MARCONE GUIMARÃES LAURE

REDES DE INTERCONEXÃO MULTIESTÁGIOS EM ARQUITETURAS DINAMICAMENTE RECONFIGURÁVEIS DE GRÃO GROSSO ACOPLADAS A PROCESSADORES RISC

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
2010

MARCONE GUIMARÃES LAURE

REDES DE INTERCONEXÃO MULTI ESTÁGIOS EM ARQUITETURAS DINAMICAMENTE RECONFIGURÁVEIS DE GRÃO GROSSO ACOPLADAS A PROCESSADORES RISC

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: 05 de março de 2010.

Prof. Carlos de Castro Goulart
(Co-orientador)

Prof. Vladimir Oliveira Di Iorio
(Co-orientador)

Prof. Henrique Cota de Freitas

Prof. Carlos Augusto Paiva da Silva

Prof. Ricardo dos Santos Ferreira
(Orientador)

Dedico essa dissertação aos meus pais
Ilton e Márcia

AGRADECIMENTOS

Antes de tudo, agradeço a Deus que me proporcionou a oportunidade para realizar esse trabalho. Pelos caminhos que me ajudou a trilhar em toda a minha vida, e por me dar forças para chegar até esse momento.

Agradeço aos meus pais, Márcia e Ilton que me criaram, me educaram e sempre me deram suporte e apoio durante toda a minha vida. Agradeço as minhas irmãs Kamillla e Maiara que sempre estiveram próximas a mim e partilharam comigo de grandes momentos da minha vida, sempre com muito carinho e amizade. Muito obrigado às minhas avós pelos momentos de paciência e bondade comigo, aos meus padrinhos que sempre foram muito próximos.

Agradeço à minha namorada Renata pelo carinho e apoio recebidos, pelas frases de consolo, pelo companheirismo e amizade.

Agradeço ao Professor Ricardo dos Santos Ferreira, por ter sido um orientador dedicado e paciente. Agradeço também aos professores do Departamento de Informática, pela formação e conhecimentos adquiridos nestes vários anos, desde o início da minha graduação até o mestrado. Pelas dicas para a carreira e para a vida.

Muito obrigado a todos os colegas, do mestrado e da graduação, que caminharam comigo em diversos períodos da minha vida, que me forneceram ajuda e companheirismo em diversas situações.

Aos vários amigos que fiz em Viçosa, companheiros de festas, gargalhadas e momentos felizes. Em especial a Dona Maria, senhoria e vizinha da República em que morei, e que durante esses vários anos foi como outra mãe para mim.

Ao CNPq, edital 13/2007 pela bolsa, que foi de grande ajuda para o desenvolvimento deste trabalho. A Banca pelos comentários construtivos.

A Universidade Federal de Viçosa.

BIOGRAFIA

MARCONE GUIMARÃES LAURE, filho de Ilton Cherubin Laure e Márcia Marídsa Guimarães Laure, brasileiro nascido em 08 de fevereiro de 1985 na cidade de Teófilo Otoni, no estado de Minas Gerais.

No ano de 2003, após concluir o ensino médio na cidade de Teófilo Otoni, ingressou no curso de graduação em Ciência da Computação na Universidade Federal de Viçosa, concluído no ano de 2007.

Em 2007, ingressou no Curso de Mestrado do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Viçosa, defendendo sua dissertação em março de 2010.

Trabalhou como professor na União de Ensino Superior de Viçosa – Univiçosa, de julho de 2007 a março de 2008. Atualmente, é Analista de Sistemas do Serviço Federal de Processamento de Dados – SERPRO, desde março de 2010.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ALU	*Arithmetic Logic Unit*
ASIC	*Application Specific Integrated Circuit*
CISC	*Complex Instruction Set Computer*
CPU	*Central Processing Unit*
CGRA	*Coarse Grain Reconfigurable Architecture*
DIM	*Dynamic Instruction Merging*
FAT	*File Allocation Table*
FIFO	*First In, First Out*
FPGA	*Field Programmable Gate Array*
GPP	*General Purpose Processor*
ISA	*Instruction Set Architecture*
Ld/Sd	*Load*/*Store*
MIN	*Multistage Interconnect Networks*
PC	*Program Counter*
PISA	*Portable Instruction Set Architecture*
RAW	*Read After Write*
RISC	*Reduced Instruction Set Computer*
SIMD	*Single Instruction Multiple Data*
TB	Tradução Binária
VHDL	*VHSIC hardware description language*
WAW	*Write After Write*
WAR	*Write After Read*
1D	Unidimensional
2D	Bidimensional
3D	Tridimensional

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

LAURE, Marcone Guimarães, M.Sc., Universidade Federal de Viçosa, Março de 2010. **Redes de interconexão multiestágios em arquiteturas dinamicamente reconfiguráveis de grão grosso acopladas a processadores RISC**. Orientador: Ricardo dos Santos Ferreira. Co-Orientadores: Carlos de Castro Goulart e Vladimir Oliveira Di Iorio.

Arquiteturas reconfiguráveis de grão grosso se apresentam como soluções escaláveis para sistemas embarcados, capazes de prover desempenho e economia de energia, ao mesmo tempo em que a granularidade grossa reduz a memória e o tempo de reconfiguração, bem como a complexidade do roteamento e do posicionamento. Contudo, mesmo em arquiteturas regulares, os custos em área de interconexão são elevados, podendo chegar a 50% da área do componente reconfigurável. Grande parte dessas arquiteturas são bidimensionais e utilizam redes totalmente interconectáveis, como redes de multiplexadores ou *crossbar*, para prover máxima roteabilidade ao custo de área extra. Neste trabalho são apresentados os benefícios do uso de redes multiestágios, de baixo custo em área e baixa complexidade, em arquiteturas de reconfiguração dinâmica e transparente. Além da economia de até 26% no total da área ocupada pela unidade funcional reconfigurável (UFR) com redes multiestágios diante da UFR com redes de multiplexadores, foi proposto um novo modelo de UFR, unidimensional, que é ainda mais compacto. Ao mesmo tempo em que a área da UFR é reduzida, a flexibilidade de acelerar aplicações heterogêneas é mantida.

ABSTRACT

LAURE, Marcone Guimarães, M.Sc., Universidade Federal de Viçosa, March, 2010. **Multistage interconnection networks in coarse grain dynamically reconfigurable architectures coupled to RISC processors**. Adviser: Ricardo dos Santos Ferreira. Co-Advisers: Carlos de Castro Goulart and Vladimir Oliveira Di Iorio.

Coarse grain reconfigurable architectures are presented as scalable solutions for embedded systems, capable of providing performance and power savings, while the coarse grain reduces memory and reconfiguration time, and reduces the routing and placement complexity. However, even in regular architectures, the interconnection costs in area are high, reaching 50% of the area of reconfigurable component. Most of these architectures are two-dimensional and uses fully conectable networks, like multiplexers networks or crossbar, to provide maximum routeability at cost of extra area. This work shows the benefits of using multistage networks, such as low-cost area and low complexity, in architectures with dynamic and transparent reconfiguration. Besides the saving of 26% in the total area occupied by the reconfigurable unit (RU) with multistage networks before the RU with multiplexers networks of multiplexers, a new model of RU, one-dimensional is proposed, which is even more compact. At the same time that the area of RU is reduced, the flexibility to accelerate heterogeneous applications is maintained.

1 INTRODUÇÃO

O mercado atual requer dispositivos eletrônicos portáteis, com diferentes funcionalidades: transmissão sem fio; reprodução; captura e edição multimídia; decodificação; localização geográfica e outras. Para cumprir todos esses requisitos, são necessários processadores de alto desempenho para aplicações heterogêneas, e que apresentem baixa potência e baixo consumo de energia.

Os ASICs (*Application Specific Integrated Circuits*) são utilizados pela indústria para prover alto desempenho com baixo custo, quando fabricados em larga escala. Contudo, não oferecem flexibilidade e possuem um alto custo de projeto.

Processadores de propósito geral, ou GPPs (*General Purpose Processors)*, por outro lado, apresentam um desempenho menor do que os ASICs. No entanto são bastante flexíveis, programáveis e aptos ao reuso de padrões. O desempenho desses processadores aumentou nas décadas de 1970 à 1990, com o desenvolvimento da tecnologia dos transistores (rapidez, miniaturização e quantidade) e de técnicas de exploração de paralelismo no nível de instrução. Todas essas características explicam por que os processadores superescalares mantiveram por tanto tempo as expectativas da lei de Moore (MOORE, 1965).

Durante as três décadas passadas, o aumento de desempenho das CPUs cresceu em torno de 40-50% por ano e serviu de estímulo para o desenvolvimento de softwares cada vez mais complexos, e com eles, novas estratégias de desenvolvimento (LARUS, 2008).

Atualmente, devido aos problemas de dissipação de potência e aos limites da exploração do paralelismo em nível de instrução, os processadores superescalares não conseguem mais acompanhar a lei de Moore, a qual afirma, desde 1965, que o número de transistores, dobraria a cada 18 meses. Um dos efeitos dessas limitações foi que a frequência de *clock* do processador Intel Pentium 4 apenas cresceu de 3,06 para 3,8 GHz entre 2002 e 2006 (INTEL, 2008). A alternativa encontrada pela indústria tem sido a replicação dos processadores, sintetizando vários núcleos em um mesmo *chip*, utilizando-se dos recursos em área obtidos com a miniaturização das

estruturas de hardware. Os novos processadores com dois ou mais núcleos superescalares da *INTEL* e da *AMD* são exemplos dessa mudança no domínio dos computadores de mesa.

No âmbito dos sistemas embarcados, a necessidade de soluções que conciliem alto desempenho e alta eficiência energética faz com que os GPPs sejam pouco interessantes quando comparados com soluções que utilizam ASICs. Em vista do custo e complexidade geradas de soluções por ASICs, as arquiteturas reconfiguráveis de grão grosso (CGRA – *Coarse Grain Reconfigurable Architecture*) vêm se destacando nos últimos anos e vários modelos de arquiteturas já foram propostos (MEI *et al*., 2005). Diferentemente das arquiteturas tradicionais, 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 específica. Outro ponto de vantagem das arquiteturas reconfiguráveis é a sua regularidade. Circuitos regulares são importantes para a escalabilidade e facilitam a fabricação do dispositivo.

As arquiteturas reconfiguráveis tradicionais, com reconfiguração estática, se aplicam 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. Nestas arquiteturas, a aceleração destes trechos envolve ainda algum tipo de transformação no código da aplicação, manual ou por meio de ferramentas específicas ou novas linguagens. Tais características dificultam a utilização destas arquiteturas como forma de prover aceleração no ambiente dos sistemas embarcados, onde existem aplicações com comportamentos heterogêneos, e um competitivo mercado (em tempo e custo).

Outro problema relacionado às arquiteturas reconfiguráveis é o alto custo das interconexões. Mesmo para as arquiteturas reconfiguráveis de grão grosso, que apresentam menor complexidade de roteamento do que os FPGA's, os custos de interconexão ainda são altos (BECK *et al*., 2008) (CLARK *et al.*, 2004) (SCHMIT *et al.*, 2002) (SWANSON *et al.*, 2003), da ordem de 50% da area total.

1.1 Motivação

A regularidade, a capacidade de prover aceleração e o baixo consumo de energia tornam as arquiteturas reconfiguráveis soluções interessantes do ponto de vista de projeto de sistemas embarcados. Contudo, as deficiências em arquiteturas

deste tipo, estão na escassez de soluções de reconfiguração dinâmica, capazes de acelerar aplicações heterogêneas, e manter a compatibilidade de software demonstrada ao longo dos anos com os velhos e ainda atuais X86 ISA. A manutenção da compatibilidade de software ocorre por meio do reuso do código binário da aplicação, e tem por vantagens a redução do tempo de projeto e a manutenção da forma tradicional de programação.

As poucas soluções de reconfiguração dinâmica geralmente se constituem em arquiteturas com topologia de duas dimensões e redes de interconexão de alto custo em área. A arquitetura abordada em Beck *et al.* (2008), usada como base para esse trabalho, é um exemplo de arquitetura que provê uma reconfiguração dinâmica e transparente, utilizando um mecanismo de tradução binária. No entanto, as estruturas de interconexão dessa arquitetura ocupam mais da metade da área do componente reconfigurável (cerca de 66%, em uma configuração com 48 linhas e 16 colunas de unidades funcionais).

A fim de potencializar o uso das arquiteturas reconfiguráveis, uma solução de reconfiguração dinâmica e com redes de interconexão de baixo custo em área, se torna um interessante objeto de estudo.

1.2 Objetivos e contribuições

Os objetivos deste trabalho estão associados a proposta do uso de redes multiestágios para reduzir a área ocupada por redes de interconexão em arquiteturas dinamicamente reconfiguráveis de grão grosso. Para tanto foi utilizado como objeto de estudo e aplicação, o modelo de arquitetura apresentado por Beck *et al.* (2008), bem como, foi proposto um novo componente reconfigurável, unidimensional, inspirado no modelo bidimensional da arquitetura original.

As principais vantagens das arquiteturas propostas neste trabalho estão na utilização dos benefícios do mecanismo de tradução binária para provimento de compatibilidade de software e no uso de redes multiestágios como alternativa de interconexão de baixo custo. Ao mesmo tempo em que as novas abordagens demonstraram economia em área de interconexão, os resultados das simulações exibiram um desempenho equivalente em relação à arquitetura proposta por (BECK et al., 2008).

