca algumas das contribuições desta arquitetura.

2.5.5 Multiestágio Aplicada as Novas Tecnologias

A arquitetura apresentada em (ANSARI *et al.*, 2009) propõe um modelo de *cache* utilizando redes multiestágio para substituir blocos com falhas. A Figura 2.26 apresenta uma representação simplificada no modelo de *cache*, tendo do lado esquerdo da figura a cache original e do lado direito linhas de reposição.



Fonte: Adaptado de (ANSARI *et al.*, 2009)

Figura 2.26. Dois cenários simples onde a permuta de linha preservou o funcionamento correto da cache ao resolver a colisão. As caixas pretas representam um conjunto defeituoso de dados

As linhas de reposição são exatamente iguais às linhas de *cache* originais, contudo elas existem em uma proporção reduzida para não duplicar o custo da *cache*. Estas linhas são acionadas quando existe uma falha em um bloco de dados da cache original, entretanto elas também estão sujeitas a terem blocos falhos.

A parte superior da Figura 2.26 ilustra uma situação de colisão entre a *cache* e a linha de reposição. A linha 1 da cache apresenta uma falha no bloco 2, sendo que a linha de reposição também apresenta falha neste mesmo bloco. Para resolver este impasse é realizada uma permutação das linhas de cache através de uma rede multiestágio Benes não bloqueante, Figura 2.27. No exemplo ilustrado a troca foi feita entre a linha 1 e a 5. Uma vez que a linha 5 não possui defeito no bloco 2.

O roteamento dos blocos de reposição usa um algoritmo baseado na coloração de grafos. O algoritmo executa uma única vez, antes do uso da cache. Uma vez configurada estaticamente a rede, a cache mantém a configuração que não muda durante a execução.

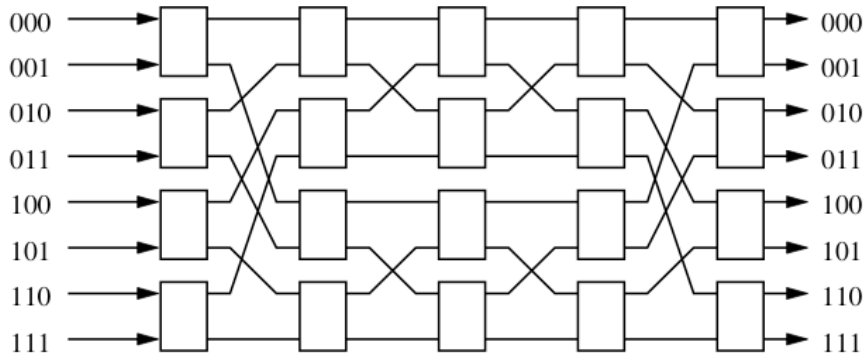


Figura 2.27. Rede Benes com *N* = 8

A maior relevância da solução apresentada em (ANSARI *et al.*, 2009) para o nosso trabalho, está no fato de ele mostrar que redes multiestágio são soluções viáveis mesmo em sistemas onde o tempo é crítico, como em *caches*. Além disso, ele mostra que este tipo de rede se adequa às tecnologias mais recentes de fabricação de circuitos. A diferença para a solução adotada nesta dissertação é o fato de executar o roteamento dinamicamente durante toda a execução enquanto que a rede da cache é configurada estaticamente após a fabricação e detecção das falhas.

3 ARQUITETURAS DINAMICAMENTE RECONFIGURÁVEIS COM TRADUÇÃO BINÁRIA

Este capítulo apresenta a arquitetura base proposta em (BECK *et al*., 2008), assim como as modificações sugeridas pelo trabalhos correlatos (RUTZIG *et al.*, 2008; LAURE, 2010; PEREIRA *et al.*, 2009). Ao final do capítulo será apresentado um resumo de cada arquitetura relacionada e suas principais características.

A arquitetura Super-VLIW proposta neste trabalho, que será descrita no próximo capítulo é uma derivação da arquitetura base descrita neste capítulo. A Figura 3.1 apresenta de forma esquemática as contribuições dos trabalhos supracitados assim como a deste trabalho.

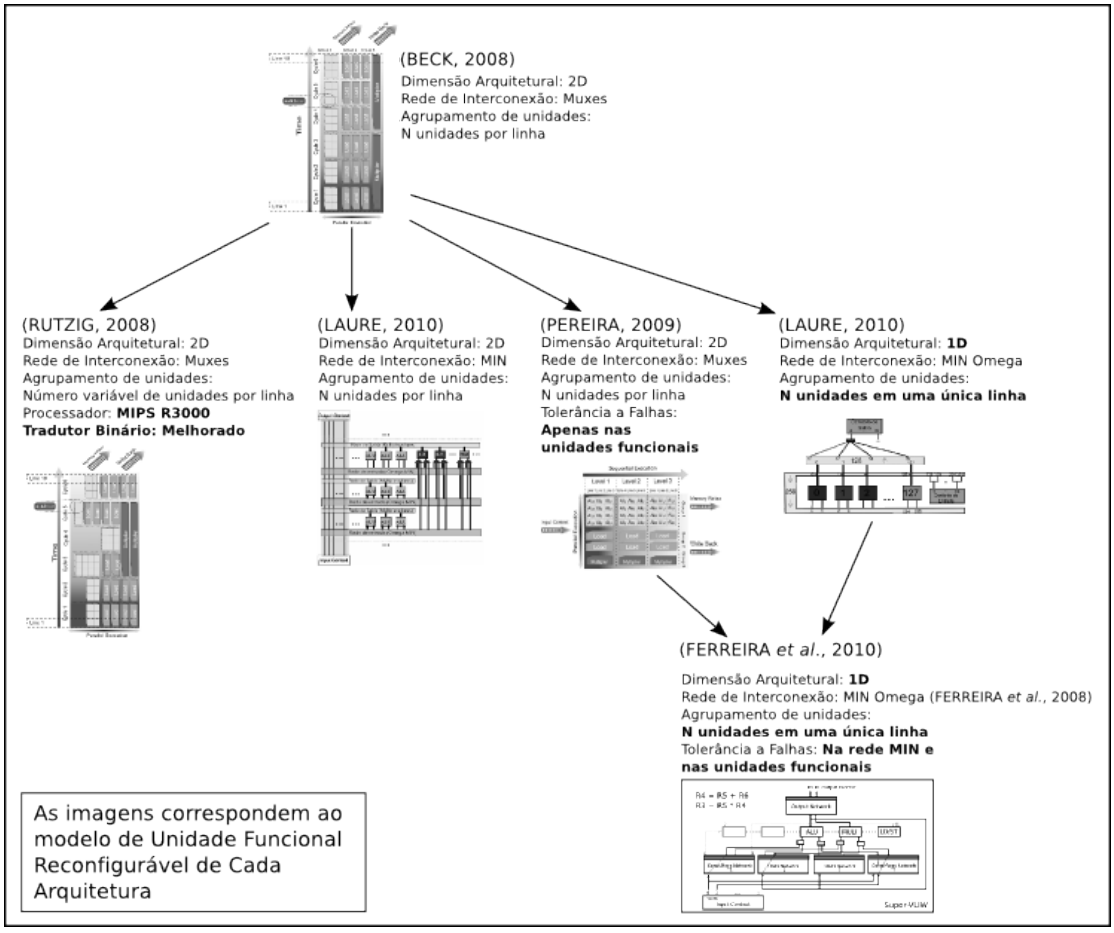


Figura 3.1. Diagrama de blocos do sistema reconfigurável proposto em (BECK, 2008) e nos trabalhos derivados

3.1 Arquitetura Base

A arquitetura reconfigurável proposta por (BECK, 2008) é dinamicamente reconfigurável e de grão-grosso, além disso, é acoplada a um processador RISC modelo MIPS R3000. Nesta arquitetura a unidade reconfigurável é composta por unidades funcionais conectadas através de multiplexadores. A Figura 3.2 ilustra um diagrama da arquitetura base completa, os blocos em cinza escuro representam os estágios do *pipeline* do processador MIPS R3000. Em cinza claro, estão os blocos que simbolizam as unidades, que foram acopladas ao processador, responsáveis por realizar a execução de forma reconfigurável: *hardware* de tradução binária (TB), unidade funcional reconfigurável (UFR) e a *Cache* de reconfigurações. Cada componente ilustrado neste diagrama será detalhado nas próximas seções, com exceção do *pipeline* do MIPS por ser uma estrutura bem conhecida na área da computação; para maiores detalhes sobre este consulte (RUTZIG, 2009).

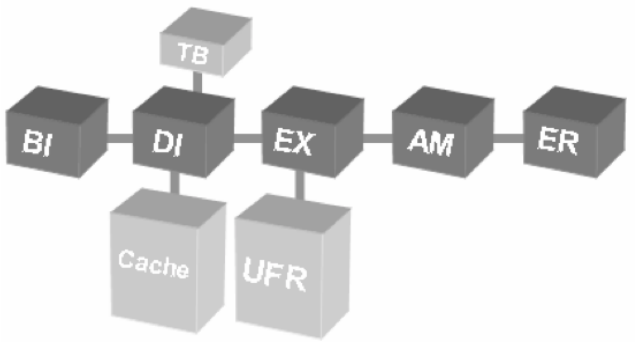


Figura 3.2. Diagrama de blocos do Sistema Reconfigurável (BECK, 2008)

3.2 Tradução binária

Atualmente, o tempo de projeto é uma das principais restrições para se lançar um novo dispositivo embarcado no mercado. A concorrência acirrada das empresas para disponibilizar no mercado o dispositivo com as funcionalidades mais recentes tem forçado a um tempo entre a concepção do projeto e lançamento no mercado cada vez mais curto. Em decorrência disto, os projetistas devem dispor de técnicas que os ajudem a aproveitar o código legado, softwares escritos para os dispositivos anteriores nas novas plataformas.

Tradução binária é uma das técnicas utilizada em processadores de propósito geral para prover compatibilidade de software (RUTZIG, 2008). Originalmente o mecanismo de tradução binária transparente foi proposto em (GSCHWIND *et al.,* 2000). A utilização desta técnica em sistemas embarcados foi proposta em (BECK,

2005), acoplado o TB a um processador Java. Neste trabalho foi demonstrado que, com a utilização de um tradutor binário e uma unidade reconfigurável, acoplada a um processador Java, eleva-se o desempenho e diminui-se a energia consumida pelo sistema.

A ideia básica do mecanismo é prover compatibilidade de software a partir da tradução binária de sequências de instruções aliada à técnica denominada *Dynamic Instruction Merge* (DIM) (BECK, 2006). Atuando em tempo de execução, elas permitem que trechos do código sejam selecionados e executados posteriormente em mecanismo mais eficiente, no caso deste trabalho e dos correlatos, uma unidade funcional reconfigurável de grão grosso.

Em (RUTZIG, 2008) a mesma abordagem proposta por (BECK, 2005) foi utilizada, contudo, em conjunto com um processador embarcado de uso geral o MIPS R3000.

3.2.1 Detecção e Execução

O processo de tradução binária dá-se a partir do momento que uma aplicação entra em execução pela primeira vez. O hardware de tradução binária monitora em paralelo cada instrução executada no processador, sempre que possível ele as agrupará em blocos denominados configuração da UFR. Cada configuração corresponde a um estado que a UFR pode assumir para lidar com as instruções daquele conjunto. Sempre que um bloco de configuração é detectado, ele é armazenado na *cache* de configuração, indexada pelo contador de programa (*Program Counter*) da primeira instrução do bloco. Quando um bloco de instrução é novamente detectado ao longo da execução, este é resgatado da *cache* e é encaminhado para a UFR.

Considerando o funcionamento da arquitetura, é possível nomear dois tempos distintos de execução das instruções:

1º Tempo de Detecção – Execução do programa em paralelo no processador e no DIM. Neste momento o DIM realiza a tradução das instruções lógicas para a lógica da UFR, com a montagem dos blocos e armazenamento das configurações na cache de configuração;

2º Tempo de Reconfiguração – Neste momento uma configuração que foi executada e está disponível na cache é recuperada e mapeada para a UFR. Neste

ponto é que a aplicação sofre efeito de aceleração com a exploração do paralelismo entre as instruções.

3.3 Unidade Funcional Reconfigurável

Nesta seção serão apresentadas as Unidades Funcionais Reconfiguráveis empregadas na arquitetura base e nas demais arquiteturas subsequentes. Através do mapeamento de um exemplo pretendemos apresentar de forma sucinta as principais características de cada arquitetura. Como iremos mostrar no capítulo 5, os blocos dos códigos avaliados podem variar de 3 a 300 instruções.

Suponha o trecho de código da Figura 3.3 (a). O bloco começa com a instrução *Add R1,R2,R3* e termina no desvio *bne*. A figura 3.3 (b) mostra o grafo de dependência das instruções. Podemos observar que existem várias instruções que podem ser executadas em paralelo. A seguir iremos mostrar como o código é mapeado na arquitetura base e nas extensões do modelo.