As principais contribuições deste trabalho foram:

- Substituição das redes de multiplexadores por redes multiestágios na arquitetura bidimensional;
- Apresentação de uma nova arquitetura unidimensional com um modelo de interconexão baseado em redes multiestágios compatível com o mecanismo de tradução binária proposto por (BECK et al., 2008).

1.3 Organização do texto

O Capítulo 2 descreve brevemente a importância dos processadores superescalares e apresenta uma visão do campo das arquiteturas reconfiguráveis. Em especial, são descritas algumas arquiteturas de grão grosso interessantes. Este capitulo apresenta também algumas redes de interconexão, que poderiam ser utilizadas em arquiteturas reconfiguráveis, em especial, as redes multiestágios.

O Capítulo 3 trata das arquiteturas de grão grosso dinamicamente reconfiguráveis. São apresentados os trabalhos prévios, bem como, a arquitetura que foi utilizada como base para este trabalho. São descritos: o mecanismo de tradução binária, como método de prover uma reconfiguração dinâmica e transparente; e as características da UFR (Unidade Funcional Reconfigurável) e do sistema reconfigurável, bem como, as estruturas de interconexão, com seu alto custo em área.

O Capítulo 4 descreve o modelo da arquitetura bidimensional desenvolvida como parte das contribuições deste trabalho, suas características e como se deu a substituição das redes de multiplexadores por redes multiestágios. O Capítulo 5, por sua vez, descreve o modelo de arquitetura unidimensional, mais compacto do que o modelo 2D, e com uma política mais eficiente de alocação de unidades funcionais.

No Capítulo 6 são apresentados e discutidos os resultados obtidos nas simulações das arquiteturas apresentadas. Também são discutidos os custos relacionados à área do componente reconfigurável e da *cache* de reconfiguração, bem como, o desempenho das arquiteturas na execução de diferentes aplicações. Por último, no Capítulo 7, são apresentadas as conclusões e sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

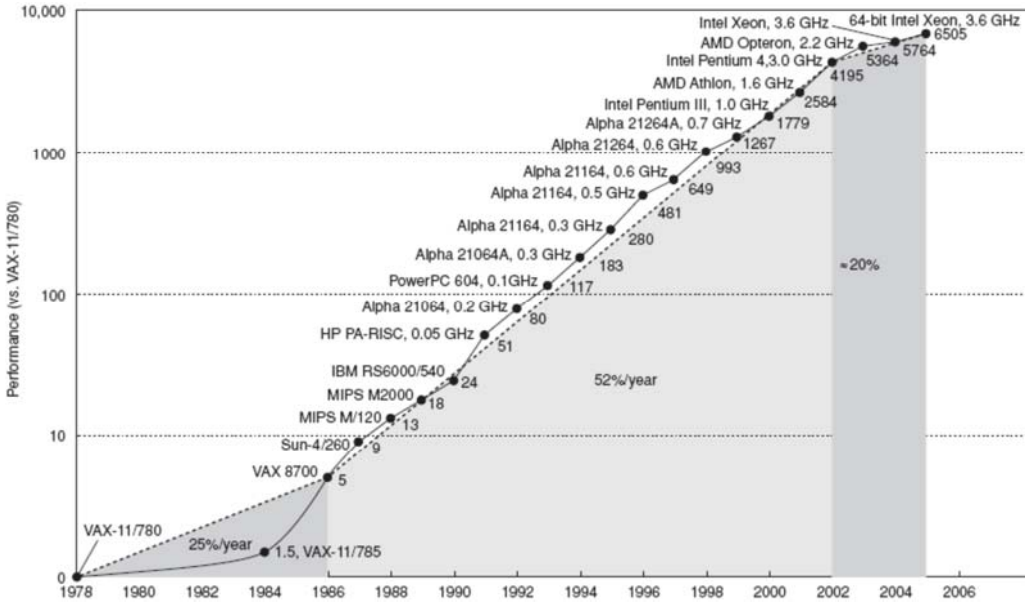
Esta seção apresenta uma breve descrição da evolução dos processadores superescalares. Logo após, é apresentada uma visão geral das arquiteturas reconfiguráveis: seus conceitos básicos; formas de classificação e alguns exemplos interessantes. Por último, são descritas diferentes redes de interconexão, uma vez que este é o foco principal desta dissertação.

2.1 Processadores Superescalares

Até a década de 1990, o desempenho dos processadores superescalares conseguiu acompanhar a lei de Moore. Entretanto, a complexidade dessa arquitetura se tornou um problema para manter o aumento esperado das taxas de desempenho (FLYNN; HUNG, 2005) (SIMA; FALK, 2004). As principais restrições são a potência, a lógica de controle e os atrasos nas interconexões (SWANSON *et al.*, 2007).

Por todas as dificuldades encontradas, a tradicional taxa de aumento de desempenho da tecnologia de processadores superescalares teve uma drástica redução de 52% ao ano entre 1986 e 2002 para menos de 25% entre 2002 e 2006, conforme ilustrado na Figura 2.1.

Como uma alternativa para continuar a prover o aumento de desempenho dos novos processadores no mercado, a Intel lançou a tecnologia *multi-core* (RAMANATHAN, 2006). No entanto, apesar da compatibilidade de software, para obter maior velocidade e eficiência de energia, novos programas e ferramentas devem ser produzidos com otimizações para as novas plataformas.



Fonte: (HENNESSY; PATTERSON, 2007)

Figura 2.1. Evolução de desempenho dos processadores de 1978 a 2006 para os benchmarks SPECint.

2.2 Arquiteturas Reconfiguráveis

2.2.1 Conceitos Básicos

As arquiteturas reconfiguráveis aparecem como um meio termo entre o desempenho dos ASICs (*hardware* específico para uma aplicação) e a flexibilidade da solução por software nos processadores de propósito geral.

Os sistemas reconfiguráveis são capazes de se adaptar a um dado contexto de execução, criando um grau de especialização do *hardware* depois de fabricado, promovendo a aceleração da aplicação, evitando sobrecargas da solução por software, e atuando na exploração do paralelismo no nível de instrução de trechos da aplicação.

Um dispositivo reconfigurável é composto por arranjos de unidades funcionais, e pelas estruturas de interconexão que as conectam. As unidades e a rede de interconexão são programadas através de uma memória de configuração, de forma a tornar o dispositivo especializado a uma dada tarefa.

No intuito de obter um maior desempenho, o *hardware* reconfigurável pode ser combinado a um processador de propósito geral. O processador se torna responsável pelas operações de controle e instruções não suportadas pelo *hardware*

reconfigurável, enquanto que blocos de instruções com alto grau de paralelismo são mapeados no *hardware* reconfigurável, como é mostrado na Figura 2.2.

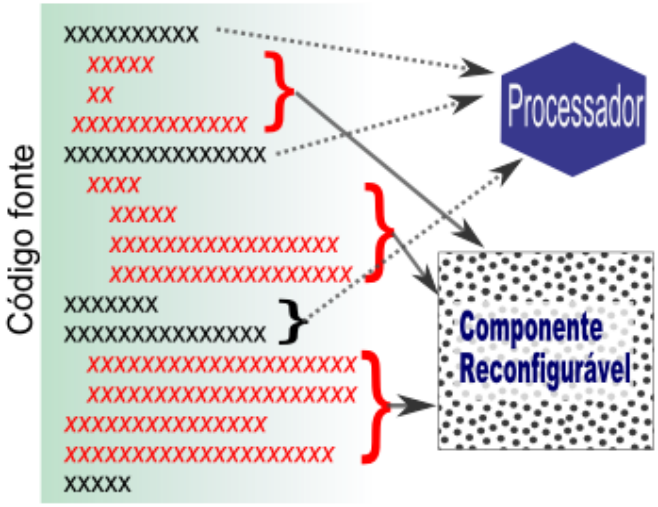


Figura 2.2. Execução combinada da aplicação no processador e na lógica reconfigurável

Vários sistemas reconfiguráveis já foram propostos como alternativa para acelerar a execução de diversas aplicações (BECK *et al.*, 2008; GAJSKI, *et al.*, 1998; GUPTA; MICHELI, 1993; VENKATARAMANI *et al.*, 2001; HUANG, 2000; GRAHAM; NELSON, 96).

2.2.2 Classificação

Não existe um modelo de classificação formal para caracterizar os sistemas reconfiguráveis, no entanto, alguns autores apresentaram modelos de classificação em seus trabalhos (COMPTON; HAUCK, 2002; CARDOSO; VESTIAS, 1999). Neste trabalho, as características consideradas mais relevantes são: o grau de acoplamento do componente reconfigurável com o processador, a granularidade do bloco lógico do componente reconfigurável, a topologia de interconexão entre os blocos lógicos e o modelo de reconfiguração do componente. Segundo esses critérios, as soluções de arquiteturas reconfiguráveis objetos desse trabalho podem ser classificadas como fortemente acopladas; de grão grosso; com topologia bidimensional (2D) ou unidimensional (1D), dependendo da arquitetura; e dinâmicas.

2.2.2.1 Acoplamento

O grau de acoplamento define a forma como o componente reconfigurável está conectado ao processador. O grau de acoplamento exerce influência na área da arquitetura e na sobrecarga de comunicação entre o componente e o processador.

São chamadas de fortemente acopladas, as arquiteturas nas quais o componente reconfigurável é integrado à via de dados (*datapath*) do processador. O componente reconfigurável nesse caso é chamado de Unidade Funcional Reconfigurável (UFR). Geralmente cabe à unidade de decodificação do processador ativar a UFR, que possui acesso ao banco de registradores do processador para ler operandos e gravar dados (MOREANO, 2005; BECK, 2008). A troca direta de dados do componente com o processador diminui a sobrecarga de comunicação.

As arquiteturas fracamente acopladas são aquelas em que a comunicação do componente reconfigurável com o processador é feita pelo barramento de entrada/saída que liga a memória ao processador, possuindo uma latência maior que o modelo fortemente acoplado. A unidade reconfigurável funciona independente do processador.

Além das arquiteturas fortemente ou fracamente acopladas existem ainda aquelas que são ditas de médio acoplamento, nas quais o componente reconfigurável está bem próximo do processador, conectado por um barramento específico (possivelmente executando concorrentemente com ele) (MOREANO, 2005). Nesse caso, o componente reconfigurável funciona como co-processador ou como elemento de processamento em um sistema multiprocessado.

2.2.2.2 Granularidade

As arquiteturas reconfiguráveis são compostas por um conjunto de blocos lógicos e uma rede de interconexão.

A granularidade de um bloco lógico se refere ao tamanho e à complexidade do bloco. A granularidade é fina quando o bloco lógico se limita a operações no nível de poucos bits (1 a 6 bits), como os FPGAs (*Field Programmable Gate Arrays*). A granularidade grossa, por sua vez, diz respeito a blocos lógicos que operam em nível de palavra, na qual cada operador implementa, por exemplo, uma operação de soma de uma palavra de 32 bits ou uma operação lógica, como um AND em nível de palavra.

A granularidade grossa perde um pouco em flexibilidade, mas reduz significativamente a sobrecarga de reconfiguração, da síntese, da área e do consumo de energia em relação aos FPGAs (SINGH *et al.*, 2000; KOREN *et al.*, 1988). Uma arquitetura de grão grosso pode também ser vista com um *datapath* reconfigurável, composto por unidades funcionais, como ALUs (*Arithmetic Logic Units*), que podem ser combinadas e configuradas para que se adequem a uma aplicação ou conjunto de aplicações (MOREANO, 2005).

2.2.2.3 Topologias

Os blocos lógicos estão conectados por meio de uma rede de interconexão que também pode ser programável. As topologias de interconexão influenciam significativamente o custo em área do componente reconfigurável. Quanto mais fina a granularidade da arquitetura (e por conseqüência, maior o número de blocos lógicos), mais complexa será a rede (MOREANO, 2005).

Arquiteturas unidimensionais são compostas geralmente por poucas unidades funcionais que comunicam dados com outras por meio de uma rede de interconexão global, como ilustrado na figura 2.3(a). O Rapid (EBELING *et al.*,1996) é um exemplo de arquitetura que utiliza topologia unidimensional.

As redes bidimensionais podem ser de dois tipos principais: malha (ou grade) ou listrada (*stripes*), ilustradas nas figuras 2.3(b) e 2.3(c). Nas arquiteturas com topologia em malha, todas as unidades funcionais podem executar operações em paralelo (a flexibilidade é alta), bem como os dados podem ser roteados em diferentes direções (norte, sul, leste, oeste, etc.). Nessas arquiteturas os dados são transmitidos aos vizinhos por meio de conexões diretas ou são transpassados às unidades de processamento mais distantes através de outras unidades intermediárias, ou de estruturas de passagem. Algumas abordagens que utilizam essa forma de interconexão são o Morphosys (SINGH *et al.*, 2000) e o WaveScalar (SWANSON *et al.*, 2003).

As arquiteturas bidimensionais listradas são baseadas nas abordagens em linha. Dentro de cada linha, todas as operações podem ser realizadas em paralelo. Os dados são enviados para a próxima linha e todas as unidades funcionais de uma linha anterior poderiam atingir qualquer unidade funcional na próxima linha. O Piperench

(GOLDSTEIN *et al.*, 2000) e a arquitetura de BECK *et al.* (2008) são exemplos de abordagens de arquiteturas listradas.