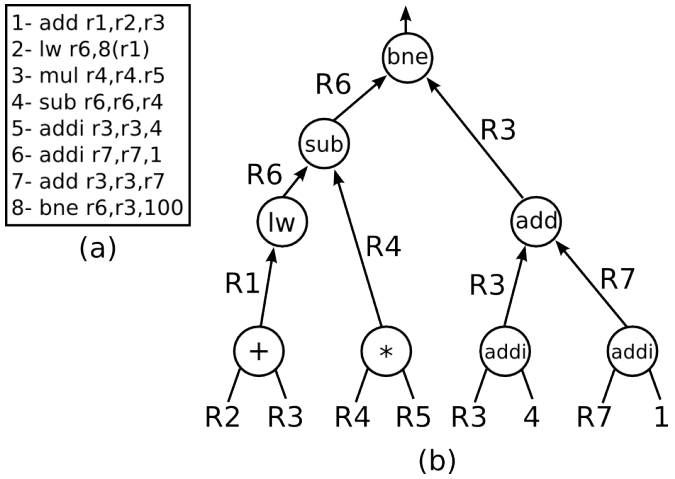


Figura 3.3. (a) Conjunto de instruções MIPS; (b) Grafo de dependência das instruções

A figura 3.4 ilustra a unidade reconfigurável da arquitetura base. Suponha um modelo simples com as seguintes unidades funcionais por linha: 9 *ALU*, 2 *LW* e 1 *MUL*. Como a *ALU* executa em um terço do ciclo, temos três sub-linhas com três *ALU* cada. Na fase de detecção da tradução binária o código é executado no MIPS e o mapeamento gerado para o arranjo bi-dimensional é gravado na cache. Aprimeira instrução, o *ADD*, é mapeada na primeira *ALU*. A segunda instrução tem que ser mapeada na segunda linha devido a dependência em R1 (*read after write*) com a primeira instrução. A terceira instrução, o *MUL*, não depende das anteriores e é mapeado na primeira linha. Se existisse um outro *MUL*, este não poderia ser

mapeado na primeira linha por falta de unidade, mas não é o caso deste exemplo. A quarta instrução, o *SUB*, é mapeado na terceira linha pois usa o valor de R6 gerado pela segunda instrução, o *LW*. A quinta e sexta instrução são mapeadas na primeira linha pois não possuem dependências. Já a sétima instrução é mapeada na segunda linha pois depende dos valores de *R3* e *R7* gerados pela quinta e sexta instrução. A oitava instrução, o *BNE* pode ser mapeada na quarta linha pois depende da quarta e sétima instrução.

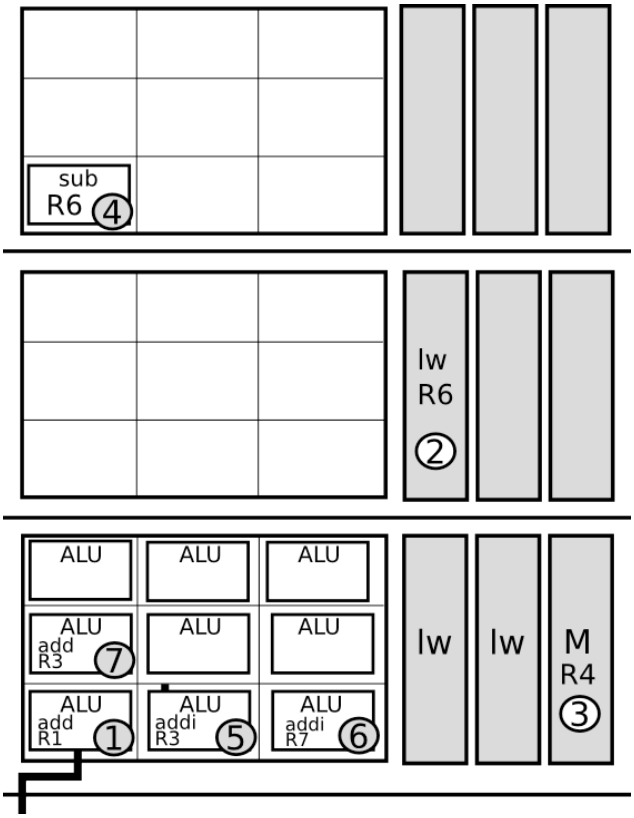


Figura 3.4. Arquitetura (BECK, 2008) executando o conjunto de instruções apresentado na Figura 3.3 (a)

A primeira arquitetura derivada proposta por (RUTZIG, 2009) busca reduzir o número de unidades ociosas. A arquitetura original é regular e possui o mesmo número de unidades por linhas. No exemplo anterior, a linha 2 usa apenas um LW e todas as outras unidades ficam inativas. A solução proposta foi executar vários benchmarks e registrar o máximo de unidades usadas por linha. Se fosse gerar uma arquitetura específica para o bloco da Figura 3.3 (a), teríamos a arquitetura da Figura 3.5 com o número exato de unidades. Entretanto, temos que gerar uma arquitetura compatível com um conjunto de programas, ou seja, para um conjunto bem maior de blocos.

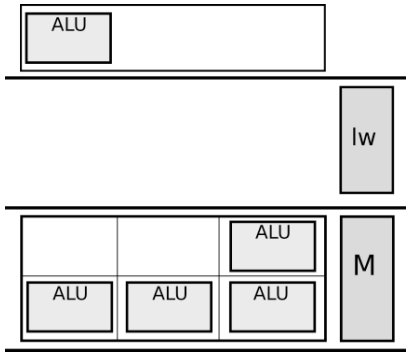


Figura 3.5. Modelo simplificado da fábrica reconfigurável da arquitetura proposta em (RUTZIG, 2009)

A segunda arquitetura derivada, proposta em (LAURE, 2010) é baseada na constatação que 40% da área é dedicada aos elementos de interconexão. Para cada unidade do arranjo bidimensional é preciso dois multiplexadores com 32 entradas, se considerarmos 32 registros no processador. Além disso, para cada linha de saída é necessário um multiplexador que recebe os valores das unidades como ilustra a Figura 3.6 (a). A rede de multiplexadores de entrada por ser substituída por uma rede MIN, reduzindo o custo $O(N^2)$ da rede de MUX para um custo $O(N \text{ Log } N)$ da rede MIN. Foi mostrada em (LAURE, 2010) que mesmo tendo conflitos de roteamento na rede MIN, a perda de desempenho é mínima, e o custo do arranjo pode ser reduzido em 30%.

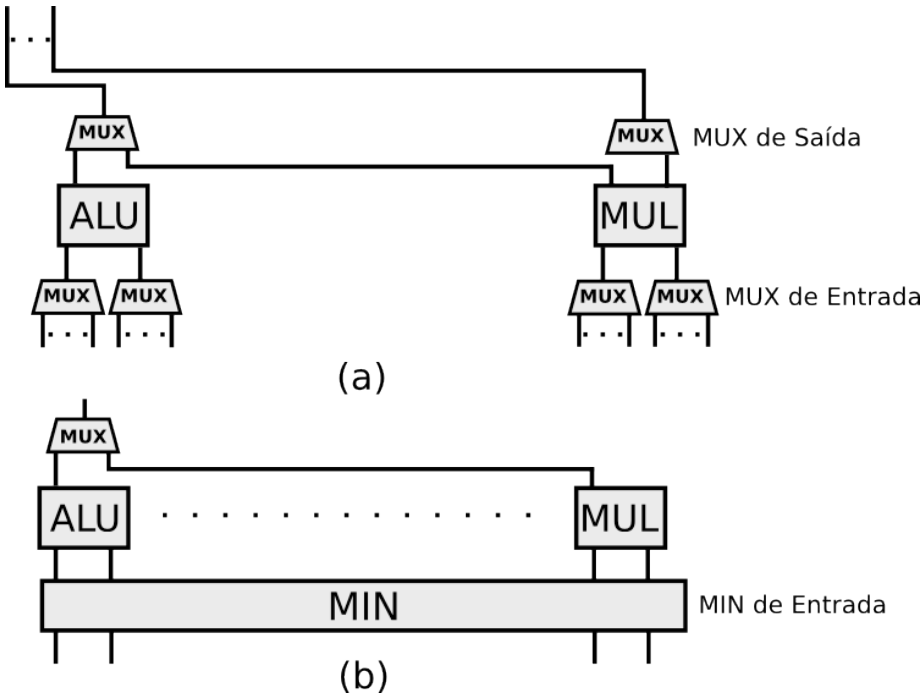


Figura 3.6. (a) (BECK, 2009) (b) (LAURE, 2010)

Uma terceira arquitetura foi também proposta em (LAURE, 2010) usando uma única rede multiestágio global e um arranjo unidimensional das unidades. A Figura 3.7 mostra o exemplo da Figura 3.3 (a) mapeado na arquitetura unidimensional. O mecanismo global de comunicação permite um uso otimizado das unidades funcionais, pois a unidade não é alocada em uma linha fixa como no modelo bidimensional. Esta arquitetura busca tanto reduzir o número de unidade com a proposta de (RUTZIG, 2009) como também a área das interconexões ao usar redes multiestágios no lugar de multiplexadores.

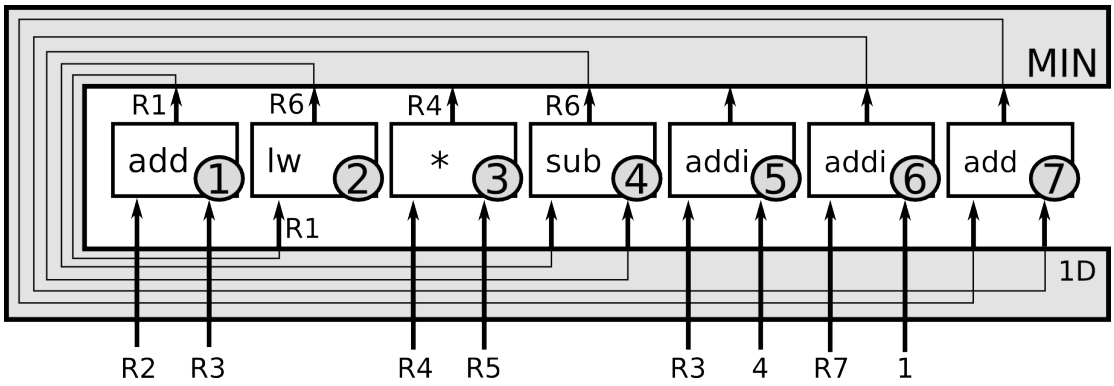


Figura 3.7. Modelo unidimensional com multiestágio proposto em (LAURE, 2010)

Uma quarta arquitetura é proposta em (PEREIRA*et al.*, 2009) incorporando a tolerância a falhas. A arquitetura é baseada no modelo original (BECK, 2008), porém tanto as unidades quanto os multiplexadores podem apresentar falhas. Em caso de falha a unidade fica sempre marcada como alocada. A Figura 3.8 mostra o arranjo com algumas unidades com falhas. Pode-se observar que o mapeamento irá se ajustar e alocar as unidades sem falhas. A tradução binária não muda, apenas existe uma fase de inicialização que marca as unidades com falhas como ocupadas.

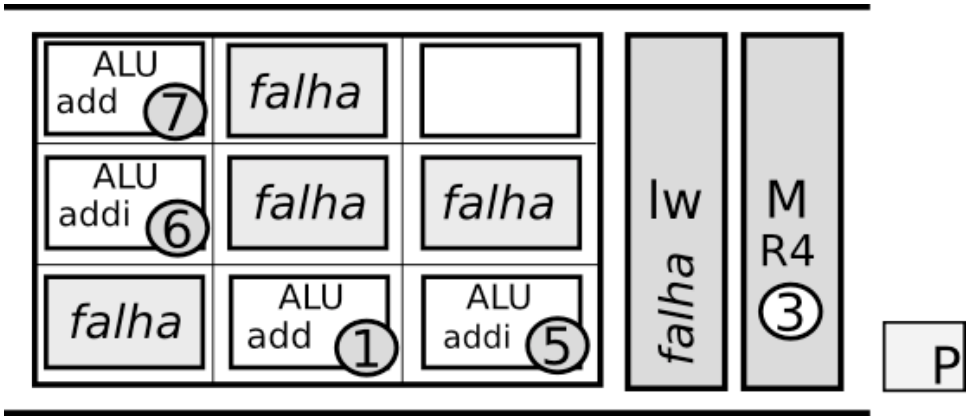


Figura 3.8. Modelo da Unidade Funcional Reconfigurável tolerante a falhas proposta em (PEREIRA *et al.*, 2009)

Nesta dissertação a proposta foi de incorporar um mecanismo de tolerância a falhas (PEREIRA*et al.*, 2009) na arquitetura uni-dimensional proposta em (LAURE, 2010), ou seja, reduzir a área das unidades, das conexões e ainda suportar as falhas que ocorrem com maior frequência nas tecnologias em escala nanométrica.

3.4 Tamanho da Configuração

Existem alguns fatores que são responsáveis por interromper a formação de uma configuração. A ocorrência de alguns tipos específicos de instruções como de desvio ou instruções que realizam operações de divisão, esta última por não ser suportada pela UFR, são alguns destes fatores.

Quando uma instrução que não é suportada pela UFR é encontrada a configuração corrente é concluída e armazenada na cache de configuração. Na sequência da execução uma nova configuração será iniciada, a partir do momento que for encontrada uma instrução executável na UFR e não for uma instrução de desvio.

Um mecanismo de especulação de desvios foi proposto em (BECK *et al.*, 2006). O objetivo é aumentar o tamanho da configuração e explorar um maior potencial de paralelismo. Em outras palavras a especulação não interrompe a formação de uma configuração quando encontra uma instrução de desvio, ela apenas será interrompida quando o número de desvios encontrados for igual ao definido como tolerável pelo projetista. Entretanto, a penalidade por erro de especulação é proporcional ao número de especulações.

Outro fator que pode determinar a quebra de um bloco é a ausência de unidades funcionais livres. Por exemplo, se a próxima instrução do bloco for uma operação

aritmética e não existirem mais unidades desta natureza disponíveis o bloco será finalizado por falta de unidades. Ainda existe a possibilidade de quebra da configuração por falha de roteamento. Isto só ocorre nos modelos com multiestágio, uma vez que as redes de multiplexadores não geram conflito de roteamento. A falha de roteamento ocorre sempre que não for possível achar uma rota para um determinado tipo de UF a partir de uma entrada ou saída específica. Esta ausência de rota pode acontecer pela presença de defeitos na rede ou por conflito de roteamento, uma vez que usamos redes multiestágio bloqueantes.

4 ARQUITETURA SUPER-VLIW

Na década de 80 surgiu um novo modelo de arquitetura denominada *Very Long Instruction Words* (VLIW) e concebida por Joseph Fisher. Como tantas outras, ela surgiu com o objetivo de aumentar a exploração do paralelismo em nível de instruções (*Intruction-Level Paralelism*).

A definição formal deste modelo varia de autor para autor e com as características de algumas implementações. Originalmente (FISHER, 1983) definiu as seguintes propriedades da arquitetura.

Cada instrução longa consiste de muitas operações independentes agrupadas, onde cada operação requer um pequeno, estático e previsível número de ciclos para executar.

Conceitualmente microprocessadores VLIW e Superescalares compartilham algumas características como a capacidade de executar múltiplas operações simultaneamente e a existência de múltiplas unidades de execução. Contudo a técnica empregada no VLIW é baseada no paralelismo explícito em suas instruções, resolvido em tempo de compilação, enquanto nas superescalares ele precisa ser descoberto pelo hardware em tempo de execução. Ao transferir a tarefa para o compilador, o VLIW simplifica a lógica de controle do processador uma vez que o hardware parte da premissa que as instruções geradas podem ser executadas ao mesmo tempo.

Um dos grandes entraves para o sucesso das arquiteturas VLIW reside no fato do mercado de microprocessadores estar alicerçado no conjunto de instruções x86 ou RISC no caso dos dispositivos móveis. Processadores VLIW são originalmente incompatíveis com estes conjuntos de instruções, além disso, uma instrução VLIW pode ser incompatível entre processadores desta mesma arquitetura, quando o número de unidades funcionais for diferente. Este fator rompe com o paradigma da compatibilidade de código, o que forçaria o mercado readaptar-se a cada versão de processador VLIW.

Mesmo com estes complicadores o mercado e pesquisadores ainda demonstram bastante interesse neste modelo de arquitetura, principalmente na área de dispositivos embarcados onde as restrições de consumo de energia e potência são maiores (CHEN *et al*., 2011).

Fato que comprova isto são os processadores VLIW lançados comercialmente e que tentam solucionar alguns dos problemas conhecidos da arquitetura. Como exemplo destes, podemos citar o Transmeta Crusoe. Este processador VLIW conta com um mecanismo denominado *Code Morphing*, cujo objetivo é realizar a tradução binária dos conjuntos de instruções x86 provendo compatibilidade entre o conjunto legado de instruções com as instruções VLIW deste processador.

A arquitetura apresentada neste trabalho compartilha algumas semelhanças com os processadores VLIW. A principal delas é a capacidade de executar múltiplas operações simultaneamente através de uma palavra de instrução muito longa, de fato, o tamanho da palavra de instruções a ser executada pode atingir super proporções em nossa arquitetura, e é daí que se origina o nome Super-VLIW.

Embora possua semelhanças com os processadores VLIW a arquitetura Super-VLIW extrapola seus conceitos e propõe uma solução mais ampla, dinâmica e fortemente acoplada a um processador de uso geral. A Tabela 4 contrasta algumas características de ambas as arquiteturas.

Table 4.1: Comparativo entre as arquiteturas VLIW e Super-VLIW

Características	**VLIW**	**Super-VLIW**
Necessita compilador ou ferramentas especiais	Sim	Não
Compatibilidade binária	Não é mantida	É mantida

A arquitetura Super-VLIW é uma composição da arquitetura base proposta por (BECK, 2008) e apresentada na Seção 3, com outras contribuições, como o modelo unidimensional e adoção de redes multiestágio propostas em (LAURE, 2010), a rede Omega proposta em (FERREIRA *et al*., 2009) e o conceito de tolerância a falhas proposto em (PEREIRA *et al.*, 2009).

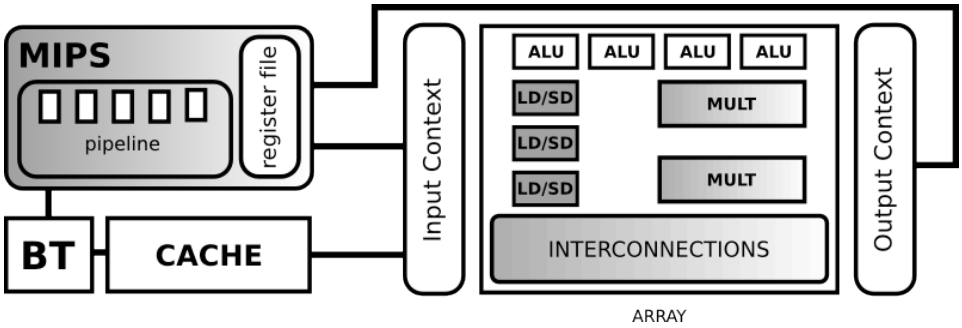


Figura 4.1. Diagrama da arquitetura Super-VLIW

A principal mudanças estrutural da arquitetura Super-VLIW em relação a arquitetura de (BECK, 2008) está na estrutura de interconexão. Em nossa arquitetura a rede de multiplexadores utilizada na arquitetura base foi substituída por um conjunto de redes multiestágio. Isto mudou a topologia da arquitetura de bidimensional para unidimensional, semelhante a modificação introduzida por (LAURE, 20010), entretanto, neste último trabalho foi utilizada apenas uma única rede multiestágio.

A arquitetura Super-VLIW faz uso de cinco redes multiestágio Omega, contudo estas redes são menores que as utilizadas em (LAURE, 2010), a divisão das redes tem uma função de balanceamento de carga e roteamento em paralelo. A estrutura das redes será detalhada na Seção 4.2

4.1 Rede de Interconexão Tolerante a Falhas

Analogamente podemos dizer que este trabalho se assemelha à proposta apresentada por (PEREIRA *et al*., 2009) e discutida na Seção 2. Neste último trabalho, a arquitetura base (BECK, 2008) é estendida com um mecanismo de tolerância a falhas. Assim como (PEREIRA *et al*., 2009) este trabalho também propõe um mecanismo de tolerância a falhas, mas estendendo a proposta de arquitetura 1D com redes MIN apresentada em (LAURE, 2010).

A solução aqui apresentada trata falhas tanto na rede de interconexão quanto nas unidades funcionais, enquanto que em (PEREIRA *et al*., 2009) apenas são tratadas falhas nas unidades funcionais.

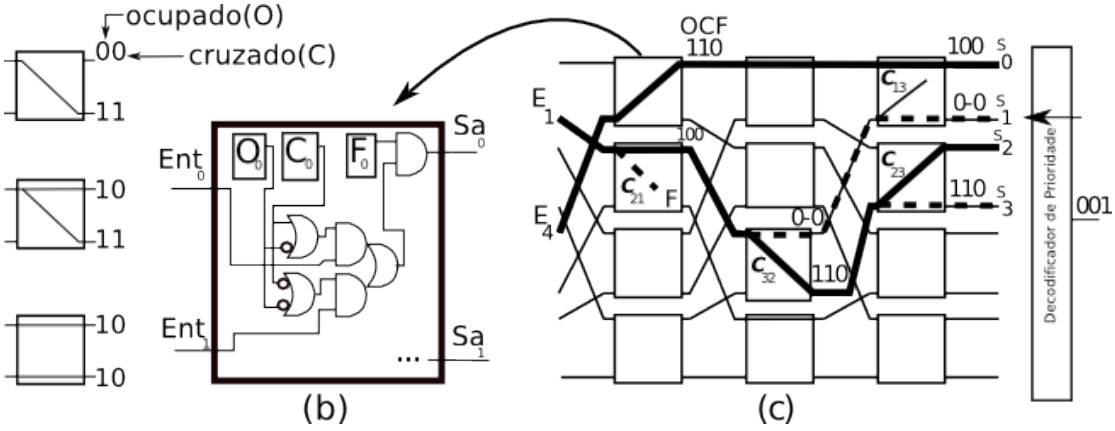


Figura 4.2. Busca e alocação dinâmica: (a) Algumas configurações do comutador; (b) Controlador lógico local do comutador; (c) Exemplo de broadcast

Como citado anteriormente o modelo de rede multiestágio e o algoritmo de roteamento utilizados foram propostos em (FERREIRA *et al*., 2009), a rede e o algoritmo de roteamento foram apresentados no capítulo anterior nas Seções 2.3.1 e 2.3.2 respectivamente. Contudo o algoritmo não levava em consideração a existência de falhas na arquitetura reconfigurável, sendo assim o algoritmo foi modificado para prover tolerância a falhas.

Para tal, nós estendemos a lógica de controle do comutador incluindo um bit extra para registrar a falha. A solução adotada foi semelhante a empregada no trabalho original, preservando assim a independência do padrão de interconexões. A lógica de controle para a saída 0 é descrita pela equação a seguir:

$$Sa_0 = Ent_0(\overline{O}_0 + C_0) + Ent_1(\overline{O}_0 + \overline{C}_0) \cdot \overline{F}_0$$

Para a equação descrita acima temos Ent_0 e Ent_1 representando as entradas do controlador, C_0 e O_0 os bits de cruzado e ocupado respectivamente e finalmente F_0 como o bit de falha da saída Sa_0, como ilustrado na Figura 4.2 (b).

A título de ilustração vamos considerar a configuração da rede MIN apresentada na Figura 4.2 (c), onde duas conexões foram estabelecidas, $E_1 \rightarrow S_2$ e $E_4 \rightarrow S_0$, representadas na figura como linhas em negrito. Vamos considerar também que a saída Sa_1 do comutador $\boldsymbol{c}_{21}$ contêm falha.

Assumindo que queremos conectar a Ent_1 a outra saída livre da rede. Vamos iniciar enviando um sinal de controle em *broadcast* a partir da Ent_1, representado na figura pela linha tracejada, o algoritmo dinâmico de busca e alocação alcançará todas as saídas livres e roteáveis.

No comutador $\boldsymbol{c}_{21}$, a saída Sa_1 está com falha ($F_1 = 1$) e o sinal de controle será bloqueado, representado na figura pela letra F. Embora a saídaSa_0 esteja ocupada ($O_0 = 1$), este é um caso de *multicast* e o bit cruzado está definido para a direção correta, sendo assim, o sinal de controle será propagado para a Sa_0.

No comutador $\boldsymbol{c}_{32}$, a saída Sa_0 está livre ($O_0 = 0$), sendo assim o sinal de controle será propagado, da mesma forma para a Sa_1, sendo esta um caso de *multicast*. Finalmente no último estágio dois comutadores receberão a requisição de roteamento.

No comutador $\boldsymbol{c}_{13}$ a saída Sa_0 está ocupada e bit cruzado está definido para a direção oposta. Contudo a saída Sa_1 é livre e roteável, portando esta saída da MIN é alcançável.

No comutador $\boldsymbol{c}_{23}$ a saída Sa_0 também está ocupada, no entanto é um caso de *multicast*. Sendo assim, S_2 da rede é roteável. Entretanto, esta saída da MIN já está alocada.

Após o último estágio, um vetor de saída da MIN é verificado, nele todas as saídas ocupadas são salvas. No comutador $\boldsymbol{c}_{23}$ também existirá a saída Sa_1 que é livre, roteável e consta livre no vetor de saída. Por fim a primeira saída livre será selecionada pelo Decodificador de Prioridade, que pode ser implementado de maneira eficiente através de técnicas *lookahead* (MOHAN *et al*., 2006).

No caso deste exemplo, a saída S_1 da MIN é selecionada. Esta abordagem permite a MIN a capacidade de conectar-se a alguns pares de entrada/saída mesmo sob a presença de altas taxas de falhas. Outro ponto importante que será mostrar a seguir, é que o objetivo é alocar unidades na saída da rede. Como existem diversas unidades equivalentes como as *ALUs*, se algumas saídas não são alcançadas devido as falhas, pode-se usar apenas as *ALUs* que são roteáveis. Esta característica difere das abordagens tradicionais de multiestágio, onde se deseja que qualquer saída seja sempre alcançada.

4.2 Unidade Funcional Reconfigurável

Em (LAURE, 2010), a arquitetura bidimensional de (BECK, 2008) foi modificada para um modelo unidimensional com uma rede global de conexão. Neste projeto de dissertação trabalhamos com o modelo unidimensional, porém a rede global será decomposta em cinco redes como ilustra a Figura 2.5. A arquitetura é composta pelo contexto de entrada, contexto de saída, conjunto de unidades e as

redes de interconexão. Durante a execução, os valores dos registradores são carregados no contexto de entrada. Através das redes de registradores/constantes estesvalores são roteados para as unidades funcionais. Os dados entre as unidades são roteados pelas redes das unidades. Finalmente, os valores a serem escritos ao final da execução da configuração são enviados para o contexto de saída pela rede de saída. O contexto de saída é transferido para os registradores do processador, posteriormente.

O arranjo reconfigurável é composto por um conjunto de unidades funcionais, cinco redes de interconexão e contexto de entrada e saída.

Na arquitetura unidimensional de (LAURE, 2010) são usadas duas redes, sendo uma rede de entrada e outra de saída. A rede de entrada tem $2N$ sinais pois cada uma das N unidades recebe dois sinais, um para cada operando. A arquitetura proposta neste trabalho usa quatro redes para entrada, mas cada rede tem N sinais. Ambas usam uma rede de saída. A principio tem o dobro do custo. A seguir iremos fazer uma análise mais detalhada.