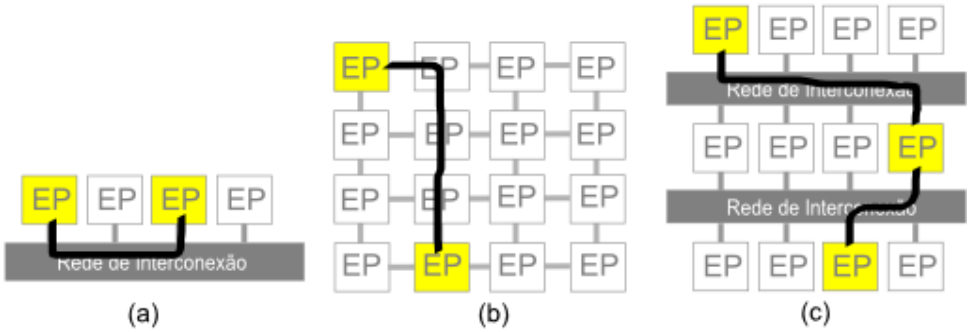


Figura 2.3. Diferentes topologias de componentes reconfiguráveis. (a) Topologia linear. (b) Topologia em Grade. (c) Topologia listrada

Além da disponibilidade em área, o custo de interligação e taxa de utilização das unidades funcionais devem ser levados em conta para o processo de reconfiguração, durante o projeto da rede de interconexão. Em geral redes muito flexíveis apresentam uma sobrecarga de reconfiguração, que deve ser compensada pelo desempenho devido ao paralelismo da aplicação.

2.2.2.4 Reconfiguração

A reconfiguração pode ocorrer em diferentes tempos, dependendo da abordagem utilizada. Se a reconfiguração é programada antes da execução da aplicação, geralmente por meio de novas instruções que estendem o conjunto de instruções do processador para capturar trechos da aplicação a serem executados em lógica reconfigurável, a reconfiguração é dita estática. A reconfiguração estática possui a desvantagem de não prover compatibilidade de código, exigindo que o código original seja recompilado para executar na plataforma reconfigurável.

A reconfiguração dinâmica por sua vez, é programada durante a execução da aplicação. Pode também ser implementada por meio do monitoramento e detecção das partes do código que executam em lógica reconfigurável, sem a necessidade de alteração (e recompilacão) de código. Essa abordagem tem a vantagem de prover compatibilidade e aceleração, enquanto contribui para diminuir o *time-to-market*. No entanto, o processo de reconfiguração dinâmica implica no aumento do tempo total de execução da aplicação, que passa a incluir o tempo de reconfiguração do componente, além do tempo gasto com a computação. Dessa forma a reconfiguração

deve ser rápida o suficiente para que o ganho obtido pela especialização do *hardware* não seja cancelado pelo tempo de reconfiguração.

Uma forma de evitar a sobrecarga da reconfiguração é permitir que o componente seja capaz de executar instruções durante este processo. Isso é possível dividindo o componente em segmentos de reconfigurações independentes, sem a necessidade de bloqueá-lo totalmente durante a reconfiguração (BECK, 2008). Moreano (2005) descreve algumas formas de acelerar a reconfiguração dinâmica da aplicação presentes na literatura. Algumas abordagens utilizam múltiplos contextos, que armazenam diversas configurações usadas em momentos distintos e rapidamente trocadas segundo a demanda da aplicação (DEHON, 1996). Outro modo de acelerar a reconfiguração é por meio da compressão das configurações para reduzir o tamanho em bits de uma configuração, e consequentemente, o tempo necessário para transferi-la para o componente reconfigurável (HAUCK *et al.*, 1998). Memórias *cache* também podem ser usadas para armazenar as configurações e acelerar a transferência de dados durante a reconfiguração (LI *et al.*, 2000). A busca antecipada das configurações permite que a reconfiguração aconteça em paralelo com a execução do processador, evitando que o processador fique bloqueado esperando pela reconfiguração (HAUCK, 1998). Existem também estudos de técnicas que visam identificar similaridades nas configurações para aproveitar segmentos já mapeados no componente reconfigurável (SHIRAZI, 1998).

2.2.3 Exemplos de arquiteturas Reconfiguráveis

Desde o surgimento do primeiro FPGA comercial desenvolvido pela *Xilinx Inc.* em meados da década de 1980, o mercado de FPGAs e outros dispositivos de lógica reconfigurável têm crescido consideravelmente. O custo dos sistemas reconfiguráveis ainda é alto se comparado com os processadores de propósito geral e a maioria dos ASICs de produção em massa. No entanto, com o aumento da demanda de mercado e da produção, os custos dos *chips* reconfiguráveis tendem a diminuir e se tornarem mais vantajosos do que os ASICs em cenários de otimização de aplicações heterogêneas, que necessitam de flexibilidade, como é caso da maioria dos sistemas embarcados atuais.

Existem ainda, soluções híbridas como o Chimaera (HAUCK, 1997) que é um FPGA acoplado fortemente a um processador de propósito geral desenvolvido para lidar com computações intensivas. O compartilhamento direto de recursos como o

banco de registradores com a unidade reconfigurável a torna uma verdadeira unidade funcional do processador. Isto simplifica a lógica de controle e diminui a sobrecarga de comunicação entre a unidade reconfigurável e o resto do sistema.

Outro exemplo é a arquitetura GARP (HAUSER, 1997), que consiste em um processador com um conjunto estendido de instruções do MIPS-II a serem executadas em um componente reconfigurável fracamente acoplado a um processador de propósito geral. A comunicação com a unidade reconfigurável é feita por meio de instruções dedicadas e, portanto, a aplicação deve ser programada e compilada especificamente para essa arquitetura.

Entretanto, os FPGAs têm um alto custo de reconfiguração, mapeamento e síntese, devido a sua granularidade fina. Em meados da década de 90, arquiteturas de grão grosso surgiram para simplificar este processo (HARTENSTEIN, 2001a).

Singh *et al*. (2000) propõem uma abordagem de arquitetura reconfigurável com as unidades de processamento acopladas a um núcleo processador RISC (*Reduced Instruction Set Computer*), denominada Morphosys. Esta abordagem opera de forma SIMD (*Single Instruction, Multiple Data*). A topologia do arranjo reconfigurável do Morphsys é uma malha 8x8, com 64 unidades funcionais de grão grosso, chamadas células reconfiguráveis, distribuídas em 4 quadrantes de quatro unidades cada. Cada unidade funcional é conectada aos quatro vizinhos adjacentes e a todas as unidades de mesma linha, ou mesma coluna, dentro do mesmo quadrante. A comunicação entre os quadrantes é feita via barramentos e na estrutura interna de cada unidade funcional existem dois multiplexadores de 16 bits, um de 13 entradas e outro de 8.

O PipeRench (GOLDSTEIN *et al.*, 2000) consiste em um arranjo bidimensional de unidades funcionais em linha, que utiliza a técnica de *pipeline* para diminuir a latência de configuração e execução. Os resultados experimentais (SCHMIT *et al*., 2002) demonstraram que o custo em área das estruturas de interconexão é maior do que 50% da área total. Portanto, um importante ponto a ser estudado são as topologias de interconexões, que consomem uma parcela significativa da área em arquiteturas reconfiguráveis.

Como alternativa ao modelo de Von Newmann, voltado para o fluxo de instruções, arquiteturas voltadas para o fluxo de dados (ou *Dataflow*) foram propostas (SANKARALINGAM *et al*., 2004; SWANSON *et al.*, 2003). Elas diferem das arquiteturas reconfiguráveis tradicionais por não serem acopladas a um processador. Apesar de possuírem um alto potencial de exploração de paralelismo,

essas arquiteturas são altamente dependentes de compiladores e ferramentas específicas, que mapeiam partes do código na ordem correta de execução. A arquitetura TRIPS (SANKARALINGAM *et al*., 2004) é um exemplo de um modelo híbrido de arquitetura *Von Newmann/Dataflow* que utiliza três diferentes modos de execução, um para dados, um para instruções e um para *threads*. O Wavescalar (SWANSON *et al.*, 2003), de forma diferente do TRIPS, é uma plataforma reconfigurável totalmente baseada na abordagem *Dataflow*, que abandona o contador de programa e a forma de execução linear de Von Newmann, que limitam a exploração de paralelismo da aplicação. O compilador do Wavescalar quebra o grafo de controle da aplicação em pedaços chamados *waves*, que serão reescritos no conjunto de instruções da plataforma alvo, e posteriormente mapeados para o arranjo reconfigurável. O arranjo do Wavescalar é denominado *Wave-cache* e possui elementos de processamento organizados em sessões de *clusters*.

O Rapid (EBELING *et al*.,1996) é um dos primeiros arranjos reconfiguráveis de grão grosso e possui topologia unidimensional. As unidades funcionais são heterogêneas e a rede de interconexão é global. No entanto, apenas poucas unidades funcionais e um conjunto específico de aplicações foram considerados. Além disso, a rede de interconexão era gerada de forma *ad-hoc*, baseada no conjunto de aplicações durante a fabricação do *chip*, o que dificulta a escalabilidade e flexibilidade da arquitetura.

Um trabalho interessante envolvendo estratégias de interconexão foi proposto por Mehta (2008). Nele, a topologia do arranjo é bidimensional em linha, com redes de multiplexadores. Cada linha tem de 15 a 25 unidades funcionais. Enquanto arquiteturas como o PipeRench conectam todos os elementos de uma linha a todos da linha subsequente, o estudo apresentado por Mehta (2008) mostra que limitando-se as conexões a um subconjunto que utiliza multiplexadores menores, pode-se obter um bom desempenho. Foram avaliadas diferentes estratégias de interconexão baseadas em multiplexadores, para aplicações com o comportamento voltado ao processamento de imagens e sinais. Os resultados indicam que a estratégia com redes de multiplexadores 5:1 apresentou melhores resultados, com uma economia de energia de 5 a10 vezes melhor em relação aos ASICs, 10 vezes melhor em relação ao *Virtex II Pro FPGA* e 100 vezes melhor do que um processador *Intel XScale*.

O DS-HIE (TANIGAWA *et al*., 2008) é uma arquitetura reconfigurável de grão grosso fortemente acoplada, que adota o modo de computação serial de bits e

um conjunto de redes de Benes (uma rede multiestágios rearranjável) como estratégias para reduzir área e garantir roteabilidade em um componente com unidades funcionais dispostas em uma topologia bidimensional em linha. Os resultados experimentais descritos pelo autor demonstraram uma taxa de desempenho de até 2,4 vezes maiores do que o desempenho de processadores *Core 2 Duo*, enquanto consome apenas 3% do número de transistores. Apesar da excelente relação desempenho/área o modelo *bit serial* e o uso da rede de Benes implicam em limitações da arquitetura. Uma delas é que a computação serial de *bits*, mesmo reduzindo a área ocupada em comparação com o método paralelo, implica em maior latência de operação, uma vez que um único cálculo consome vários ciclos. Essa latência é amenizada com o uso da técnica de *pipeline*. Outra limitação é que as ligações de dados na rede Benes devem ser conhecidas a priori, ou seja, não podem ser alteradas dinamicamente, sob demanda, em tempo de execução. Outro problema é a necessidade de *D-flip-flops* e multiplexadores nos comutadores (*switches*), para ajustar diferenças nos tempos de geração de dados, causadas por diferenças nas sequências de operações. Duas características dessa arquitetura que são semelhantes às utilizadas nesta dissertação são o uso de redes multiestágios e a capacidade de retro-alimentação de dados na mesma linha, envolvendo unidades funcionais dependentes de dados gerados por outras. Entretanto, não foi apresentada nenhuma ferramenta para o mapeamento da aplicação, os exemplos foram mapeados manualmente.

O foco deste trabalho está no uso de redes multiestágios como alternativa de baixo custo de interconexão, em arquiteturas de reconfiguração dinâmica e transparente, com o uso do mecanismo de tradução binária. O roteamento é feito dinamicamente, sob demanda.

2.3 Redes de interconexão

De acordo com Hartenstein (2001a; 2001b), a estrutura regular dos arranjos das arquiteturas de grão grosso permitem uma maior eficiência desta perante a arquitetura de grão fino, principalmente por causa da economia em área de roteamento e redução da complexidade do posicionamento, roteamento e configuração. Apesar das vantagens dos CGRAs perante os FPGAs (HARTENSTEIN, 2001b), os custos de interconexão também têm um alto impacto no custo final da arquitetura, podendo ser superior a 50% do custo total, como

acontece com o PipeRench (SCHMIT *et al.*, 2002) . O padrão de interligação pode ser avaliado pelo seu diâmetro, sua distância média, o grau de seus nós, o número de recursos para roteamento ou chaveamento, a área em silício, a potência, etc. O diâmetro é a distância entre os dois nodos mais distantes. O grau de um nó é o número de vizinhos ligados diretamente a qualquer nó particular. O custo de implementação de um nó, na maioria dos casos, é dominado pelo número de ligações ou número de recursos para roteamento (como multiplexadores).