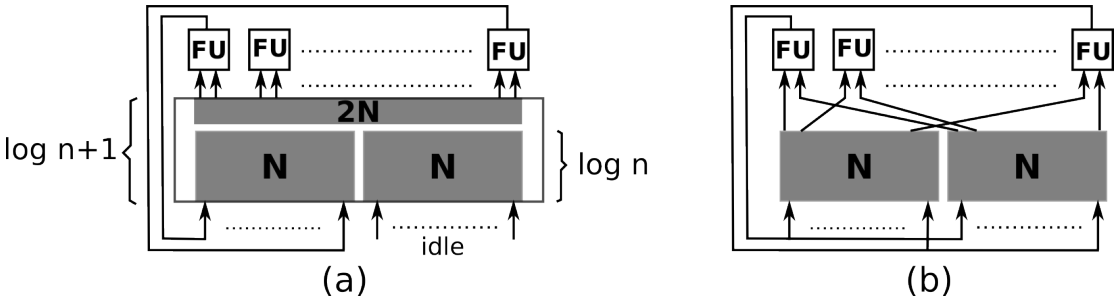


Figura 4.3. (a) Uma MIN conectando todas as Unidades Funcionais (FUs) (b) Uso de uma MIN por operando

Primeiro, na arquitetura com uma única rede é necessário avaliar duas conexões em paralelo, portanto temos que ter duas logicas de controle para cada comutador. Além disso, para o roteamento de $2N$ sinais é necessário um decodificador de prioridade com $2N$ sinais o que aumenta significativamente o atraso do decodificador, que não é linear. Ou seja, uma arquitetura com quatro redes terá o dobro de comutadores, porém o mesmo custo para a lógica de roteamento e um atraso menor pois usa um decodificador de prioridade com N sinais. O fato de ter o dobro de comutadores aumenta o custo mas também melhora a tolerância a falhas. O custo dos comutadores pode ser reduzido se usarmos também uma rede menor para a conexão dos registradores e constantes.Podemos observar uma MIN para interligar as

UFs Figura 4.3 (a), onde apenas metade das entradas será utilizada, nos propomos a utilizar uma rede separada para cada entrada FU como mostrado na Figura 4.3 (b).

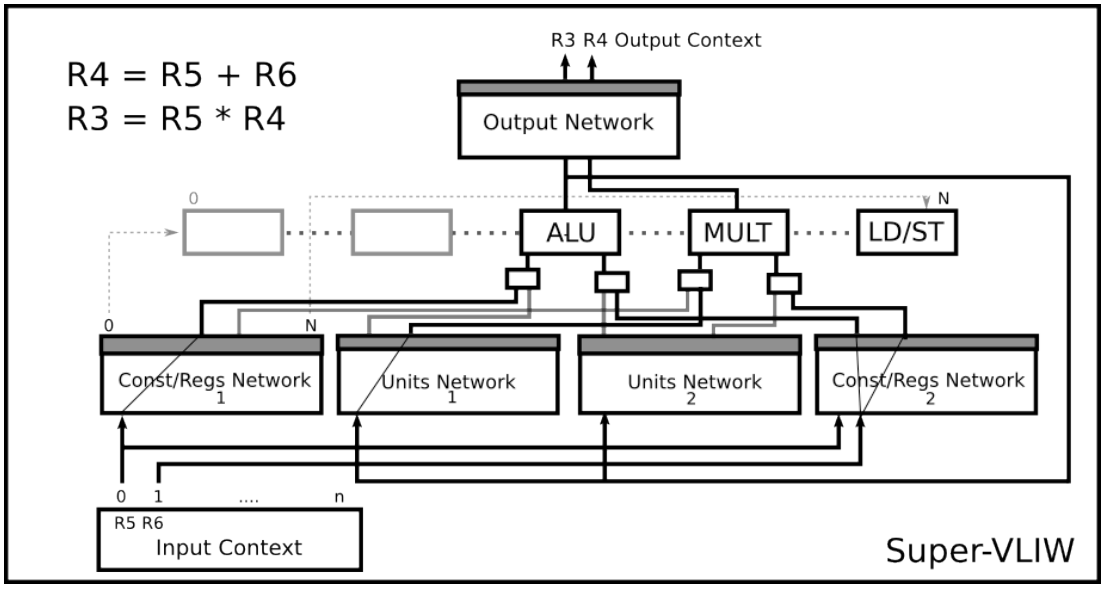


Figura 4.4. Unidades Reconfiguráveis e Redes de Interconexão

Para exemplificar o uso das redes que compõem o arranjo reconfigurável descritas acima, duas instruções foram mapeadas na Figura 4.4. A instrução $R_4 = R_5 + R_6$ usa a rede de registros/imediatos da esquerda e de operandos da direita. Como a segunda instrução $R_3 = R_5 * R_4$ depende da anterior, a rede de unidades é utilizada para rotear o resultado da primeira instrução realizando o encaminhamento (*forward connection*). Além disso, como o operador R_5 é o mesmo da instrução anterior, uma conexão do tipo *multicast* (uma origem e vários destinos) será utilizada na rede de operandos da direita. Por fim as saídas das unidades funcionais devem ser conectadas ao contexto de saída, através da quinta rede MIN da arquitetura.

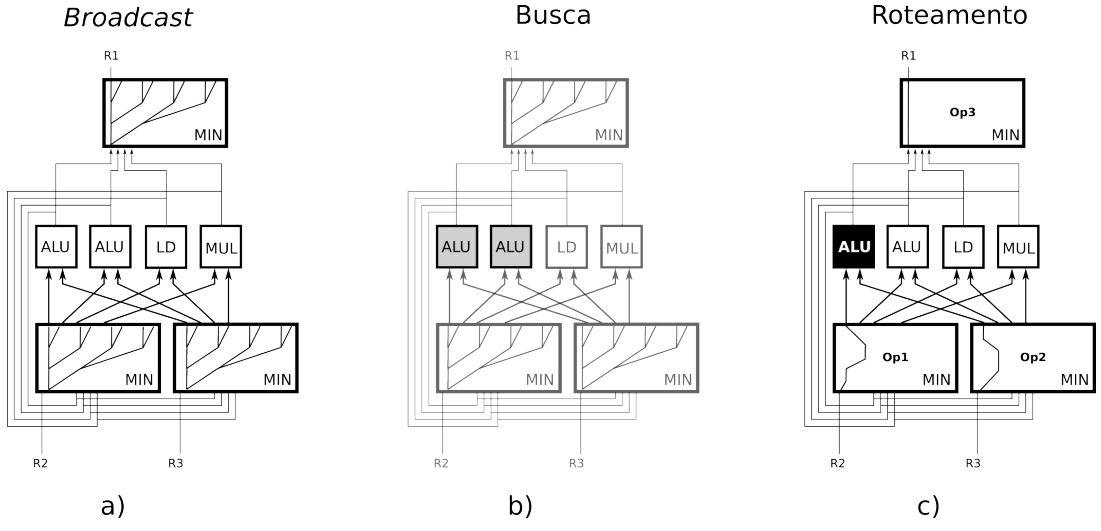


Figura 4.5. Um exemplo de alocação e roteamento de instrução

No exemplo anterior a rede de saída conectará R_3 e R_4 ao contexto de saída.

Quando uma operação WAW (*write-after-write*) ocorrer dentro de um bloco de instruções, apenas a última operação de escrita será roteada pela rede de saída.

Durante a fase de detecção o tradutor binário verificará os operandos da instrução. Considerando uma instrução com três operandos *Op3 = Op1op Op2*, onde *Op1* e *Op2* são os operando da esquerda e direita respectivamente, *Op3* é o operador destino e *op* é a operação realizada pela UF. Quando o tradutor binário seleciona um ainstrução para ser executada no arranjo reconfigurável, o primeiro passo será o envio do sinal de broadcast em paralelo através das redes da esquerda e da direita Figura 4.5 (a). As redes podem ser utilizadas para enviar sinal de registradores/imediatos ou para unidades. Ao mesmo tempo é enviado um broadcast na MIN de saída a partir da saída correspondente ao operador *Op3*.

Para facilitar a explicação, vamos ilustrar com a Figura 4.5 que mostra o conceito com apenas duas redes de entrada. Supondo que queremos alocar a instrução *Add R1,R2,R3*. Cada broadcast encontrará as UFs alcançáveis para cada operando. No nosso exemplo, os operandos 1 e 2 alcançaram as *ALUs* e multiplicadores Figura 4.5 (a). Contudo, apenas a *ALU* é candidata a executar a instrução *Add*, Figura 4.5 (b). Finalmente uma operação *bitwiseAND*, dos três vetores resultantes das saídas alcançadas em cada rede é realizada e o decodificador de prioridade irá selecionar dentre os resultados desta operação a primeira UF capaz de executar a instrução, no caso do exemplo a primeira *ALU* como ilustrado na Figura 4.5 (c). Além disso quando o número de registros é menor que o número de unidades, um registro pode ser conectado a mais de uma posição da rede de saída, o que melhora a capacidade de roteamento do contexto de saída.

4.3 Falhas no Contexto do Super-VLIW

Na Seção 4.1 apresentamos lógica por trás dos mecanismos de tolerância a falhas da rede multiestágio Omega. Nesta seção apresentaremos o comportamento da Unidade Funcional Reconfigurável na presença de falhas. Antes de discutirmos o comportamento precisamos primeiro assumir que um comutador é considerado interno se este não se conectar diretamente a uma entrada ou saída da rede, como ilustrado na Figura 4.6.

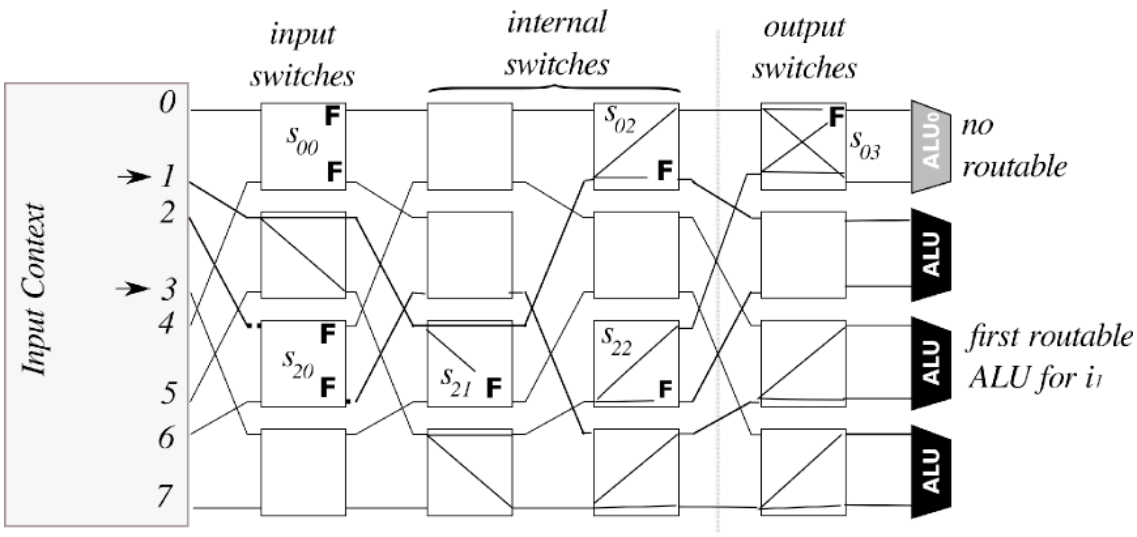


Figura 4.6. Tolerância a falhas durante a busca e alocação dinâmica

Quando uma simples falha afeta um fio, três casos distintos poderão ocorrer. Primeiro, uma falha em um fio de comutador interno vai afetar múltiplos caminhos. Contudo, se a MIN possuir estágios extras, outros caminhos alternativos estarão disponíveis. Por exemplo, a conexão da entrada 1 é afetada por falhas presentes nos comutadores $\boldsymbol{c}_{02}$, $\boldsymbol{c}_{21}$ e $\boldsymbol{c}_{22}$. Todos os casos estão ilustrados na Figura 4.6., contudo o roteamento pode ser feito ao selecionar outros caminhos.

No segundo caso, um fio com falha ocorre no comutador de saída, por exemplo, no comutador $\boldsymbol{c}_{03}$ e na unidade ALU_0. Portanto esta saída estará inacessível. Como as unidades são alocadas dinamicamente o algoritmo de alocação vai tratar estas unidades como indisponíveis e as evitará, buscando pela primeira unidade roteável que esteja disponível. Finalmente, quando um fio de um comutador do primeiro estágio falhar, como por exemplo, o comutador $\boldsymbol{c}_{00}$ e $\boldsymbol{c}_{20}$, ilustrado na Figura 4.6, apenas uma das duas entradas estará disponível e se ambos os fios falharem, as duas entradas associadas serão desconectadas. Neste caso, uma abordagem de renomeação de registro é utilizada para superar as falhas da entrada. Isto é implementado selecionando-se o primeiro registro disponível, como exemplificado na Figura 4.6, onde a primeira entrada disponível na rede é Ent_1, seguida pela Ent_3, uma vez que as entradas Ent_0 e Ent_2 estão inacessíveis.

A renomeação de registros é realizada durante a fase de detecção. Se um registro já foi alocado, o seu índice no contexto de entrada é obtido a partir de um arranjo indexado pelo número do registro. Por exemplo, a instrução $AddR_1,R_0,R_2$ é detectada. R_0 já está alocado na entrada 1 por uma instrução anterior. Portanto, o

mesmo índice será utilizado por R_0. No entanto, se R_2 não foi definido ainda e a primeira entrada livre e saudável é a entrada 5, o R_3 será mapeado sobre esta entrada.

Como mencionado anteriormente, é importante ressaltar que este trabalho considera apenas as falhas permanentes geradas durante o processo de fabricação. Assim, as informações sobre as unidades com defeito são geradas antes da unidade de tradução binária iniciar, através de técnicas de teste clássicas. Além disso, nenhuma modificação nem sobre o algoritmo de tradução binária nem sobre o posicionamento dinâmico e roteamento são necessários, e a solução para fornecer tolerância a falhas é transparente.

5 RESULTADOS

Nesta seção iremos apresentar o ambiente de simulação, critérios e métricas utilizadas para realização dos testes, avaliação das aplicações de teste e considerações sobre os resultados obtidos do ponto de vista de desempenho e área.

5.1 Ambiente de Teste

Para avaliarmos a arquitetura proposta neste trabalho executamos um conjunto de aplicações testes conhecido como MiBench (GUTHAUS, 2001). Originalmente estas aplicações foram executadas na ferramenta de simulação ArchC (RIGO, 2004). Da execução foram gerados arquivos contendo o *trace* de cada programa. Um simulador implementado na linguagem Java baseado nas ferramentas desenvolvidas por (LAURE,2010) e (RUTZIG,2009) foi utilizado para executar os *traces* e avaliar o desempenho da arquitetura tendo como métrica o número de ciclos.

Dentre os *traces* gerados pelo ArchC a partir das aplicações do MiBench o *benchmarkString* foi o que apresentou o menor número de instruções, um total de 279664, sendo o Dji o *trace* com o maior número de instruções, no total 59352899 instruções do MIPS R3000.

O tempo de execução dos testes varia com o *hardware* do computador e principalmente com o tamanho do arquivo de rastreio de execução da aplicação (*trace)*. Portanto, o tempo de leitura dos arquivos é o principal gargalo do simulador para a execução dos testes, o tamanho dos arquivos varia de 5 *MB* até 926 *MB*.

Os parâmetros da arquitetura também influenciam diretamente o tempo dos testes. Aumentar o número de estágios eleva o tempo de execução, mas nada muito significativo, por outro lado elevar o número de falhas diminui consideravelmente o tempo de execução. Este comportamento deve-se ao maior caminho a ser percorrido quando se insere novos estágios e por um encerramento do processo de roteamento uma vez que o número de insucessos aumenta de forma diretamente proporcional às falhas.

Para permitir uma análise estatística satisfatória as falhas foram distribuídas aleatoriamente pela rede e unidades a cada repetição do teste. A quantidade de falhas injetadas variava de acordo com a escala do teste. Para fins comparativos executamos testes com 0.1, 1, 5, 10, 15 e 20% da área total da arquitetura reconfigurável com falhas. Cada um destes percentuais foi aplicado a uma configuração com diferentes valores para os estágios extras, sendo os valores observados 0, 4 e 7 estágios extras. Cada combinação (estágio extra + percentual de falha) foi executada num total de 30 repetições.

Apenas a título de ilustração temos que o tempo médio de execução do Dji foi de 44,42 minutos, considerando todas as repetições para cada percentual de falha testado, temos um total de 9,25 dias de simulação para apenas um (1) *benchmark* em uma única configuração, no caso em questão este valor corresponde a configuração de rede multiestágio Omega com $N = 128$, $k = 0$ de estágios extras 90 unidades aritméticas, 34 unidades de *load/store* e 4 multiplicadores. Vale lembrar que o Dji compreende o pior dos casos, mas existiam no total 16 aplicações teste, portanto o período de tempo necessário para executar todas as aplicações testes e configurações se estendeu por semanas.

Para conseguir executar todas as repetições em tempo hábil foi utilizado um laboratório com 24 máquinas com processador Intel Core Duo de 2.4 Mhz, 2GB de memória RAM e disco rígido de 5.400 *rpm*, no Linux Ubuntu 9.0. Vale ressaltar que o sistema operacional não foi relevante para o tempo das simulações, estas apresentavam tempo médio semelhante quando executadas no Linux, Windows ou Mac OS X.

Para otimizar o tempo os *benchmarks* foram agrupados pelo tempo médio de execução em grupos de 4 de forma que o tempo total de execução de 5 repetições para cada aplicação teste não superasse 10h, tempo que o laboratório ficava disponível no período noturno. Sendo assim, cada máquina executava as repetições de 4 aplicações diferentes, totalizando 4 máquinas para executar uma configuração. Isto permitia obter resultados parciais para várias configurações distintas, algumas delas não foram consideradas nos resultados finais, pois foram utilizadas para testes de validação dos resultados. Também realizamos testes de saturação para conhecer o limite de tolerância a falhas da arquitetura e percebemos que na presença de 20% de falhas ou mais a arquitetura não escala satisfatoriamente.Outros testes foram

realizados para definirmos a quantidade correta de unidades funcionais a serem utilizadas.

O algoritmo de geração de falhas leva em consideração o peso de cada unidade integrante da arquitetura, dos *muxes* das redes multiestágio aos multiplicadores do MIPS. Como os *muxes* correspondiam a menor unidade assumimos peso 1 para este.Proporcionalmente, o multiplicador que é a maior unidade na arquitetura assumiu o peso 111. Conhecendo os pesos de cada unidade e a quantidade de cada uma obtemos a área ocupada por elas na arquitetura, multiplicando os pesos pela quantidade. Por fim, é gerado um número aleatório dentro de um intervalo que corresponde a área total de todas unidades. Se o número gerado corresponder a qualquer parte de uma unidade esta é considerada falha como um todo. Um novo número é gerado sucessivamente até atingir a quantidade de falhas equivalente ao percentual testado, vale ressaltar que foi permitida repetição de unidades.

As falhas são geradas para as redes e para as unidades em separado. O motivo desta abordagem é o fato da granularidade das unidades das redes (*muxes*) ser muito pequena quando comparada a das unidades funcionais. A diferença de área das unidades diminui as chances probabilísticas das falhas incidirem nos *muxes* da rede, portanto não distribuindo bem as falhas injetadas na arquitetura.

Cada falha gerada é armazenada em matrizes e arranjos que correspondem às redes e ao conjunto de unidades.Esta informação é lida pelo simulador antes da execução do teste e a partir deste momento as unidades indicadas são tratadas como unidades com falhas durante o processo de roteamento.

5.2 Estudo de Caso

Como estudo de caso foram analisados três processadores VLIW: o VLIW simples de 8 unidades, um VLIW dinâmico também com 8 unidades (denominado neste trabalho por VLIW8) e a arquitetura proposta com reconfiguração dinâmica, tradução binária e tolerância a falhas denominado Super-VLIW. A arquitetura VLIW8 também usa reconfiguração dinâmica e tradução binária, e será usada para avaliar o desempenho de uma arquitetura com poucas unidades. Elas diferem essencialmente no número de unidades funcionais e na rede de interconexão, sendo que no caso da Super-VLIW a rede de interconexão provê um mecanismo de tolerância a falhas. Além disso, a Super-VLIW pode ter 128 unidades.

O VLIW simples é composto por 4 unidades aritméticas (ALUs), 3 de carregamento/persistência (*Load/Store)* e 1 de multiplicação (Mult)e não suporta nenhum tipo de dependência de dados, em outras palavras, ele funciona exatamente como uma máquina VLIW tradicional como apresentado na Figura 5.1 (a).

Um dos problemas da arquitetura VLIW tradicional é a necessidade de compilar os códigos fontes ou binários para instruções VLIW, eliminando a dependência de dados. O exemplar de código binário apresentado na Figura 5.1 (a), equivale a três instruções VLIW e a execução destas necessitaria três ciclos de processamento para serem executadas.

O processador dinâmico VLIW8 também é composto por 3 unidades funcionais, entretanto cada unidade possui dois multiplexadores completos e podem conectar a qualquer contexto de entrada ou unidade funcional através de uma rede não bloqueante que não necessita de algoritmo de roteamento das conexões. Todas as unidades são capazes de receber valores de registradores ou repasse de dados.Além disso, todos os padrões de interconexão são possíveis uma vez que a rede de interconexão é feita por multiplexadores. Na arquitetura VLIW8 o bloco de instruções é construído dinamicamente e cada bloco pode conter 4 unidades de *Alus*, 3 *Load*/*Store* e 1 *Mult*. No melhor dos casos cada unidade funcional trabalhará em paralelo. Isto acontecerá quando não existir nenhuma dependência de dados e no pior dos casos eles trabalharão de forma sequencial.

Para efeito de comparação a sequência MIPS apresentada na Figura 5.1 (a) seria mapeada para três palavras (*words*) do VLIW simples, pois existe dependência de dados, em contrapartida apenas uma palavra (*word*) dinâmica do Super-VLIW seria suficiente para armazenar a mesma sequência MIPS. A palavra do Super-VLIW é capaz de armazenar a unidade e a informação de interconexão. Como foi mencionado anteriormente, esta configuração é dinâmica e construída em tempo de execução.

O Super-VLIW, que é composto por um total de 128 unidades funcionais (90 *Alus*, 34 *Load/Store* e 4 *Mult*) que são conectadas através de uma rede MIN. A Figura 4.1 apresenta um desenho esquemático da arquitetura.

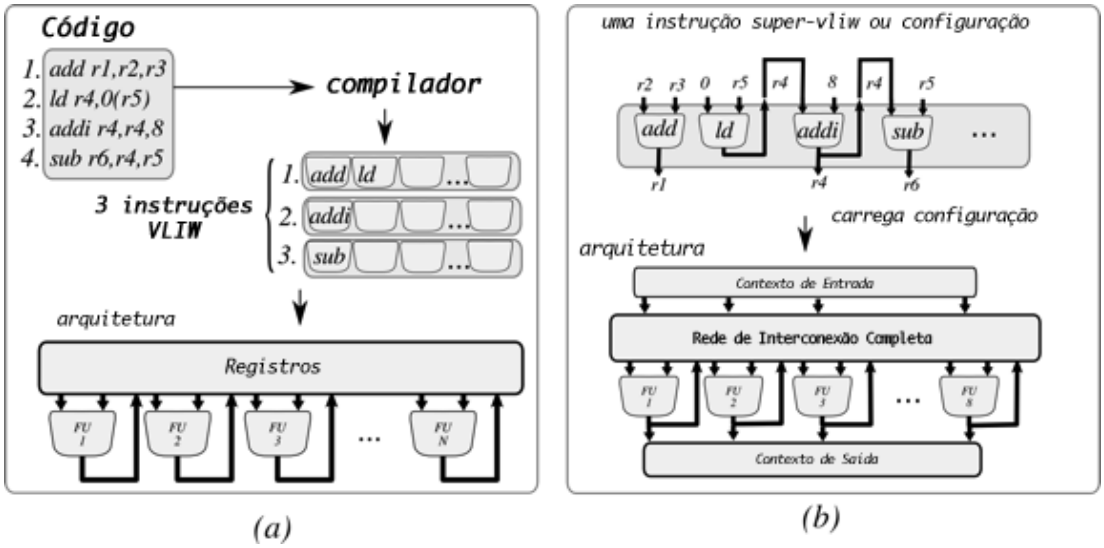


Figura 5.1. Uma sequência de quatro instruções mapeadas em um VLIW simples (a) e em um VLIW8 (b)

5.3 Benchmarks

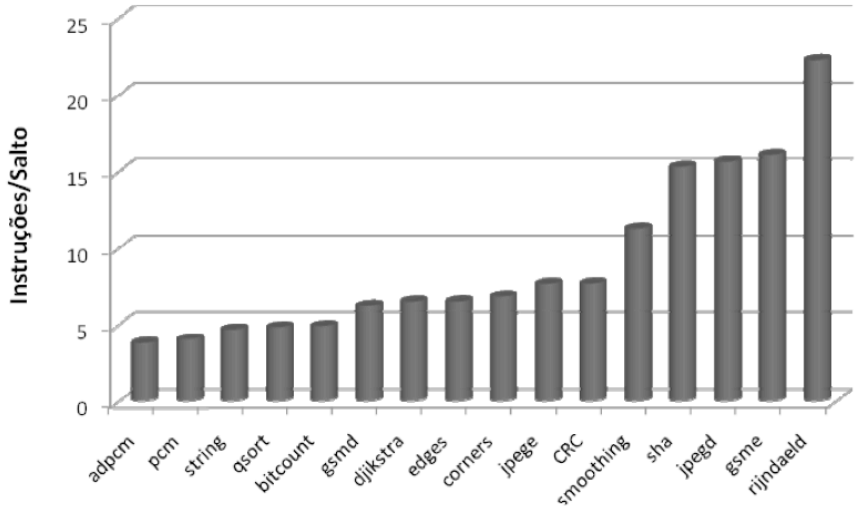
Nesta seção iremos apresentar uma caracterização dos benchmarks usados para avaliar se a arquitetura proposta foi capaz de prover ganho de performance mesmo na existência de um número elevado de falhas.

5.3.1 Benchmarks

Para aferir os ganhos em aceleração utilizamos um sub-conjunto dos algoritmos de teste do MiBench (GUTHAUS, 2001). Dada a natureza heterogênea das aplicações que um computador executa, este conjunto de testes foi escolhido pois seus algoritmos abrangem diversos comportamentos. Em virtude disto os 16 benchmarks listados a seguir foram analisados: *Adpcm, Bitcount, CRC32, Dijkstra, Jpeg Encode/Decoder, Gsm Coder/Decoder, Pcm, Quicksort, Rijndael Decoder, Sha, String, Susan Edges/Smoothing/Corners*.