Os CGRAs podem ser classificados de acordo com o posicionamento e o padrão de comunicação entre seus elementos de processamento: Arquiteturas baseadas em malha; Arquiteturas baseadas em arranjos lineares e Arquiteturas *crossbar*. Dentre os três tipos citados, o modelo de rede *crossbar* se destaca por prover alto desempenho, mínima latência e mínimo congestionamento, e devido a sua natureza livre de bloqueios (HUR, 2007). Uma malha 2D é uma rede de baixo custo e grau, porém seu diâmetro e distância média tem complexidade O ($N^{1/2}$), onde N é o número total de nodos. Uma rede *crossbar* tem distância O(1), porém tem um custo de O(N^2) (GRAMMATIKAKIS *et al.*, 2001).

As redes de interconexão se dividem em duas abordagens: redes estáticas e redes dinâmicas. As redes estáticas podem ser classificadas de acordo com a dimensão da rede e topologia (padrão de ligação entre os nós). São exemplos de redes estáticas: barramento; malha 2D e 3D; anel; estrela; árvore. As redes dinâmicas, não possuem uma estrutura fixa entre os nós, as ligações passam por elementos de chaveamento (comutadores), proporcionando maior flexibilidade. Exemplos de redes dinâmicas são as multiestágios, *crossbar* e de multiplexadores. Neste trabalho lidamos com redes dinâmicas.

2.3.1 Redes Dinâmicas de interconexão

As redes *crossbar* e de multiplexadores são capazes de perfazer quaisquer grupos de ligações ponto a ponto ou *multicast* entre suas entradas e saídas. No entanto, essas redes possuem custo em área na ordem de N^2 elementos de chaveamento, onde N é o número de entradas, e também o número de saídas. A Figura 2.4 demonstra uma rede *crossbar* com as ligações 2 → 7 e 4 → 6 mapeadas. A Figura 2.5(a) exemplifica uma rede de multiplexadores mapeada com estas mesmas ligações. Um multiplexador N:1 tem custo de (N-1) multiplexadores 2:1, conforme o exemplo do multiplexador 8:1 da figura 2.5(b).

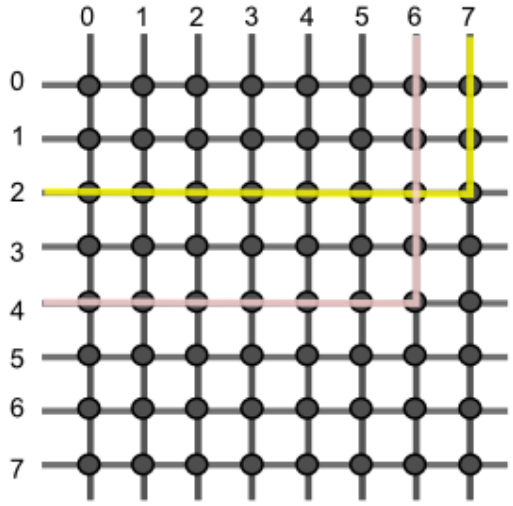


Figura 2.4. Roteamento das ligações 2→7 e 4→6 numa rede cross-bar 8x8

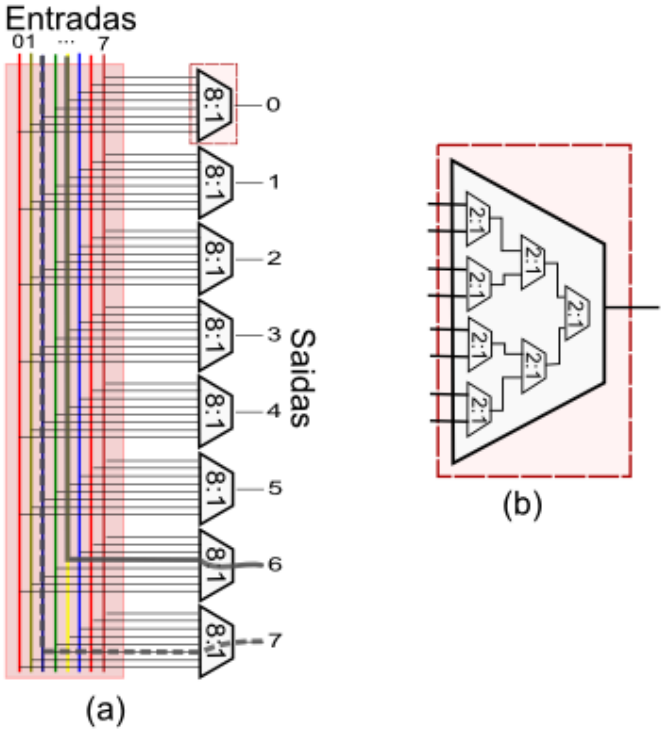
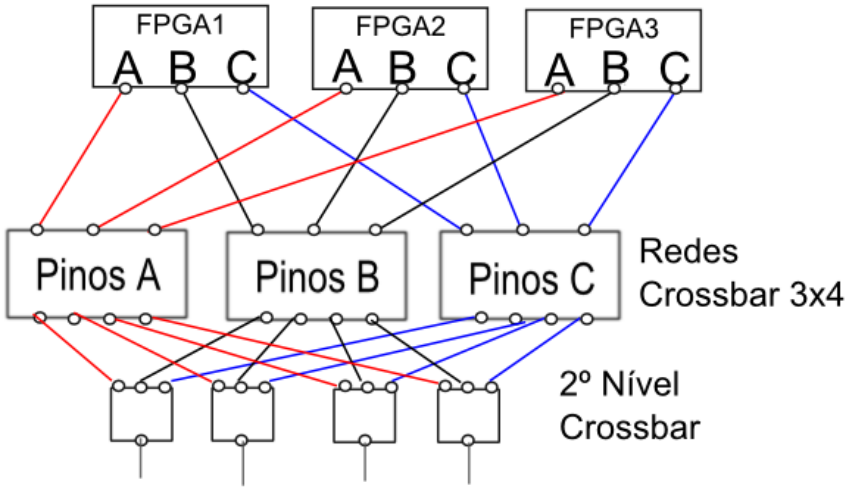


Figura 2.5. Roteamento das ligações 2→7 e 4→6 numa rede de multiplexadores 8x8 (a). Cada multiplexador 8:1 é composto de 7 multiplexadores 2:1 (b)

As vantagens das redes *crossbar* são: o atraso com custo O(1) , uma vez que, uma ligação atravessa apenas um elemento de chaveamento para conectar uma entrada em uma saída; e sua capacidade de realizar quaisquer permutações de ligações. Contudo o seu alto custo em área, $O(N^2)$, incentivou a busca por novas alternativas de redes de interconexão. Nesse contexto, surgiram as redes *crossbar*

parciais. Tais redes consistem em dividir o conjunto de entradas e de saídas totais em conjuntos disjuntos menores. Cada conjunto de entradas se liga a um dado conjunto de saída por uma rede *crossbar*. Uma rede *crossbar* com N=A+B entradas/saídas tem custo $(A+B)^2$ enquanto uma rede parcial formada por duas redes com A e B entradas/saídas tem custo A^2+B^2. As diferentes redes podem ainda estar ligadas entre si usando lógica semelhante na forma de multiníveis (Figura 2.6), como alternativa para aumentar o número de ligações (VARGHESE *et al.*, 1993).



Fonte: Adaptado de (VARGHESE *et al.*, 1993)

Figura 2.6. Rede *crossbar* parcial multiníveis. Na Figura dois níveis *crossbar* são utilizados para transpassar dados até as entradas dos FPGA.

2.4 Redes Multiestágios

As redes de interconexão multiestágios, do inglês "*Multistage Interconection Networks*" ou simplesmente MIN, têm sido estudadas por mais de cinco décadas. Elas consistem em redes com mais de um estágio de comutadores interligados, capazes de conectar uma entrada qualquer a uma saída arbitrária (se não houver conflitos por recursos de interconexão).

Os primeiros trabalhos foram publicados por Clos (1953) e Benes (1965) com aplicação voltada para o domínio das redes de telefonia. Em 1975, Larwrie (1975) propôs a rede Omega, com apenas $\log_2(N)$ estágios; roteamento simplificado e; apenas uma alternativa de ligação entre origem e destino. Durante os anos 80 e 90, diversos trabalhos foram propostos para explorar alternativas de padrões de conexão e roteamento em redes MIN. Alguns exemplos são os trabalhos apresentados em:

Feng & Seo (1994); Wu & Feng (1980); Gazit (1989) e; Yeh & Feng (1992). Ao longo dos anos as redes MIN têm sido usadas para a conexão de elementos de processamento ou memória em processadores vetoriais e máquinas SIMD, e recentemente, em redes ATM ou Ópticas (TIAN *et al.*, 2006). A alta capacidade de integração proporcionada pelo avanço da Tecnologia, estimula o uso de redes MIN em SoCs (*Systems on Chip*). As NoCs (*Networks on Chip*) proveêm escalabilidade e reusabilidade, quando comparadas às redes de barramentos, tipicamente utilizadas em SoCs (MEFTALY *et al.*, 2005). NoCs diferem das redes tradicionais por geralmente consumirem menos recursos de hardware, software ou memória e pelo *design* mais flexível. Contudo, consomem mais fios e pinos do que as redes tradicionais (MEFTALY *et al.*, 2005; BENINE; MICHELI, 2002). Um trabalho recente sobre o uso de redes multiestágios NoCs é descrito por Ludovici *et al.* (2009). Tal abordagem propõe uma rede multiestágios chamada RUFT (*Reduced Unidirectional Fat-Tree*), que é parecida com uma rede Butterfly, com diferença no padrão de conexão dos comutadores do último estágio. Foram comparadas arquiteturas que utilizam redes com topologia em malha 2D ou multiestágios, para interligar núcleos de processamento ou de memória. Os resultados da análise demonstram que NoCs RUFT podem prover melhor desempenho, em relação a malha 2D, a medida que a escala do sistema aumenta, e ao mesmo tempo reduzem a sobrecarga de área e potência. A área avaliada das redes RUFT ou em malha, em uma arquitetura de 16 núcleos é praticamente igual, enquanto que, em uma arquitetura com 64 núcleos o *overhead* de área da RUFT é cerca de 30% em relação a rede de malha 2D. As redes utilizadas neste trabalho diferem da abordagem anterior ao conectarem simples unidades funcionais (como ALUs), ao invés de elementos de processamento, isso permite que fios mais curtos possam ser utilizados. O trabalho anterior utiliza também comutação por pacotes e a política FIFO, entre as ligações de dados, neste trabalho foi utilizada a comutação por circuito e as ligações anteriores permanecem na rede. O estudo de uma versão NoC das redes apresentadas nesse trabalho é uma interessante proposta de trabalho futuro, uma vez que, essas redes estão próximas de se tornarem as arquiteturas de interconexão dos sistemas de computação paralela de alto desempenho (BENINE; MICHELI, 2002).

As redes multiestágios podem ser classificadas em três classes distintas: bloqueantes; rearranjáveis ou; não bloqueantes. A Figura 2.7(a) ilustra uma rede bloqueante Omega e a Figura 2.7(b) ilustra uma rede rearranjável Benes.

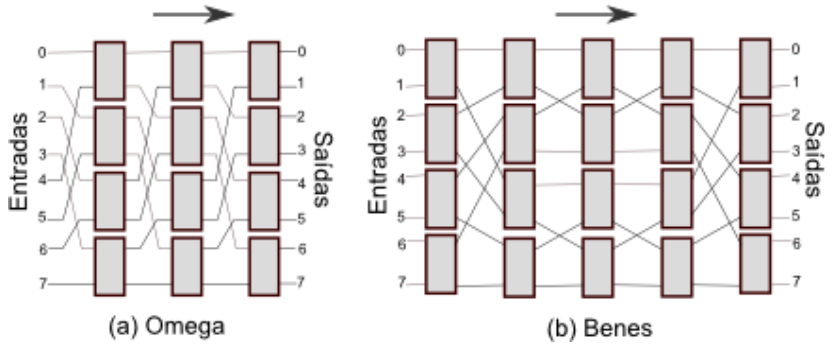


Figura 2.7. Exemplos redes de interconexão multiestágios 8x8: (a) rede Omega de um lado e (b) rede Benes de dois lados

Uma rede é bloqueante quando não é capaz de realizar qualquer permutação devido a conflitos entre as ligações. Alguns exemplos de redes bloqueantes são: Omega (LAWRIE, 1975), Baseline (WU; FENG, 1980), Butterfly (GRAMMATIKAKIS *et al.*, 2001) e SW Banyan (GOKE; LIPOVISKI, 1973). Wu e Feng (1980) estabeleceram o relacionamento de equivalência entre as diferentes topologias dessa classe. Uma rede bloqueante NxN (onde N é o número de entradas e o número de saídas) tem em geral $\log_2(N)$ estágios. A Figura 2.8 demonstra exemplos de redes MIN equivalentes pela classe. As diferentes redes da figura diferem quanto aos padrões de conexão dos comutadores, embora possuam a mesma capacidade de roteamento.