Inicialmente os algoritmos foram classificados em *controlflow* ou *dataflow*, tomando como base os números de instruções executadas por desvio. Como pode ser observado na Figura 5.2, o *Adpcm* é o algoritmo mais *controlflow* (possui um número elevado de desvios por programa) enquanto o *Rijndael Decoder* é o extremo oposto, ou seja, mais *dataflow*. Esta análise é importante para arquiteturas reconfiguráveis, pois o espaço para se explorar o paralelismo da aplicação é proporcional ao número de instruções por bloco base. Bloco base pode ser definido como o conjunto de instruções entre duas instruções de

desvio. Quanto maior o número de instruções por bloco base, menor será o número de rotas e área consumida, por sua vez, um número elevado de desvios e consequentemente menor número de instruções por bloco base, significa que existirão várias rotas alternativas que podem ser tomadas, aumentando assim o tempo de execução e a área consumida por uma dada configuração, quando esta for implementada em lógica reconfigurável.



Fonte: Adaptado de (RUTZIG; CARRO, 2008, p. 42)

Figura 5.2. Correlação entre instruções de dados por instruções de desvio no conjunto de teste do MiBench

5.3.2 Tipo de instruções

Para definirmos a melhor configuração para o ambiente de simulação precisávamos conhecer melhor a natureza dos programas executados. Desta maneira analisamos os *benchmarks* quanto ao tipo e predominância das instruções.

A Figura 5.3 apresenta algumas das instruções do conjunto de instruções MIPS incluindo as de maior ocorrência nos *benchmarks* analisados.

Na arquitetura de simulação utilizada, as instruções eram decodificadas pelo tipo, sendo eles: *load, store, aritmetic, mult* e *branch*. Para cada tipo existe uma unidade funcional específica capaz de processá-lo, exceto para o tipo desvio que corresponde às instruções de encerramento do bloco base. O balanceamento das unidades funcionais na arquitetura de simulação permite ajustes finos para se obter os melhores resultados, uma vez que um tipo de unidade é mais relevante em um contexto geral que outras.

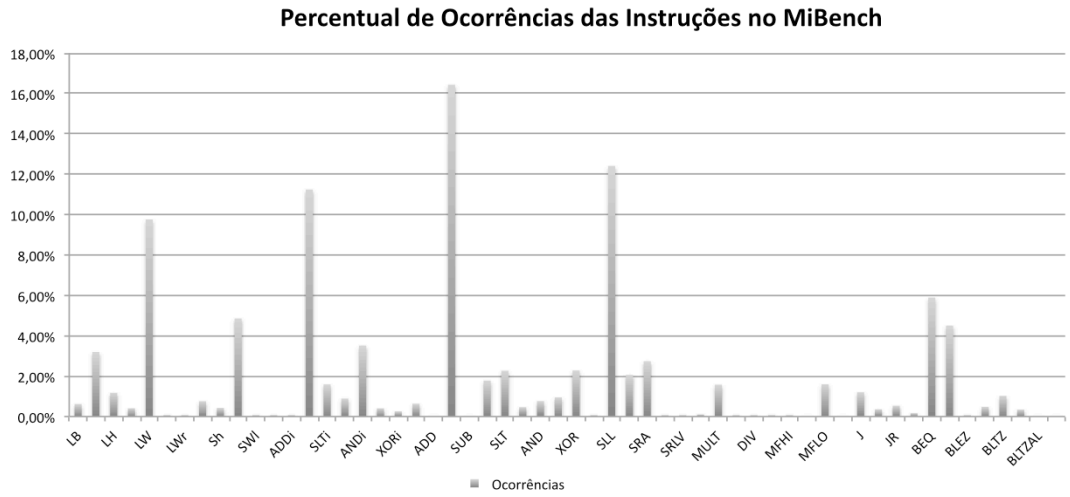


Figura 5.3. Gráfico de ocorrências de um sub-conjunto de instruções do assembly MIPS nos benchmarks do MiBench

Outro fator relevante para os resultados diz respeito à área que cada unidade funcional ocupa na arquitetura. Uma vez que os defeitos do processo de fabricação em nanoescala são injetados de forma aleatória, os mesmos podem atingir regiões correspondentes às unidades funcionais. Se a área de uma unidade funcional for parcialmente atingida, como por exemplo um dos fios de entrada da unidade toda a unidade será inutilizada.

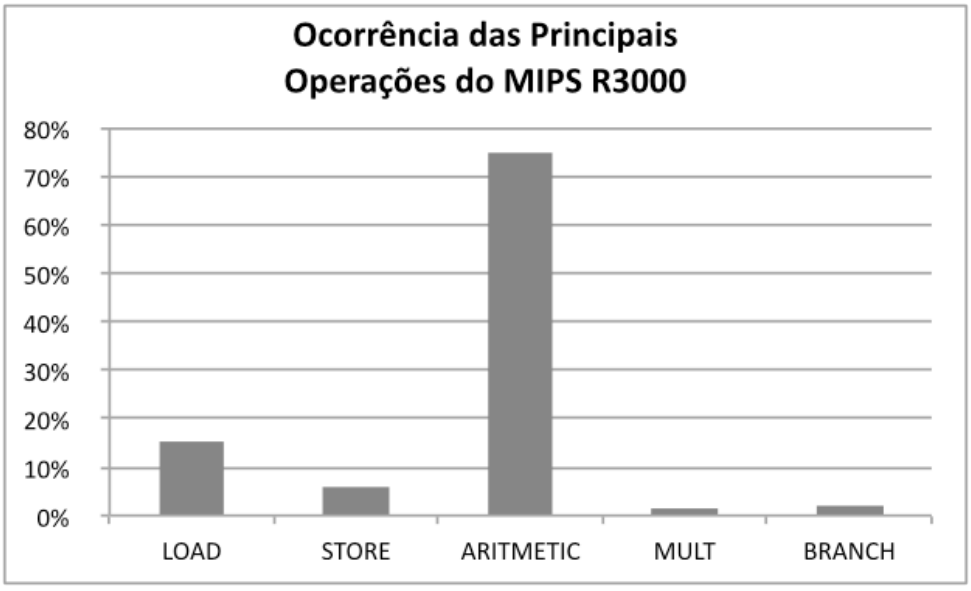


Figura 5.4. Gráfico de distribuição das instruções do assembly MIPS nos benchmarks do MiBench agrupadas por tipo

Na figura acima é fácil constatar que conjunto de testes do MiBench possui uma ocorrência maior de instruções do tipo aritmética, seguidas pelas instruções do

tipo *Load* e *Store* respectivamente. Consequentemente existirá uma demanda maior por unidades análogas na arquitetura.

5.3.3 Distribuição dos blocos base

A forma como as instruções se distribuem ao longo do programa é fator determinante para o melhor aproveitamento do mecanismo de tradução binária e exploração do paralelismo da aplicação pela arquitetura.

Cada *benchmark* analisado neste trabalho possui seu próprio comportamento. Este comportamento deve-se à distribuição, recorrência e tamanho dos blocos base. Por exemplo, a Figura 5.5 (b) apresenta o histograma do tempo de execução em função do tamanho do bloco para o *benchmarkPulse Code Modulation* (PCM). O maior bloco tem tamanho 12 e é responsável por 15% do tempo total de execução. No entanto, a maior parte do tempo de execução, cerca de 70% é gasta com blocos pequenos (menores que 7), como mostrado na Figura 5.5 (a). A fração de tempo exibida na Figura 5.5 é relativaà fase de execução.

Como mencionado no capítulo 3, o primeiro bloco é executado no MIPS e paralelamente o tradutor binário cria o bloco reconfigurável durante a fase de detecção. Na fase de execução o bloco é recuperado da cache reconfigurável e em seguida executado na unidade reconfigurável. Isso explica por que o tempo total acumulado apresentado na Figura 5.5 (a) é inferior a 100%, uma vez que somente a fase de execução está sendo levada em conta. Como o PCM é dominado por blocos pequenos, ou seja, mais orientado a controle (*controlflow*), o desempenho dele na arquitetura VLIW8 com apenas 8 unidades funcionais, foi bem semelhante ao do Super-VLIW com 128 unidades, Figura 5.2.

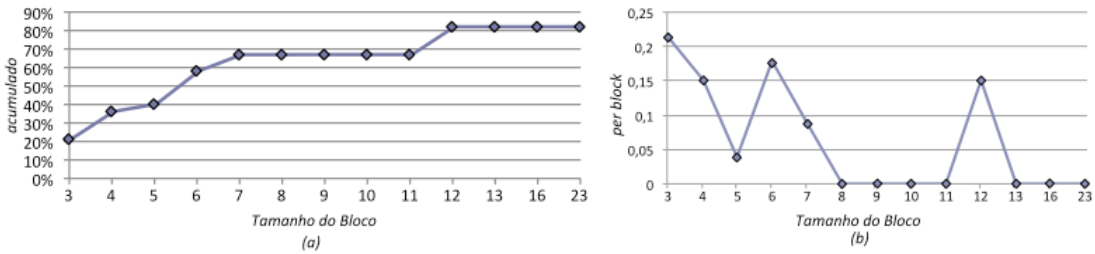


Figura 5.5. Comportamento do tamanho do bloco no PCM (a) Histograma acumulado pelo tempo de execução (b) Histograma tempo de execução por tamanho do bloco

Em contrapartida, existem os benchmarks mais orientados a dados (*dataflow*), como por exemplo, o *Susan Edges* (um algoritmo de processamento de imagens para

detecção de bordas), cujo comportamento dos blocos é apresentado na Figura 5.6. A maior parte do tempo de execução deste, é dominada por blocos com mais de 8 instruções (cerca de 60%). Além disso, se considerarmos a execução especulativa usando dois níveis de predição de desvio, o tamanho do bloco será ainda maior. Para este caso em questão, existem blocos bem grandes contendo até 169 instruções, ocupando 15% do tempo de execução. Vale ressaltar que blocos desta dimensão não podem ser capturados por completo mesmo pela arquitetura Super-VLIW com a configuração utilizada neste trabalho, uma vez que esta trabalha com um máximo de 128 unidades funcionais.

Portanto, dependendo do comportamento de cada aplicação, o Super-VLIW pode conseguir uma aceleração significantemente superior ao VLIW8 e em alguns casos a aceleração do Super-VLIW poderá ser consideravelmente maior que a obtida pelo VLIW simples.

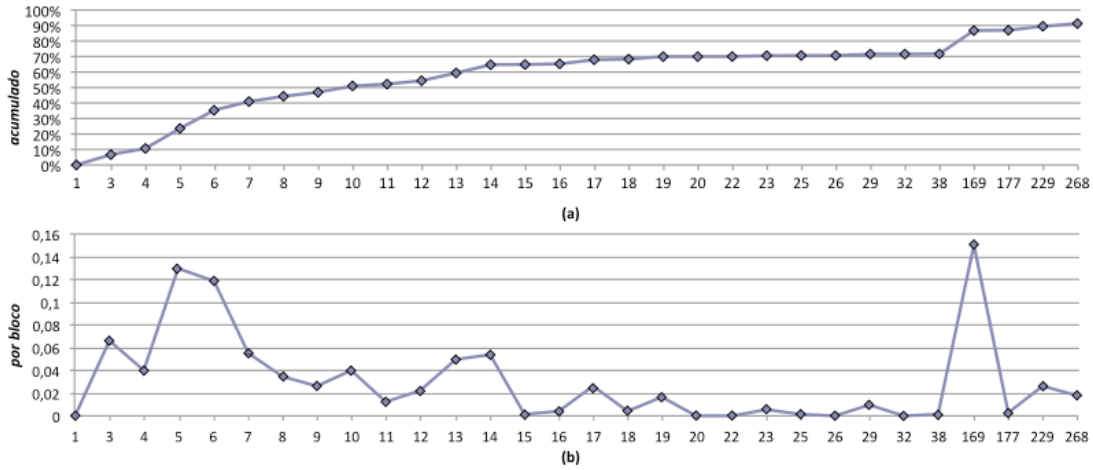


Figura 5.6. Comportamento do tamanho do bloco no Susan Edges (a) Histograma acumulado pelo tempo de execução (b) Histograma tempo de execução por tamanho do bloco

5.4 Desempenho

Podemos comparar o desempenho de cada uma das arquiteturas VLIW, VLIW8 dinâmico e Super-VLIW observando o gráfico da Figura 5.7.

Em um cenário ideal sem a presença de falhas, como o apresentado na Figura 5.7 o Super-VLIW seria capaz de atingir uma aceleração média de 2,84 vezes nos *benchmarks* do conjunto de teste MiBench, em comparação o VLIW8 e o VLIW simples obtiveram aceleração média de 1,7 e 1,17 respectivamente. Se considerarmos os sete testes com os melhores resultados de aceleração individual, o resultado médio do Super-VLIW sobe para 3,4 contra 1,64 do VLIW8. Desta forma podemos concluir

que para aplicações mais *dataflow* o Super-VLIW pode ser até 2 vezes mais rápido que o VLIW8. Essa diferença dá-se pelo número superior de unidades funcionais necessárias para capturar os blocos maiores comuns em aplicações *dataflow*.