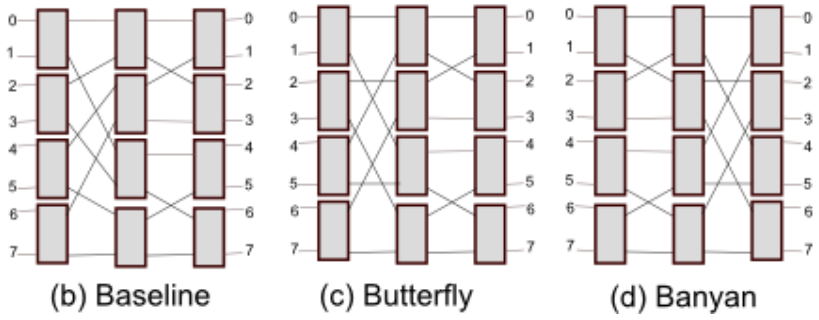


Figura 2.8. Exemplos de redes multiestágios 8x8 equivalentes pela classe.

As redes rearranjáveis são aquelas que são capazes de realizar qualquer permutação de entrada/saída desde que seja feito o rearranjo das conexões existentes. Já as redes não bloqueantes, são aquelas que realizam quaisquer permutações sem a necessidade de rearranjo das ligações. As redes de Benes (1965) e Clos (1953) são exemplos de redes rearranjável e não-bloqueante, respectivamente. Todavia, as redes rearranjáveis e não-bloqueantes apresentam um custo maior. O número de estágios

de comutadores em uma rede rearranjável é no mínimo $2 \times \log_2(N) - 1$, o equivalente a quase duas redes Omega, com $\log_2(N)$ estágios. Outro ponto a ser ressaltado é que a maioria dos algoritmos de roteamento para redes rearranjáveis, como os descritos por Çam & Fortes (1999), requerem o conhecimento *a priori* de todo o conjunto de ligações a ser mapeado na rede. De forma diferente, neste trabalho consideramos um ambiente no qual as conexões são feitas sob demanda (*on-the-fly*).

Neste trabalho utilizamos uma abordagem de comutação de circuitos, diferente de outros trabalhos que utilizam a abordagem baseada em comutação de pacotes (SHIMIZU, 1990). São abordadas as ligações entre unidades funcionais de uma arquitetura reconfigurável, visando uma redução considerável da área *on-chip* de interconexão, com pouca ou nenhuma perda de desempenho.

2.4.1 Redes Multiestágios Omega

Uma grande vantagem da rede Omega é o seu comportamento fácil de analisar em comparação com outras topologias equivalentes (WU; FENG, 1980). A rede Omega foi inicialmente proposta por Larwrie (1975) para a conexão e acesso de dados em processadores vetoriais. Uma rede Omega N×N, em que $N = 2^M$ consiste em $M = \log_2(N)$ estágios idênticos de N/2 comutadores, cada estágio é interligado ao próximo estágio através de um padrão de interconexão *perfect-shuffle*. A rede Omega é uma das redes multiestágios mais populares em computação paralela (HWANG; BRIGGS, 1990). A Figura 2.9 (a) exemplifica uma rede Omega 8×8. Cada comutador pode assumir originalmente dois diferentes estados completos: "*straight through*" (ligação direta) e "*exchange*" (ligação cruzada); e dois estados de *broadcast*, o "*down broacast*" (dispersão abaixo) e "*upper broadcast*" (dispersão acima); e quatro estados "incompletos" derivados dos completos; além desses estágios pode ainda existir ainda o estágio "desligado". Para cada comutador é gasto um bit para representar apenas estados completos, 2 bits para representar também os estados de *broadcast* e de 3 a 4 bits para incluir os estados incompletos e desligado. A Figura 2.9 (b) exemplifica os diferentes estados que um comutador pode assumir. Neste trabalho iremos considerar comutadores com 2 bits (estágios completos e de *broadcast*).

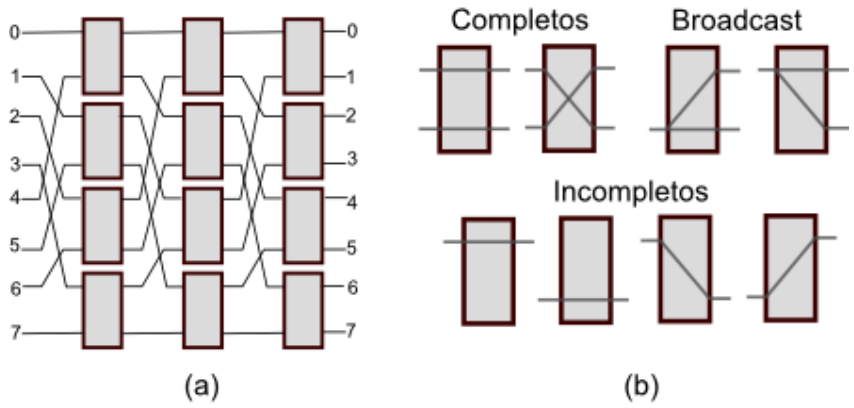


Figura 2.9. (a) Rede multiestágios Omega 8x8 e (b) os diferentes estados que cada um de seus comutadores pode assumir.

O roteamento entre uma dada entrada e uma dada saída na rede consiste na configuração de um caminho único, que passa por um comutador em cada estágio. A configuração dos comutadores em cada estágio pode ser facilmente obtida da operação lógica de XOR ("ou exclusivo") entre endereços (em binário) dos terminais de entrada e saída. Por exemplo, vamos considerar a rede 8x8 da Figura 2.10 (a) na qual se deseja rotear a entrada de índice 1 (001) com a saída de índice 5 (101). O resultado da operação 001 XOR 101 = 100 nos diz que o primeiro comutador da ligação estará no modo cruzado (*exchange*) e os dois seguintes, em ligação direta (*straight through)*. Cada i-ésimo bit do resultado determina a configuração do comutador do i-ésimo estágio da rede. O bit mais significativo (da esquerda para a direita) determina o resultado do primeiro estágio (da esquerda para a direita) e o bit menos significativo determina o resultado do último estágio, com $\log_2(N)$ estágios na rede.

Em muitos casos, é importante que um dado proveniente de uma origem possa atingir todos os destinos (*broadcast*) ou a um subconjunto de destinos (*multicast*). É o que ocorre no ambiente de aplicação, em que um dado de origem pode ser roteado para diferentes destinos (correspondentes as unidades funcionais da arquitetura reconfigurável). Essa possibilidade de uma origem atingir mais de um destino é obtida com um ou mais comutadores em estado de *broadcast*. A Figura 2.10 (b) demonstra um exemplo de *multicast* para as ligações 1→5 e 1→3.

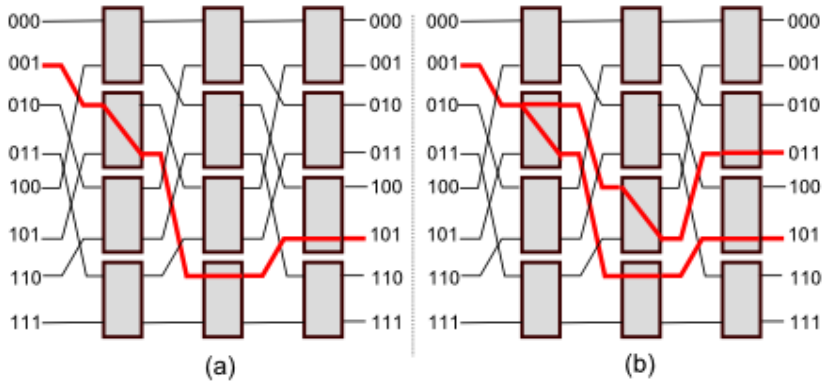


Figura 2.10. (a) Rede Omega roteando a ligação 1→5. (b) Roteamento em *multicast* das ligações 1→5 e 1→3.

Como existe um único caminho entre uma entrada e uma saída, e um mesmo comutador da rede pode ser solicitado por diferentes ligações, é comum uma ligação não ser realizada ao solicitar um segmento que já estava sendo usado por outra, gerando um conflito de roteamento. A Figura 2.11(a) demonstra um caso de conflito na rede. Ao tentar rotear a entrada 2 (010) com a saída 7 (111) ocorre um conflito no comutador requisitado no segundo estágio da rede (2º comutador dessa coluna) A ligação 2→7 conflita com a ligação 4→6, pré-existente, e não pode ser realizada enquanto esta última estiver ativa.

Para minimizar os efeitos dos conflitos e aumentar a efetividade das ligações na rede Omega é possível aumentar o número de rotas possíveis entre uma origem e um destino por meio do uso de estágios extras (SHEN; ZHANG, 2000). Para cada estágio (ou coluna de comutadores) extra adicionado a rede, o número de caminhos possíveis dobra, ou seja, se adicionarmos 1 estágio extra a rede, existirão 2 rotas possíveis entre uma entrada e uma saída quaisqueres, se houverem 2 estágios extras, o número de rotas sobe para 4, e assim por diante. Todavia, cada estágio extra possui seu próprio custo em área, latência e dissipação de potência estática da rede. Isso significa que um estágio aumenta a área ocupada pela rede e o esforço de configuração. Uma rede Omega originalmente com $M = \log_2(N)$ estágios ao dobrar seu número de estágios ($2 \log_2(N)$) passa a ter o mesmo custo de uma rede rearranjável. Portanto, o número K de estágios extras deve ser pequeno dentro do intervalo $0 \leq K \leq \log_2(N)$ de forma a proporcionar capacidade de conexão a um baixo custo.

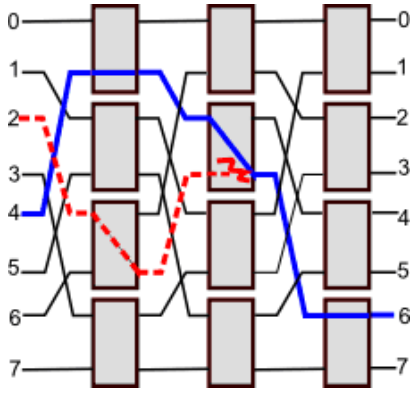


Figura 2.11. (a) Conflito de roteamento da ligação 2→7 com a ligação 4→6.

Um algoritmo de roteamento para redes Omega é descrito em Shen & Zhang (2000) e leva em conta a possibilidade de *multicast*, a identificação de conflitos e o uso de estágios extras. Em uma rede com M+K estágios (onde K é o número de estágios extras) o deslocamento de uma janela de comprimento M ao longo de uma sequência de M+K+M bits, composta pela concatenação do endereço de entrada com um valor de K bits concatenado ao endereço de destino, fornece estágio por estágio, a linha correspondente a porta de saída do comutador a ser usado. Ao variarmos o valor dos K bits, obtemos diferentes rotas para uma mesma ligação, com o limite de 2^K rotas. A Figura 2.12 esboça o processo de roteamento da ligação da entrada 2, em binário 010, com a saída 5 (101 em binário). Primeiro os bits dos dois endereços são concatenados com o bit de estágio extra no meio, formando a palavra 010**?**101. O bit de estágio extra **?** pode ser 0 ou 1. Fazendo **?** = 0, teremos a palavra 0100101. Uma janela de 3 bits irá se deslocar sobre a palavra, fornecendo a cada estágio descrito na figura 2.12 (a) o endereço da próxima porta de saída do conjunto de switches correspondente ao estágio, sendo a origem (E0), a porta de entrada. A porta de saída no estágio 1 é a porta 4 (100 em binário), no estágio 2, é a porta 1 (001) e assim por diante, conforme esboçado na figura 2.12 (b) para a rede com **?** = 0. A segunda possível rota é obtida fazendo **?** = 1. Nesse caso a janela se deslocará ao longo da palavra 0101101, fornecendo diferentes portas de saída nos estágios intermediários, conforme esboçado na figura 2.12 (b), para **?** = 1.

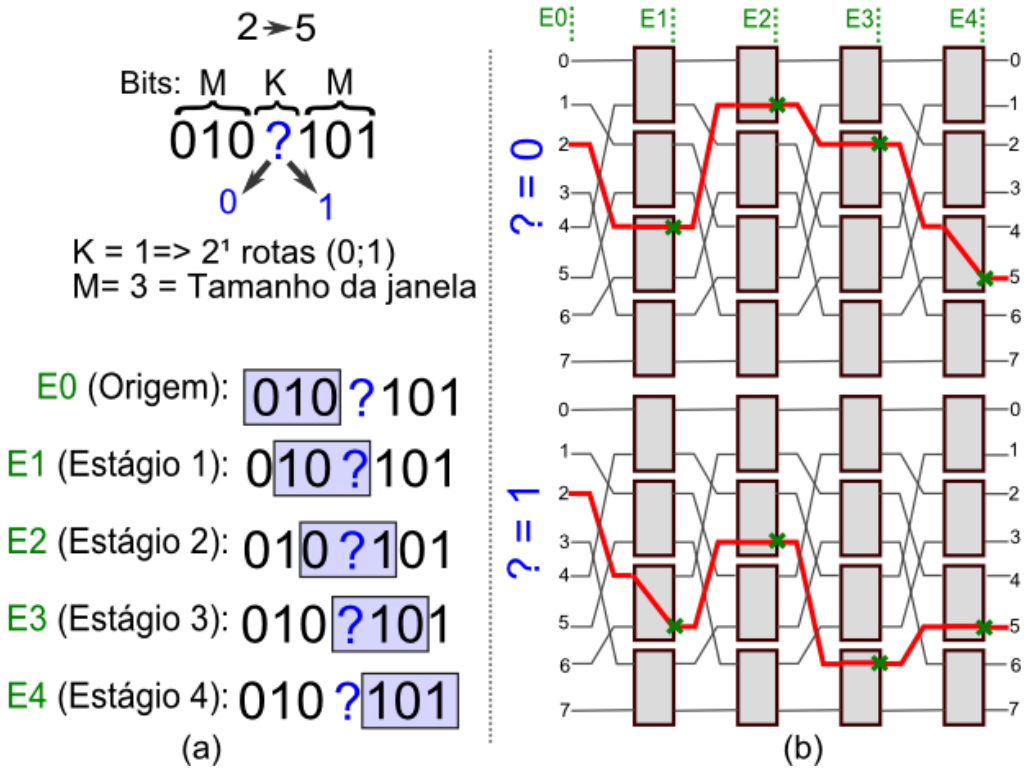


Figura 2.12. Roteamento da ligação 2→5. Os estágios de deslocamento da janela (a) fornecem os índices das portas de saídas dos comutadores (b)

3 ARQUITETURAS DINAMICAMENTE RECONFIGURÁVEIS

Vimos que os sistemas reconfiguráveis podem aliar alto desempenho e baixo consumo de energia. Contudo, os sistemas reconfiguráveis tradicionais apresentam duas desvantagens principais para aplicações heterogêneas em sistemas embarcados:

- A primeira é que eles são concebidos para lidar apenas com a manipulação intensiva de dados em *kernels*. A idéia principal é a de que otimizando pequenas porções de código uma grande aceleração pode ser obtida, e não é adequada a aplicações com comportamentos heterogêneos.

- A segunda desvantagem reside na necessidade de algum tipo de transformação, manual ou por meio de linguagens especiais ou ferramentas de apoio, que modificam o código binário (ou em alto nível) dos trechos da aplicação a serem mapeados na lógica reconfigurável. Desta forma, a reconfiguração da aplicação ocorre em tempo de compilação e não há reuso de código.

A reconfiguração dinâmica aliada à tradução binária, mantém a compatibilidade de *software*, ao contrário de técnicas que exigem a recompilação de código. A reconfiguração dinâmica (CLARK *et al.*, 2004; BECK *et al.*, 2008) se dá em tempo de execução, agrupando os blocos de instruções de uma dada aplicação, mapeando-a paralelamente no *datapath* reconfigurável. Esse comportamento de execução dispensa a necessidade de recompilação de código. A portabilidade da aplicação é mantida e o *time-to-market* e os custos de produção dos *softwares* podem ser drasticamente reduzidos. Neste capítulo, iremos apresentar a arquitetura dinamicamente reconfigurável, com tradução binária, usada como referência para este trabalho.

3.1 Detecção dinâmica e Reconfiguração

Clark *et al.* (2004) propôs uma abordagem dinâmica de reconfiguração da unidade funcional, sem a necessidade de extensão do conjunto de instruções do

processador. O CCA (*Configurable Computer Acelerator*) consiste em um arranjo de unidades funcionais heterogêneas fortemente acoplado a um processador ARP e com geometria triangular. A saída de uma linha e a entrada da próxima linha do CCA estão totalmente conectadas, isso aumenta os custos de interconexão e o número de bits de controle, mas reduz a complexidade da tradução dinâmica que monta as configurações do CCA. Nessa abordagem são omitidas as medidas de área, consumo de energia e tempo; além de detalhes do mapeamento das unidades no processo de reconfiguração.

Os primeiros estudos sobre os benefícios do particionamento dinâmico de aplicações para execução em um sistema reconfigurável foram reportados por Lysecky (2006). Sua abordagem, chamada de *Warp Processing* é um complexo *System on Chip* (SoC) composto por dois processadores: um responsável pela execução da aplicação e outro pelo particionamento. Além destes processadores, uma memória local e um FPGA compõem a arquitetura. Em primeiro lugar, o microprocessador executa o código binário original enquanto um *hardware* monitor verifica o perfil das instruções, a fim de detectar regiões críticas. Em seguida, o software de particionamento remonta o fluxo de instruções em um grafo de fluxo de controle e, após sintetizá-lo, o mapeia no FPGA. Por fim, o código binário é modificado para incluir o suporte que foi gerado para a plataforma reconfigurável. As deficiências dessa plataforma estão no alto consumo de memória para a execução do algoritmo de particionamento (cerca de 8 MB), e nas limitações dos FPGAs em latência, área e consumo de energia.

Beck *et al.* (2008) propôs um algoritmo de tradução binária implementado em *hardware* que trabalha em paralelo a um processador MIPS acelerando aplicações *dataflow* e *controlflow*. Esse algoritmo mapeia sequências de instruções, em tempo de execução, em um arranjo reconfigurável de grão grosso. A unidade reconfigurável e o *hardware* de tradução binária são fortemente acoplados ao processador. A execução dinâmica e transparente proporcionada pela tradução binária elimina a necessidade de extensão do conjunto de instruções e, portanto, de recompilação do código. A idéia de execução dinâmica é semelhante a do CCA e do *Warp Processing*. O modelo de Beck apresenta uma topologia bidimensional em linha, no qual as unidades funcionais estão dispostas em linhas que correspondem, cada uma, a um ciclo de execução. As unidades são conectadas em redes de interconexão que existem entre as linhas. Todas as unidades de uma mesma linha podem propagar suas

saídas para unidades da próxima linha. A rede de interconexão é baseada em multiplexadores e ocupa uma parcela significativa (cerca de 50%) da área da unidade reconfigurável.

3.1.1 Trabalhos prévios

Esta subseção apresenta alguns trabalhos que precederam essa dissertação, envolvendo reconfiguração dinâmica e tradução binária em arquiteturas de grão grosso. Em (BECK *et al.*, 2005) foi proposto um algoritmo de tradução binária que executa sobre um arranjo reconfigurável de grão grosso acoplado a um processador que executa nativamente Java.

Em (BECK *et al.*, 2006) o algoritmo de tradução binária foi revisto para execução com um processador RISC, utilizando o simulador *SimpleScalar* (BURGER, 1997), executando o conjunto de instruções PISA.

Já em (BECK *et al.*, 2007), o algoritmo de tradução binária foi novamente atualizado para incluir um mecanismo de especulação de desvios, executando sobre aplicações do Mibench (GUTHAUS *et al.*, 2001). Usando a abordagem de arquitetura anterior como caso de estudo, Rutzig *et al.* (2008) propõe uma ferramenta para balancear os requisitos de área e desempenho do *datapath* reconfigurável segundo os requisitos de execução das aplicações, demonstrando que o sistema otimizado pode reduzir em até quatro vezes a área, com uma redução de desempenho média de apenas 5,8%, na avaliação de um subconjunto do benchmark Mibench. Esta ferramenta verifica o comportamento de execução da aplicação e utiliza-o para montar o grafo de dependências de dados entre as instruções. Esses grafos são armazenados em uma base de dados. Por último, a ferramenta extrai algumas características desses grafos (altura, largura, taxas de reuso, número e tipos de instruções, relações de dependências, etc.) e as mescla para decidir qual o melhor configuração para a unidade reconfigurável, principalmente com base no paralelismo das instruções e na taxa de reuso.

3.2 Arquitetura de Referência

Como dito antes, o sistema reconfigurável que foi utilizado como ponto de partida para este trabalho é descrito por Beck *et al.* (2008), e consiste de uma Unidade Funcional Reconfigurável de grão grosso fortemente acoplada ao

processador escalar MIPS R3000, e a um *hardware* de tradução binária e a uma memória *cache* de reconfiguração.

A reconfiguração transparente é realizada com o auxílio do *hardware* de tradução binária, o qual utiliza uma técnica chamada de *Dynamic Instruction Merging* (DIM) (BECK et al., 2008), que traduz os blocos de instruções em lógica reconfigurável durante a execução da aplicação no processador. Os blocos traduzidos são armazenados na *cache* de reconfiguração, de onde poderão ser obtidos para posteriormente executarem na UFR.

O MIPS R3000 é um processador RISC, cujo pipeline é composto por 5 estágios (HENNESSY; PATTERSON, 2007): Busca da Instrução (BI ou *Instruction Fetch*), Decodificação da Instrução (DI ou *Instruction Decode*), Execução (EX ou Execution), Acesso a Memória (AM ou *Memory Access*) e escrita do resultado (ER ou *Write Back*).

A Figura 3.1 apresenta o esquema de acoplamento dos componentes do sistema reconfigurável. A UFR bidimensional é composta pelos seus elementos de processamento (EP) e estruturas de interconexão. O *hardware* de tradução binária (TB) é implementado em pipeline e a detecção ocorre em paralelo com a execução no processador, sem interrompê-lo. A detecção no tradutor binário passa por 4 estágios distintos (RUTIZIG; CARRO, 2008):

- **Decodificação da Instrução (DI)**: A instrução é decodificada para descobrir seus operandos, o seu tipo de operação e qual o tipo de unidade funcional deve recebê-la.

- **Validação de Dependências (VD)**: Nesta etapa são validadas as dependências RAW (*Read After Write*). Dependências WAR (*Write After Read*) ou WAW (*Write After WRITE*) não precisam de verificação especial, e são garantidas pela própria lógica de alocação de operandos do tradutor binário.

- **Alocação de Recursos (AR)**: Nesta etapa verifica-se a disponibilidade das unidades candidatas, alocando a primeira disponível para receber a instrução.

- **Atualização das tabelas e mapas de bits (AT)**: Nesta última etapa, os dados das tabelas e mapas de bits são atualizados para serem utilizados pela próxima instrução. Ao final da detecção do bloco de instruções os dados referentes à reconfiguração são enviados para a *cache* de reconfiguração.

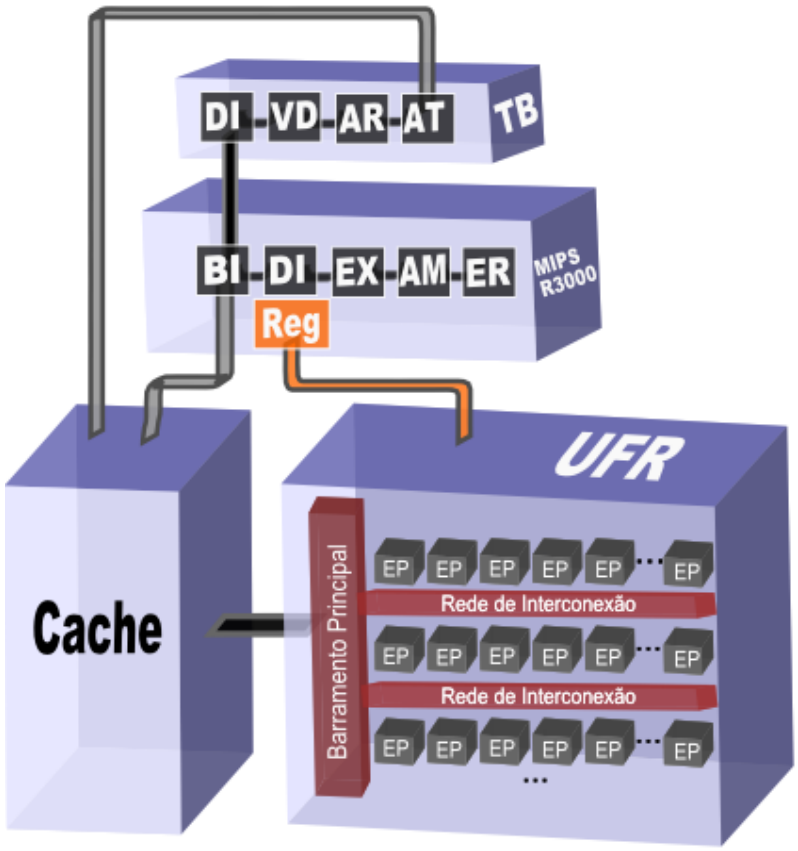


Figura 3.1. Arquitetura Reconfigurável bidimensional fortemente acoplada ao processador MIPS

3.2.1 UFR bidimensional listrada

O grão mínimo de configuração da UFR é a unidade funcional, daí a sua classificação como arquitetura de grão grosso. Os tipos de unidades funcionais suportadas e quais instruções (referentes ao MIPS) cada uma é capaz de lidar, bem como o tempo de execução de cada uma delas, são descritos na tabela 3.1. Instruções fora desse conjunto são tratadas pelo tradutor binário como incompatíveis e serão executadas no fluxo normal do processador, provocando o término da execução corrente na UFR. Vale ainda ressaltar que operações com ponto flutuante não são suportadas pelo arranjo reconfigurável, ou seja, somente operações que manipulam valores inteiros podem executar na UFR.