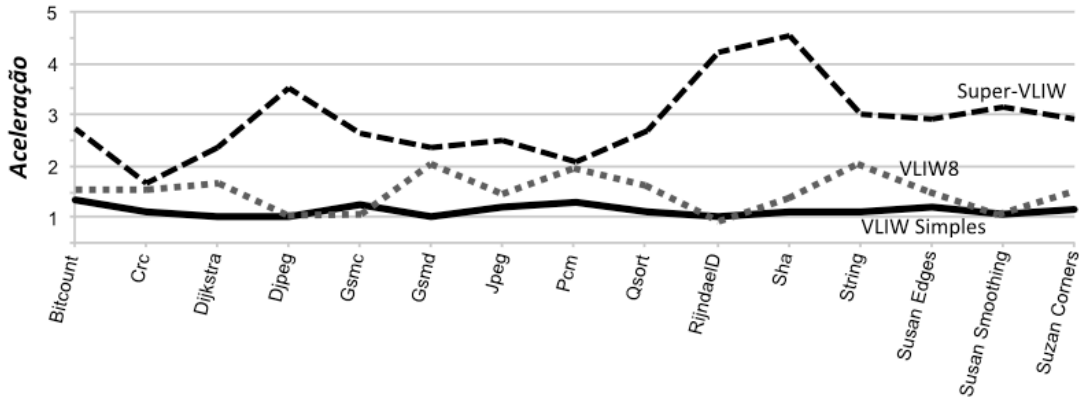


Figura 5.7. Desempenho do VLIW Simples, VLIW8 e Super-VLIW executando os benchmarks do MiBench

Para conduzir as análises deste trabalho e avaliar a degradação de desempenho apresentada pela arquitetura Super-VLIW sob as diferentes taxas de falhas, consideramos como valor de referência o desempenho do processador MIPS R3000 com pipeline de 5 estágios. Logo todos os resultados de aceleração são relativos ao MIPS. Neste contexto, se a arquitetura gera um fator de aceleração de 2, isto significa que a arquitetura proposta é duas vezes mais rápida que o processador MIPS executando sozinho.

Em uma avaliação superficial o Super-VLIW pode não parecer tão vantajoso, visto que em termos de *hardware* ele é mais complexo e caro quando comparado ao VLIW8. Esta ideia pode ser reforçada pelo fato que este último seja capaz de apresentar uma média de aceleração de 1,7 vezes, para os *benchmarks* do MiBench. Contudo, arquiteturas semelhantes ao VLIW8 não são realistas, uma vez que elas não provêm mecanismos de tolerância a falhas. Se uma falha ocorrer em qualquer parte dos multiplexadores ou unidades funcionais, todo o recurso será invalidado.

Em um cenário mais realista, onde falhas existem e devem ser consideradas, a melhor solução apresentada é a proposta pela arquitetura Super-VLIW, uma vez que ela garante a execução da aplicação mesmo com várias unidades funcionais ou de interconexão defeituosas.

Para analisar a degradação de desempenho apresentada pelo Super-VLIW na presença de falhas, o mesmo conjunto de *benchmarks* foi executando considerando-se diferentes casos. A Figura 5.8 apresenta uma análise da aceleração em função da taxa de falhas. O gráfico apresenta a aceleração mais significativa (ausência de falhas) e cinco taxas diferentes de falhas.

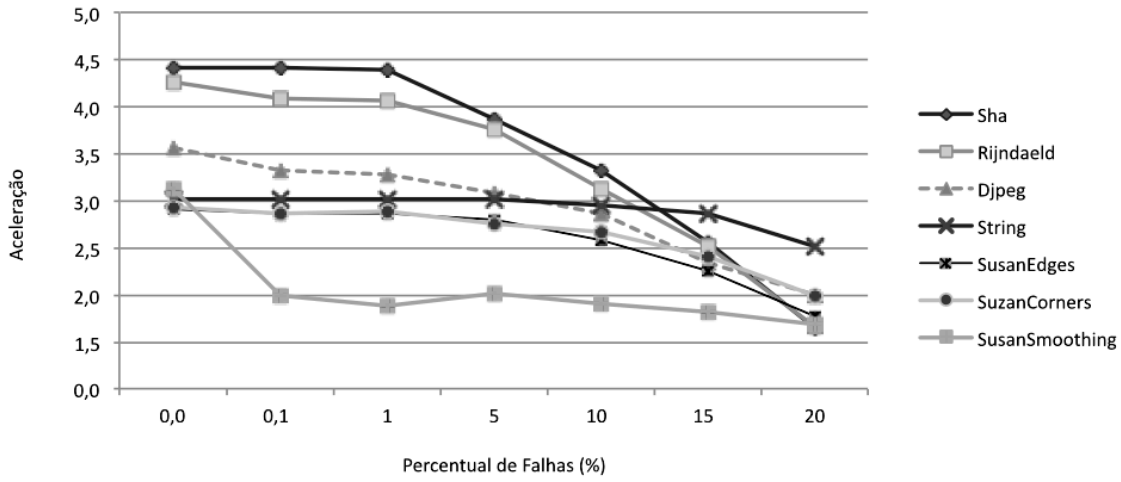


Figura 5.8. Queda de aceleração na presença de falhas para os *benchmarks* mais significantes do MiBench no Super-VLIW

Como podemos observar na Figura 5.8, o Super-VLIW não apenas continuou funcionando na presença de 20%, como também foi capaz de acelerar a execução da aplicação em relação ao MIPS trabalhando sozinho.

De acordo com os resultados apresentados, a aceleração média foi em torno de 2,24 para 15% de falhas e de 1,90 para em torno de 20%, durante a execução de todos os *benchmarks* do conjunto de testes do MiBench. Portanto, o Super-VLIW pode ser escalado para novas tecnologias de fabricação em nanoescala, mesmo na presença de elevados índices de falhas. A configuração utilizada no teste da Figura 5.8 foi uma rede Omega N = 128 e $K = 7$.

5.5 Área

Como salientado anteriormente a área é um fator crucial para o sucesso de uma arquitetura computacional. Uma arquitetura bem dimensionada pode significar dentre outras coisas, melhor desempenho e custo de fabricação menor. Além disso, tolerar as novas escalas de fabricação de circuitos integrados pode significar vantagem competitiva no mercado de dispositivos móveis, onde a tendência dos dispositivos é sempre de buscar o design mais compacto. A Figura 5.9 apresenta uma relação do

dimensionamento das unidades computacionais que compõem o Super-VLIW e o aumento do número de falhas.

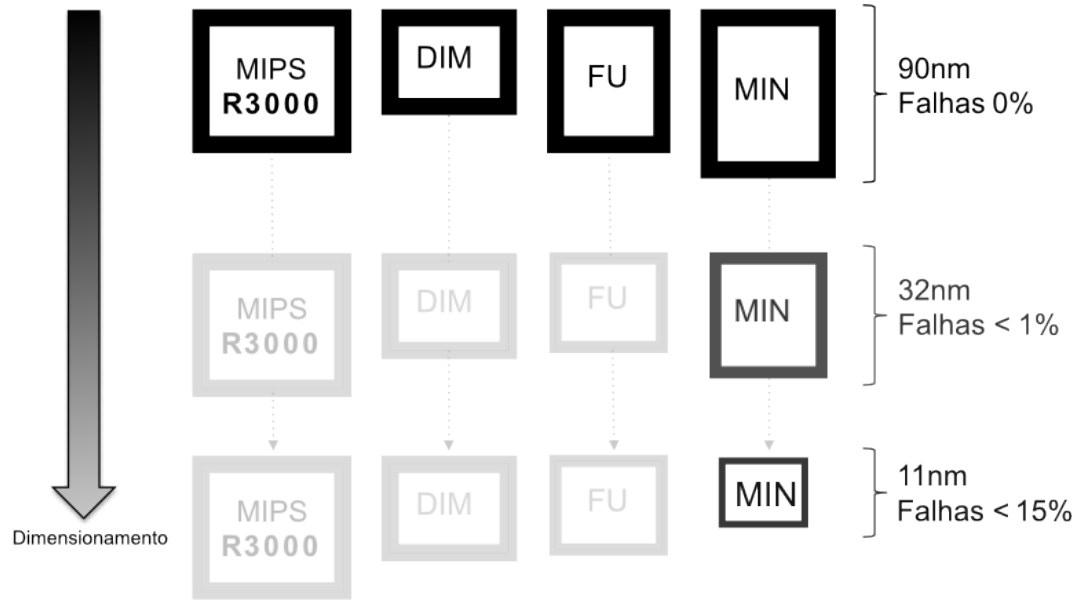


Figura 5.9. Relação de falhas com o dimensionamento da arquitetura Super-VLIW

A Tabela 5.9 apresenta uma análise da área em função do dimensionamento da tecnologia. Uma vez que o MIPS e a unidade de tradução binária são fundamentais para o funcionamento correto do sistema, optamos por não escaloná-los; manter a área deles aumenta a confiabilidade do sistema, já que em escalas maiores o percentual de defeitos inseridos é virtualmente zero. Portanto, o tamanho de ambas as unidades na tecnologia de 90*nm* vai ser de 0,4*mm*. Além disso, a área de VLIW8 será 0,6*mm*, que é 3 vezes maior do que um único MIPS e o desempenho é de apenas 1,7 vezes melhor.

Em contrapartida, o Super-VLIW pode ser dimensionado para as novas tecnologias que apresentam taxas de falhas mais elevadas. Uma vez que as unidades funcionais são menos tolerantes a falhas que a rede, estas podem ser fabricadas em uma escala maior que a da rede. Por exemplo, o arranjo de unidades funcionais por ser dimensionado para 0,16mm^2na tecnologia de 32*nm*, e a rede MIN pode ser dimensionada para 0,03mm^2 a 11*nm*. O desempenho da MIN irá degradar a 11*nm* devido a elevada taxa de falhas. Contudo como demonstrado na Figura 5.8, a aceleração média da arquitetura para um percentual de falhas de até 15% é de 2.24, e a área é aproximadamente 2.4 vezes maior que a de um único MIPS (0.53 para o Super-VLIW e 0.22 para o MIPS sozinho).

Table 5.1. Tamanho estimado do chip de acordo com o dimensionamento da tecnologia

Tecnologia	90nm	32nm	22nm	18nm	11nm
MIPS	**0.22**	0.22	0.22	0.22	0.22
Tradutor Binário	0.12	0.12	0.12	0.12	0.12
128 Unidades Funcionais	1.3	0.16	0.16	0.16	0.16
MIN	1.8	0.23	0.11	0.07	0.03
Área Total do Super-VLIW	3.5	0.73	0.61	0.57	**0.53**

Portanto, a aceleração é proporcional a área extra. Se o percentual de falhas for inferior, a aceleração pode ser melhorada (chegando a 4.5 vezes para alguns *benchmarks* como apresentado na Figura 5.8). Nestes resultados não foi levado em consideração o fato de que o tempo de ciclo pode diminuir à medida que a dimensão da tecnologia diminui.Isto significa que pode-se obter acelerações ainda maiores ao reduzir o Super-VLIW.

6 CONCLUSÕES

Este trabalho demonstrou o potencial de aceleração de uma arquitetura dinamicamente reconfigurável, na execução de aplicações heterogêneas mesmo na presença elevada de falhas. A arquitetura foi denominada pelo termo Super-VLIW e explorou conceitos apresentados em outros trabalhos como na arquitetura reconfigurável proposta por (BECK, 2006). Além de prover tolerância a falhas e aceleração a arquitetura se mostrou viável para ser aplicada a dispositivos embarcados e aderente as novas tecnologias de fabricação em nanoescala.

Uma grande contribuição do trabalho de (BECK, 2008), base para este trabalho, foi mostrar que é viável uma arquitetura reconfigurável com um mecanismo de tradução binária acoplada. Entretanto, a arquitetura base apresentava um alto custo devido ao número excessivo de unidades funcionais e o alto custo das redes de interconexões. Os trabalhos derivados buscavam reduzir o número de unidades (RUTZIG, 2008) e os custos de interconexão (LAURE, 2010) bem como incorporar tolerância a falhas (PEREIRA, 2009). Este trabalho seguiu a mesma linha, dando continuidade aos anteriores.

As principais contribuições foram:

- Redução da área total das unidades funcionais em relação ao trabalho de (BECK, 2006), através da introdução da rede multiestágio Omega de forma análoga a obtida por (LAURE, 2010). Entretanto a lógica de controle, que executa o algoritmo de roteamento em hardware nas redes multiestágio, foi simplificada em relação a abordagem empregada em (LAURE, 2010), ao dividir a estrutura de uma rede global em quatro redes menores de interconexão; Extenção do roteamento dinâmico (FERREIRA *et. al*., 2009) para rede Omega com a inclusão de mecanismo para tolerância a falhas. Esta contribuição foi publicada em (FERREIRA *et al.*, 2010);
- Avaliação das aplicações do conjunto de teste MiBench quanto as características das instruções e distribuição e tamanho dos blocos base; Avaliação de uma arquitetura dinamicamente reconfigurável capaz de prover

ganho de desempenho equivalente a área estendida mesmo na presença de até 15% de falhas.

- Modelos de tolerância a falhas em redes multiestágios podem suportar altas taxas de falhas (20%) quando usadas em contexto de posicionamento e roteamento de unidades funcionais com disponibilidade de recursos. No contexto apresentado, a arquitetura possuía 90 ALUs, algumas com falhas e outras inacessíveis, porém o posicionamento e roteamento dinâmico permitiu encontrar as unidades que estavam disponíveis. A maioria dos trabalhos anteriores de arquiteturas paralelas com multiestágios e tolerância a falhas só consideravam contextos estáticos, onde é necessário garantir que qualquer entrada pode alcançar qualquer saída, o que aumenta muito o custo das redes e suporta apenas pequenas taxas de falhas (1%).

Em suma, este trabalho mostra resultados preliminares de avaliação para futuras implementações de arquiteturas reconfiguráveis de grão grosso com tradução binária e tolerância a falhas. Estes modelos são uma alternativa interessante pois aliam a flexibilidade a execução paralela em hardware, regularidade das estruturas e tolerância a falhas que são aspectos fundamentais com o avanço das nanotecnologias. Os próximos passos devem envolver a síntese e avaliação criteriosa de aspectos de layout em nível de circuito e dissipação de potência.