Tabela 3.1. Unidades funcionais da arquitetura, suas operações e tempos de execução

Unidades Funcionais	Operações Executadas	Instruções Alocadas	Tempo de execução
Grupo 1: Unidade Lógica e Aritmética	Soma, Subtração, Deslocamento, Comparação, Lógica de Bit	*add, addu, sub, subu, addi, addiu, and, andi, or, ori, xor, nor, slt, slti, sll, srl, sra, beq, bne, j, jr, jal, lui, mfhi, mflo*	1/3 ciclo
Grupo 2: Load/Store	Leitura/Gravação de dado na memória	*lw, lh, lhu, lb, lbu, sw, sh, sb*	1 ciclo
Grupo 3: Multiplicador	Multiplicação	*Mul*	1 ciclo

Fonte: Adaptado de (RUTZIG, 2008)

A UFR tem topologia bidimensional em linha, isso significa que as unidades funcionais se distribuem ao longo do arranjo em conjuntos de unidades em camadas. A Figura 3.2 exemplifica a alocação de um bloco de instruções na UFR. Cada ciclo de execução é composto por três linhas de unidades funcionais e as unidades de uma mesma linha executam em paralelo. No exemplo da Figura 3.2, no mapeamento das instruções do grafo de dependências da Figura 3.2(b), primeiro a instrução 1 é alocada na primeira linha da UFR, conforme a Figura 3.2(c). Como a instrução 2 é dependente do resultado de r7 gerado pela instrução 1, ela será alocada na segunda linha. A instrução 3 depende do valor de r8 gerado pela instrução 2. Já as instruções 4 e 5 podem ser executadas em paralelo com a instrução 2 na segunda linha, pois ambas dependem de r7, gerado na primeira linha pela instrução 1. A instrução 6 grava em r6 é alocada na linha 1. A instrução 7, por sua vez, utiliza o resultado de r1, gerado pela instrução 4 e deve ser alocada em uma linha acima desta, contudo, instruções de *load/store* ou de multiplicação ocupam as 3 linhas do ciclo de execução e só podem ser alocadas no início de cada ciclo, logo, a instrução 7 é alocada na linha 4. Por último, a instrução 8 depende do valor de r5 gerado pela instrução 7 ao final do segundo ciclo e é alocada na linha 7.

O número da linha em que o resultado de uma instrução mapeada na UFR se torna disponível é conhecido como a profundidade de execução da instrução. Essa profundidade nos fornece uma medida do tempo de execução na UFR até aquele momento. Do mesmo modo, a profundidade de execução máxima da UFR, também chamada de profundidade da configuração, indica o tempo de execução do bloco de instruções na UFR. No exemplo da Figura 3.2, o bloco de instruções gasta 8 ciclos

para executar no MIPS e apenas 3 ciclos para executar na UFR, sem desconsiderarmos o tempo de reconfiguração.

Em meio às unidades funcionais suportadas pelo arranjo, o grupo de unidades de *Load/Store* merece atenção especial, pois o número de unidades em paralelo nesse grupo é limitado pelo número de portas disponíveis na memória e consistência de dados.

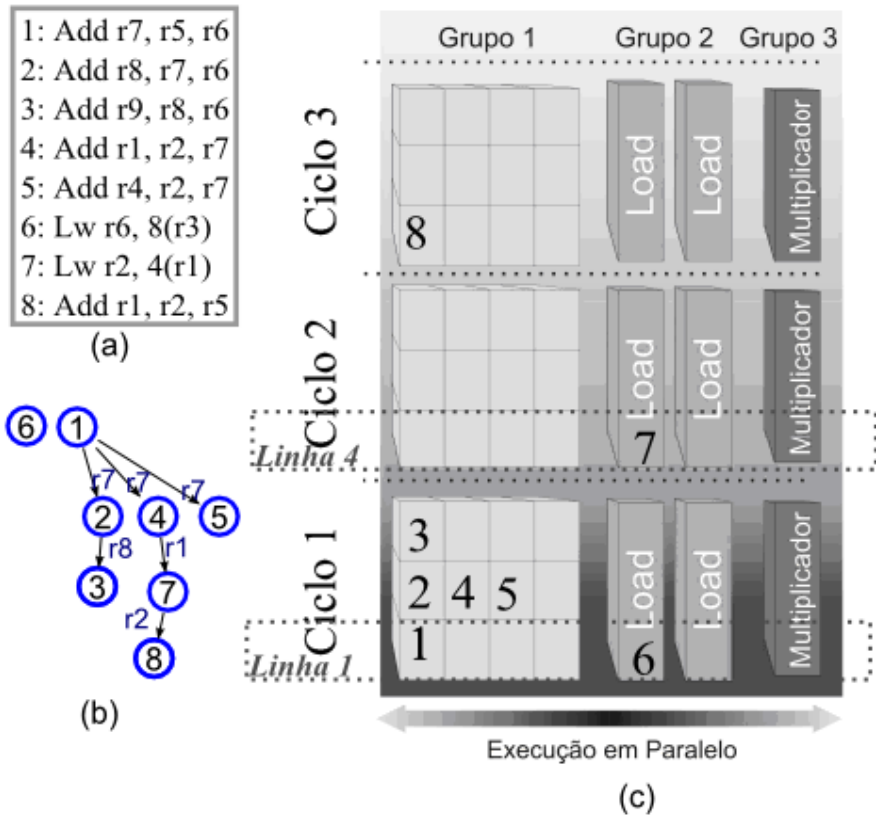


Figura 3.2. Mapeamento de instruções no arranjo bidimensional. As unidades estão distribuídas em níveis de execução que valem um ciclo cada.

Cada dado que alimenta alguma entrada de uma unidade funcional é fornecido por uma via específica do barramento principal, e pode ser proveniente de um registro do contexto de entrada ou do resultado de outra unidade funcional, em uma linha inferior, no caso de dependência de dados entre as duas unidades.

O contexto de entrada é um conjunto de registros que constituem cópia dos valores correntes do banco de registradores do processador no momento em que uma configuração é mapeada para executar na UFR. Uma vez que o MIPS R3000 tem banco de 32 registradores de 32 bits cada, também existem 32 registradores de 32

bits no contexto de entrada. Cada linha do barramento principal corresponde a um desses registros. À medida que as saídas das unidades funcionais vão produzindo resultados, estes vão sendo repassados para o barramento principal. Os resultados atualizados no barramento se tornam disponíveis para as unidades das linhas superiores. Na outra extremidade do barramento principal estão os registradores do contexto de saída que armazenam os resultados finais das vias do barramento. Esses resultados são copiados para o banco de registradores do processador ao fim da execução. A Figura 3.3 mostra o exemplo de um nível de execução de um arranjo genérico, como estão organizadas as suas unidades, estruturas de interconexão e contextos de entrada e saída.

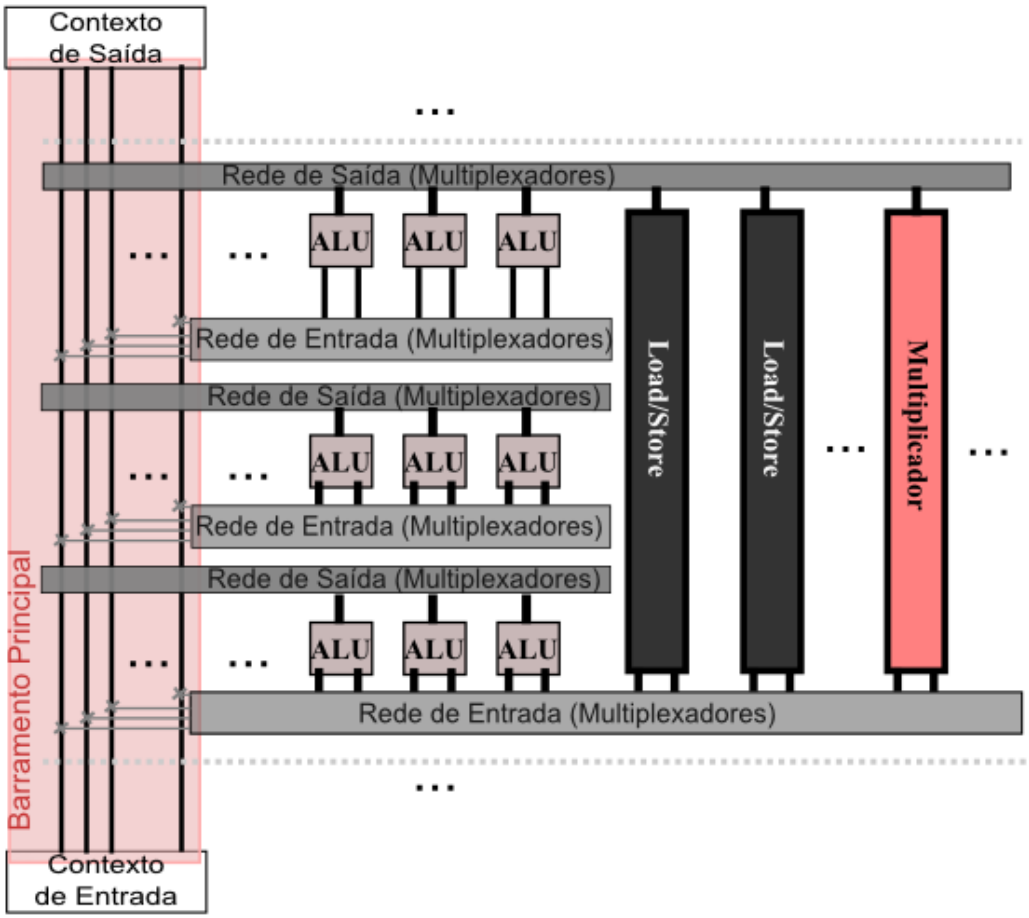


Figura 3.3. Um nível de execução genérico do arranjo bidimensional.

3.2.2 Interconexão

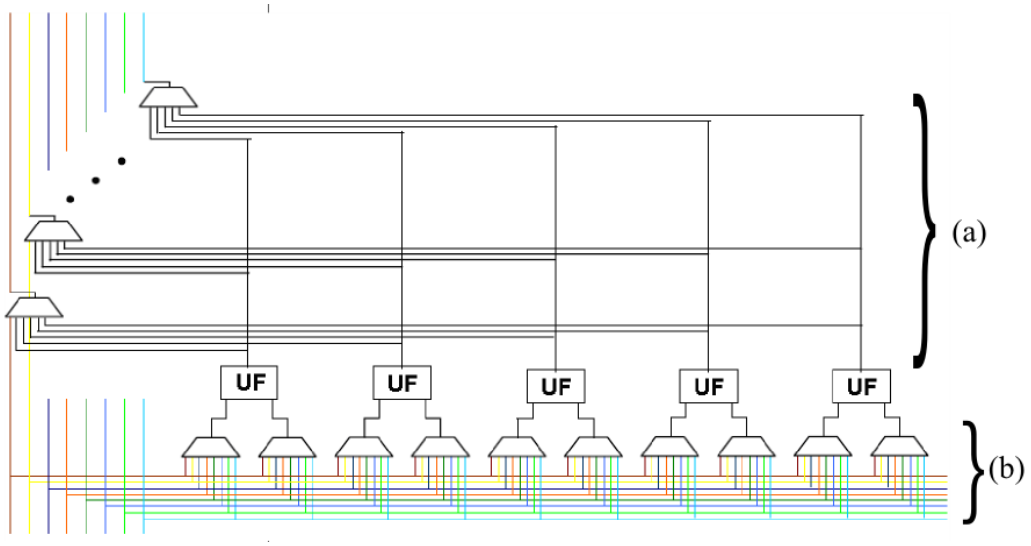
Conforme dito anteriormente, as redes de interconexão que compõem a UFR consistem no barramento principal e nas redes de multiplexadores. No barramento principal trafegam os valores (em bits) dos operandos. Suas extremidades conectam

cada registro do contexto de entrada ao seu correspondente no contexto de saída, conforme a Figura 3.3.

Ao longo do barramento principal existem duas diferentes redes de multiplexadores em cada linha da UFR que interligam os circuitos do barramento às unidades daquela linha. Chamamos essas duas redes de rede de Entrada e rede de Saída. A Figura 3.4 exibe uma abstração de como essas redes se apresentam no interior de um dos níveis da UFR, interligando as unidades e o barramento principal.

A rede de multiplexadores de Entrada faz a conexão de cada operando do barramento principal com cada uma das entradas das unidades funcionais. Quando um valor de operando é designado a uma dada unidade, o multiplexador é configurado para que o dado flua da via que o contém para a entrada requerida da unidade. A Figura 3.4(b) destaca uma rede de entrada e seus multiplexadores ligando o barramento principal às entradas das unidades de uma linha qualquer da UFR.

A rede de multiplexadores de Saída, diferentemente da rede de entrada, liga as saídas das unidades funcionais ao barramento principal. Para cada uma das vias do barramento principal existe um multiplexador que seleciona se o valor do operando correspondente é proveniente do próprio circuito do barramento ou do resultado de uma das unidades funcionais. A Figura 3.4(a) exibe uma rede de saída para uma linha qualquer da UFR. Observe que ao sobrescrever os valores de um operando do barramento principal, as unidades que caracterizam as instruções dependentes nas linhas superiores automaticamente receberão os valores atualizados.



Fonte: (BECK, 2008, p. 87)

Figura 3.4. Redes de multiplexadores em uma linha do arranjo bidimensional. (a) Rede de saída. (b) Rede de entrada

As redes de multiplexadores têm a vantagem de serem redes não bloqueantes, ou seja, capazes de perfazer qualquer conexão. Porém, possuem a desvantagem da extensa área que ocupam em relação à área total da arquitetura. Cerca de 50% da área da UFR são estruturas de interconexão. Essa área extra poderia ser melhor aproveitada com mais estruturas funcionais ou de memória, ou mesmo para criar um *chip* mais compacto. As contribuições desta dissertação estão relacionadas à redução da área ocupada pelas redes de interconexões. Para tanto utilizamos redes multiestágios, que apesar de bloqueantes, possuem baixo custo em área. Além disso, demonstramos que a aceleração da aplicação é mantida.