6.1 Trabalhos futuros

6.1.1 Análise do Consumo Energético da Arquitetura Super-VLIW

Nenhum levantamento da potência consumida pela UFR pode ser realizado e este é um fator crucial se considerarmos algumas soluções de dispositivos embarcados. Sendo assim, seria interessante que um trabalho futuro realizasse esta análise e se necessário propusesse melhores na arquitetura a fim de otimizá-la sob esta ótica. O trabalho proposto por (LO *et al.*, 2010) apresenta uma proposta de virtualização para a memoria de configuração, que seria responsável por 63% de toda a energia gasta em uma arquitetura semelhante a proposta neste trabalho. Este seria um bom ponto de partida para otimizar o consumo energético da arquitetura.

A cache de configuração utilizada no Super-VLIW é a mesma empregada em (BECK, 2008) e (LAURE, 2010). Ela explora a natureza de execução do modelo Von-Neumann, onde o contador de programa é o elemento que dirige o fluxo de

execução. Assim, a existência de um laço ou de saltos remete a repetição de certo trecho de código da aplicação.

Conhecendo esta condição a é uma tabela indexada e cada posição da cache armazena os dados necessários para a execução de uma configuração na UFR. Ela armazena os bits necessários para controlar as redes, bits de controle de cada unidade funcional e dos dados imediatos contidos nas instruções.

Esta abordagem torna a cache extensa e as informações armazenadas nelas não são necessariamente as mais utilizadas.

(LO *et al*., 2010) propuseram uma técnica denominada busca por demanda (*fetch on demand*), nela apenas os níveis que sejam efetivamente utilizados são carregados. Embora o número de acessos a memória aumente nesta abordagem a energia gasta pela a configuração é significantemente reduzida, por duas razões: o tamanho da cache é reduzido para o mesmo tamanho dos bits necessários para apenas um nível de configuração; e apenas os níveis que realmente serão utilizados são buscados da cache. O processo se assemelha ao empregado em sistemas de arquivos não contínuos, como as tabelas FAT do sistema operacional DOS.

REFERÊNCIAS BIBLIOGRÁFICAS

ABED, F.,OTHMAN, M. “Fast method to find conflicts in optical multistage interconnection networks”, International Journal of The Computer, The Internet and Management Vol. 16.No.1 pp 18-25, Jan. 2008.

ADAMS, G.B.;AGRAWAL, P.D.; SEIGEL, H.J. “A survey and comparison of fault-tolerantmultistage interconnection networks”,Computer, vol. 20, no. 6, pp. 14-27, Jun 1987.

BECK, A.C.S; RUTZIG, M.; GAYDADJIEV, G.; CARRO, L. “Transparent Reconfigurable Acceleration for Heterogeneous Embedded Applications” In IEEE/ACM DATE: pp.1208-1213,2008.

BECK, A.C.S.; CARRO, L. “Dynamic reconfiguration with binary translation: breaking the ILP barrier with software compatibility” In: DESIGN AUTOMATION CONFERENCE, DAC, 42. 2005, Anaheim. Proceedings... New York: ACM Press, p. 732 – 737, 2005.

BECK, A.C.S.; CARRO, L. “Transparent Acceleration of Data Dependent Instructions for General Purpose Processors”.In International Conference on Very Large Scale Integration66-71, 2007.

BECK, A.C.S.; GOMES, V.F.; CARRO, L. “Automatic Dataflow Execution with Reconfiguration and Dynamic Instruction Merging,” Very Large Scale Integration, 2006 IFIP International Conference on, vol., no., pp.30-35, 16-18 Oct. 2006.

BENES, V.E. “Mathematical Theory of Connecting Networks and Telephone Traffic,” New York: Academic Press, 1965.

CALLAHAN, T.J.; HAUSER, J.R.; WAWRZYNEK, J. “The garp architecture and c compiler”.Computer, vol. 33(4)pp. 62–69, 2000.

CHEN, D., RABAEY, J. “PADDI: Programmable Arithmetic Devices For DIgital Signal Processing”, In VLSI Signal Processing IV, 1990.

CHEN, S., CHEN, X., XU, Y., WAN, J., LU, J., LIU, X., CHEN, Shenggang.“Design and chip implementation of a heterogeneous multi-core DSP”Inst. of Microelectron.& Microprocessor, Nat. Univ. of Defense Technol., Changsha, China, 25-28 Jan. 2011.

CHEREPACHA, D.; LEWIS, D. "DP-FPGA: An FPGA Architecture Optimized for Datapaths," Journal - VLSI Design, 1996.

CLOS,C. “A study of non-blocking switch networks,” Bell System Tech. J. 32:407-425, March, Tech. Rep., 1953.

COMPTON, K. AND HAUCK, S. “Reconfigurable Computing: A Survey of Systems and Software”. ACM Computing Survey, v.34, no.2, 171-210, 2002.

COMPTON, K., HAUCK, S. “Automatic design of reconfigurable domain-specificflexible cores”. IEEE Trans. Very Large Scale Integr.Syst. 16(5), 493-503, 2008.

DEHON, A., NAEIMI, H. “Seven Strategies for Tolerating Highly Defective Fabrication”. IEEE Design & Test of Computers, v.22, no.4, 306-315, 2005.

DUATO, J.; YALAMANCHILI, S.; NI, L. Interconnection Networks: An Engineering Approach. Second Edition, Elsevier Science, 2003.

EBELING, C., DARREN C.C, FRANKLIN, P., FISHER, C. RaPiD: A configurable computing architecture for compute-intensive applications. Technical Report TR-96-11-03, University of Washing- ton Department of Computer Science & Engineering, 1996.

FENG T.Y., WU, C.L. "Fault diagnosis for a class of multistage interconnection networks," IEEE Trans. Comput., vol. C-30, pp. 743-758, Oct. 1981.

FERREIRA, R., LAURE, M.; BECK, A.C.S.; LO, T.; RUTZIG, M.; CARRO, L. “A Low Cost and Adaptable Routing Network for Reconfigurable Systems”In IEEE Reconfigurable Architecture Workshop RAW,2009.

FISHER, J.A. "Very Long Instruction Word Architectures", Yale University ACM 0149-7111, 1983.

FLYNN,M.J. "Veryhighspeedcomputingsystems,"Proc.IEEE,vol54,pp.1901-1909,Dec.1966.

GOLDSTEIN, S. C., SCHMIT, H., BUDIU, M., CADAMBI, S., MOE, M., and TAYLOR, R. R. “PipeRench: A Reconfigurable Architecture and Compiler”. IEEE Computer 33 (4), 2000.

GOLDSTEIN,S.C.; SCHMIT,H.; MOE,M.; BUDIU,M.; CADAMBI,S.; TAYLOR,R.R.; LAUFER, R. “PipeRench: a coprocessor for streaming multimedia acceleration”. In Published in proceedings of the 26th International Symposium on Computer Architecture ISCA 99, pp. 28–39. Atlanta, GA, 1999.

GSCHWIND K. E. M., ALTMAN E., SATHAYE S., "Binary translation and architecture convergence issues for IBM System/390" in Int. Conf. Supercomputing (ICS’00), Santa Fe, NM, pp. 336–347, 2000.

GSCHWIND, M.; ALTMAN, E.; SATHAYE, P.; LEDAK, APPENZELLER, D. “Dynamic and Transparent Binary Translation”. In IEEE Computer, vol. 3 n. 33, pp. 54-59, 2000.

GUTHAUS, M.R. *et al*. “MiBench: A Free, Commercially Representative EmbeddedBenchmark Suite” In: Workshop on Workload Characterization, 2001, Austin. Proceedings… Washington: IEEE Computer Society, pp. 3-14,2001.

HARTENSTEIN, R. “A decade of reconfigurable computing: a visionary retrospective”. In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 642-649, 2001.

HARTENSTEIN, R. “Coarse Grain Reconfigurable Architectures”.Proceedings of Asia and South Pacific Design Automation Conference, pp. 564-569, 2001.

HARTENSTEIN, R.; KRESS, R.; REINIG H.: A Dynamically Reconfigurable Wavefront Array Architecture. In Int’l Conference on Application Specific Array Processors (ASAP’94), Aug. 22-24, pp. 404-414, 1994.

HARTENSTEIN, R.; KRESS, R.: A Datapath Synthesis System for Reconfigurable Datapath Architecture. Proc. of Asia and S. Pacific DAC, pp. 479-484, 1995.

HAUSER, J.R.; WAWRZYNEK, J. “Garp: A MIPS processor with a reconfigurable coprocessor”. In Proceedings FCCM, pp. 24–33. April 1997.

KAMIURA, N.; KODERA, T.; MATSUI, N. “Fault Tolerant Multistage Interconnection Networks with Widely Dispersed Paths”, ATS '00 Proceedings of the 9th Asian Test Symposium, 2000.

KRESS, R.“A Fast Reconfigurable ALU for Xputers”, Ph. D. Dissertation, Universidade Kaiserslautern, 1996.

LAURE, M.G.; “Redes de interconexão multiestágios em arquiteturas dinamicamente reconfiguráveis de grão grosso aclopadas a processadores Risc”, Universidade Federal de Viçosa – Viçosa, MG, 2010.

LAWRIE, D.H. “Access and alignment of data in an array processor,” *IEEE Trans. Comput.*, vol. 24, no. 12, 1975.

LO, T.B., BECK FILHO, A.C.S., RUTZIG, Mateus Beck, CARRO, L. “A Low-Energy Approach for Context Memory in Reconfigurable Systems”, IEEE International Parallel And Distributed Processing Symposium (IPDPS) - Reconfigurable Architectures Workshop (RAW), 2010.

LSU, Louisiana State University, Department of Electrical & Computer Engineering. EE 7725 Interconnection Networks - Set 4: Omega Networks Lectures, 1997.

MEI, B., VERNALDE, S., VERKEST, D., DE MAN, H., and LAUWEREINS, R. “Exploiting Loop-Level Parallelism on Coarse-Grained Reconfigurable Architectures Using Modulo Scheduling”. In Proceedings of the Conf. on Design, Automation and Test in Europe, 2003.

MIRSKY, E., DEHON, A. “MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources”, in: Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, pp. 157-166, April 1996.

MOHAN, N., FUNG, W., SACHDEV, M.: “Low power priority encoder and multiple match detection circuit for ternary content addressable memory”. In: Advanced International Conference on Telecommunications AICT, 2006.

NAGELDINGER, U.“Coarse-grained Reconfigurable Architectures Design Space Exploration” University of Kaiserslautern, CS department (Informatik)Ph. D. Dissertation, 2001.

PATEL, J.H.; "Desempenho of Processor-Memory Interconnections for Multiprocessors," Computers, IEEE Transactions on, vol. C-30, no.10, pp.771-780, Oct. 1981.

PEREIRA, M. M.; LO, T. B.; CARRO, L.A Self-adaptive Approach for Fault-Tolerance in Future Technologies. The 1st HiPEAC Workshop on Design for Reliability (DFR’09), 2009.

REINIG, H. “A Scalable Architecture for Custom Computing”, Ph.D.Thesis, Univ. of Kaiserslautern, Germany, July 1999.

RIGO, S., ARAÚJO, G., BARTHOLOME, M., AZEVEDO, R. “ArchC: A SystemC-based architecture description language”. In Proceedings of 16th Symposium on Computer Architecture and High Performance Computing, 66-73, 2004.

RUTZIG, M.; BECK, A.C.S.; CARRO, L. “Balancing Rconfigurable Data Path Resources According to Applications Requirements”. In: Reconfigurable Architecture Workshop, RAW, Miami, 2008.

SANCHEZ, E.; SIPPER,M.; HAENNI,J.O.; BEUCHAT, J.L.; STAUFFER,A.; PEREZ-URIBE,A. “Static and Dynamic Configurable Systems”, IEEE Transactions on Computers, vol. 48, No. 6, pp. 556-564, June 1999.

SIEGEL, H.J.; NATION, W.G.; KRUSKAL, C.P.; NAPOLITANO, L.M., Jr. "Using the multistage cube network topology in parallel supercomputers," Proceedings of the IEEE , vol.77, no.12, pp.1932-1953, Dec 1989.

SWEETMAN D. See MIPS Run – Second Edition, Morgan Kaufmann, 2006.

TANIGAWA, K.; ZUYAMA, T.;UCHIDA, T.; HIRONAKA, T. “Exploring Compact Design on High Throughput Coarse Grained Reconfigurable Architectures”. Field Programmable Logic and Applications pp. 543-546, 2008.

TIAN,H.e.a. “A novel multistage network architecture with multicast and broadcast capability,” The Journal of Supercom- puting, vol. 35, pp. 277–300(24), 2006.

WAINGOLD, E.; TAYLOR,M.; SRIKRISHNA,D.; SARKAR, V.; LEE, W.; LEE, V.; KIM, J.; FRANK, M.; FINCH, P.; BARUA, R.; BABB, J.; AMARASINGHE, S.; AGARWAL, A. “Baring it all to software: Raw machines”. Computer, vol. 30(9)pp. 86–93, 1997.

WU, C.L.; FENG, T-Y. “On a class of multistage interconnection networks,” in IEEE Transactions on Computers, vol. C-29, pp. 694–702, August 1980.

YEUNG,A.K.W.; RABAEY, J.M. “A Reconfigurable Data-driven Multiprocessor Architecture for Rapid Prototyping of High ThroughputDSP Algorithms”, Proc. HICSS-26, Kauai, Hawaii, Jan. 1993.