3.2.3 Tradução Binária

O mecanismo de tradução binária apresentado nessa subseção foi proposto em (BECK *et al.*, 2008) e é chamado de *Dynamic Instruction Merging* (DIM). De forma diferente do que acontece em outras arquiteturas reconfiguráveis, que aceleram a aplicação mapeando somente sequências específicas de código, o DIM tem a vantagem de atuar sobre toda a aplicação.

3.2.3.1 Detecção e execução

O tradutor binário monitora cada instrução que passa pelo processador agrupando-as em blocos. Cada bloco gera uma configuração da UFR. Cada configuração é armazenada em uma posição da cache de reconfiguração, indexada pelo contador de programa (PC, do inglês *Program Counter*) da primeira instrução. Ou seja, na primeira vez que aparece, o bloco irá executar no processador e ao mesmo tempo, a configuração é criada e guardada na cache. Esta é a fase de detecção. No momento em que um PC índice se repete durante a busca de uma instrução na memória, o DIM recupera a configuração da cache para ser executada na UFR, caracterizando a fase de reconfiguração.

Quando consideramos a execução especulativa, com predição de desvios, se durante a execução da configuração ocorre um erro de predição, a reconfiguração é interrompida, e uma nova configuração entra em fase detecção a partir da primeira instrução válida após o erro.

Após o término da execução na UFR o valor do contador de programa é atualizado para a instrução seguinte ao bloco. Ao mesmo tempo, o banco de

registradores é atualizado com os valores do contexto de saída e são feitas as escritas de dados na memória.

O término da configuração em tempo de detecção pode ocorrer por três formas: quando o limite de especulação é atingido; por limitações dos recursos da arquitetura ou; quando ocorre um erro de especulação de desvios. A configuração termina do modo desejado se o limite de especulação de desvios é atingido, seja ele nulo, quando não há especulação, ou com alguns blocos especulados, conforme descrito na seção 3.2.6. Quando o término acontece devido a limitações do *hardware* ou por erro de especulação, dizemos que ocorreu uma quebra. As quebras por limitações de *hardware* ocorrem devido a instruções não suportadas (ex. Divisão), falta de registradores ou falta de unidades. Uma quebra devida a uma instrução incompatível, como uma instrução de ponto flutuante ou de divisão, faz com que a instrução seja executada no processador. A quebra por limite de especulação está relacionada às instruções de desvios e será abordada na seção 3.2.6.

3.2.4 Reconfiguração e execução na UFR

Em uma execução ideal, ou seja, que não termine por meio de uma quebra, logo que uma configuração termina de executar, outra entra em fase de detecção, indexada pelo PC da instrução seguinte. Caso este PC índice aponte para uma configuração já armazenada em *cache*, a execução entra na fase de reconfiguração e a configuração é mapeada para a UFR. Do contrário, a nova configuração será montada e armazenada na *cache*. A Figura 3.5 apresenta o fluxograma do processo de execução na arquitetura.

A reconfiguração consiste na leitura dos bits de configuração das redes de interconexão e das unidades funcionais da *cache* de reconfiguração para a UFR; também são capturados e carregados no contexto de entrada: os operandos provenientes do banco de registradores e os valores imediatos armazenados em cache.

Para começar a execução da configuração no arranjo, no mínimo são necessários 3 ciclos após a captura do PC: um para encontrar a linha da cache com a configuração e dois para o processo de reconfiguração. O processador é colocado em estado de espera (*stalled*) até que a reconfiguração termine (BECK, 2008).

Durante a reconfiguração, as operações de *load/store* fazem seus acessos à memória, inclusive as que dependem do endereço calculado em tempo de execução.

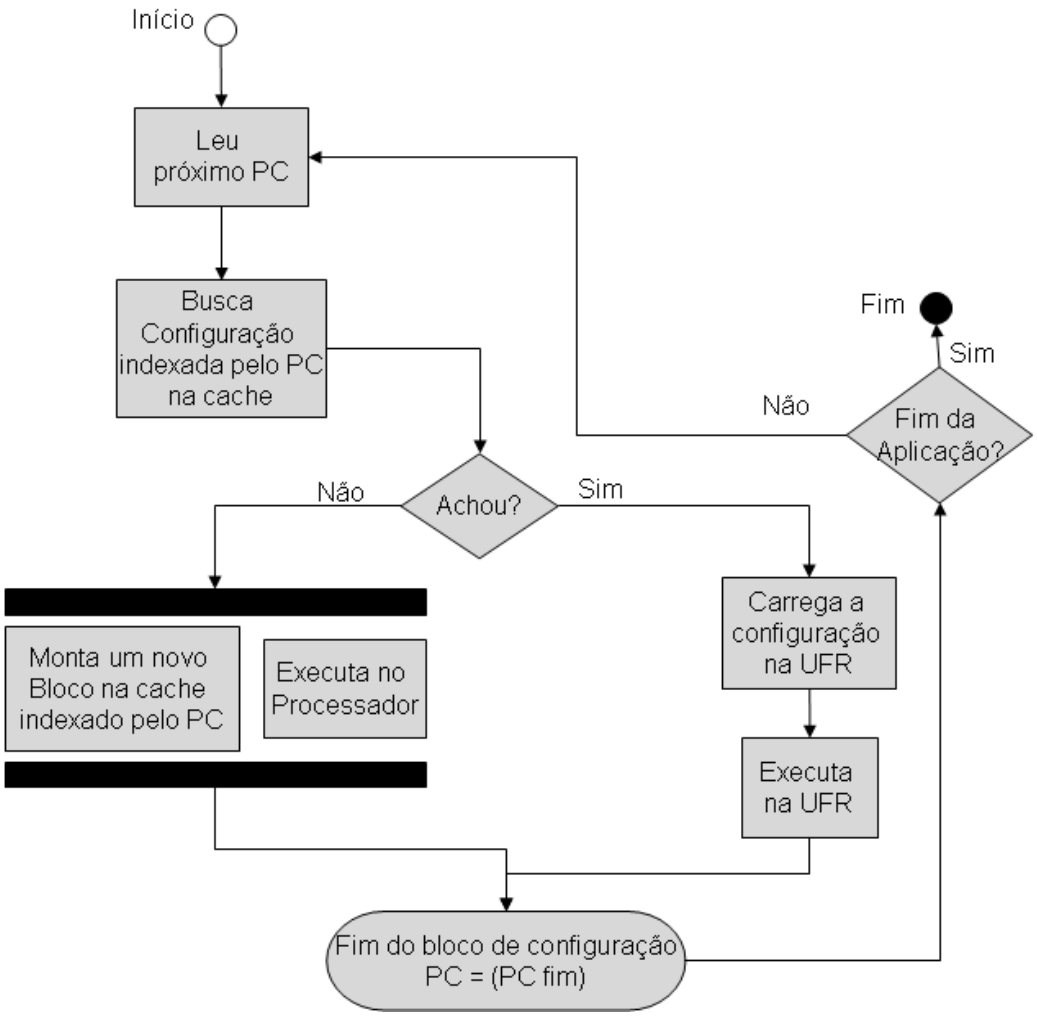


Figura 3.5. Execução da aplicação na arquitetura reconfigurável desconsiderando possíveis quebras de configuração.

3.2.5 Cache de Reconfiguração

Os três pontos chaves para a aceleração transparente da aplicação com a arquitetura reconfigurável em contexto são: a exploração do paralelismo de Instrução; a tradução binária; e o reuso de instruções. Este último ponto, segundo (SODANI; SOHI, 1998) citado por Beck (2008), baseia-se no princípio da repetição, em que uma instrução é repetida várias vezes ao longo da execução de um programa. No modelo tradicional de Von Newmman, a repetição do PC da instrução indica sua repetição, ocasionada por laços ou saltos no decorrer da aplicação. Tendo em mente que uma instrução executa várias vezes, utilizando a técnica de reuso podemos

guardar suas informações de configuração em uma memória cache para acelerar sua próxima execução.

O tradutor binário, armazena em tempo de execução, as informações necessárias para a posterior reconfiguração de um novo bloco de instruções na cache de reconfiguração. Quando o bloco é reusado, as informações são rapidamente obtidas da cache e as instruções em reuso são executadas aproveitando seus níveis de paralelismo na UFR. Isso implica na aceleração da aplicação e, ao mesmo tempo, evita a análise repetida das mesmas instruções pelo processador. Indexar a configuração pelo PC da sua primeira instrução evita buscar e decodificar as instruções seguintes. A primeira instrução do bloco é suficiente para a configuração de todo ele e, a execução das instruções é feita em paralelo, respeitando-se as dependências.

A cache utilizada nas abordagens dessa dissertação é do tipo associativa e utiliza a política de alocação FIFO (do inglês *First In, First Out*). O desempenho depende do tamanho da cache de configuração. Nos resultados apresentados em (BECK, 2008; RUTZIG, 2008) foram avaliados diferentes tamanhos de cache de reconfiguração dentre 2 a 512 linhas.

3.2.6 Execução Especulativa

A execução paralela das instruções na unidade reconfigurável é vantajosa quando existe repetição de uma configuração e o tamanho dela compense o tempo necessário para ler ou escrever os dados das operações e reconfigurar as unidades funcionais. Como veremos mais adiante (Seção 6.1), os programas normalmente tem poucas instruções de dados entre duas instruções de desvios, e nem todas elas podem executar em paralelo. Por tanto, para detectar uma configuração com tamanho que possa gerar ganhos razoáveis de desempenho, consideramos uma execução especulativa que incorpora mais de uma instrução de desvio a uma mesma configuração. Dizemos que um conjunto de instruções entre duas instruções de desvios dentro de uma mesma configuração é um bloco básico da configuração. Compor uma configuração com mais de um bloco básico ajuda a melhor aproveitar os recursos da UFR e a quantidade de instruções que executam em paralelo.

O valor da profundidade de especulação corresponde ao número de blocos especulados para montar a configuração. Para prover a especulação, o processo de tradução binária é adaptado. Uma configuração passa a ser subdividida em blocos

básicos e cada um destes blocos passa a ser indexado pelo PC de sua primeira instrução, da mesma forma como a configuração é indexada pelo PC da primeira instrução do primeiro bloco básico.

Durante a reconfiguração, a configuração é armazenada em cache é mapeada na UFR bloco por bloco. Após o mapeamento de cada um desses blocos, o PC da instrução que vier a seguir é comparado com o PC índice do bloco da próxima profundidade. Se o PC for igual ao índice, então a especulação do bloco seguinte está correta, e ele será mapeado na UFR. Caso o PC seja diferente do índice, o processo de reconfiguração é interrompido (dizemos que ocorreu erro de especulação), finalizado, e uma nova configuração entra em detecção. Vale ressaltar que os blocos anteriores, corretamente especulados, executam na UFR e deles vai depender a aceleração da aplicação.

Uma especulação certa pode significar uma aceleração significativa de parte da aplicação, por outro lado, as chances de falhas de especulação aumentam com o crescimento do número de blocos básicos especulado.

3.3 Considerações Finais

Para prover flexibilidade na execução de aplicações heterogêneas, o número de unidades por linha do arranjo se torna extenso. Isto é para evitar o crescimento da profundidade máxima do arranjo (que implica na perda de desempenho), mesmo que algumas aplicações utilizem apenas algumas poucas unidades. Nos dois capítulos seguintes, são descritas duas novas abordagens que nos permitem reduzir a área ocupada pela UFR. Na primeira abordagem, o arranjo bidimensional original tem suas redes de entrada substituídas por redes multiestágios, de menor custo em área. Na segunda abordagem, um modelo mais compacto, unidimensional, é proposto. O modelo de arquitetura unidimensional utiliza a mesma forma de reconfiguração dinâmica e transparente com as vantagens de uma economia significativa na área do componente reconfigurável, tanto em número de unidades, quanto em termos de redes de interconexão.

4 ARQUITETURA BIDIMENSIONAL COM REDES MULTIESTÁGIOS

4.1 Introdução

O modelo bidimensional de unidade reconfigurável proposto por Beck *et al.* (2008) utiliza redes de conexão baseadas em multiplexadores. Conforme dito antes, esse é um tipo de rede não bloqueante, ou seja, capaz de realizar quaisquer permutações de conexões. No entanto, a principal desvantagem deste tipo de rede é o alto custo em área, com complexidade $O(N^2)$, onde N é o número de entradas da rede.

Com o objetivo de reduzir a área da UFR bidimensional, sem perda significativa de desempenho, foi proposta a substituição das redes de multiplexadores de entrada por redes multiestágios (FERREIRA *et al.*, 2008). As redes de entrada são as responsáveis pela conexão do barramento principal com as entradas das unidades funcionais.

Para avaliar a relação área/desempenho das duas abordagens (com multiplexadores e com redes multiestágios) foi selecionado o modelo 2 de arquitetura bidimensional descrito em (RUTZIG, 2008). Essa arquitetura foi escolhida por apresentar ganhos de desempenho com 16 unidades funcionais por linha e, portanto, uma rede 32x32 para interligá-las ao barramento principal (lembrando que cada unidade funcional tem 2 entradas e o barramento principal tem 32 vias). Esta rede pode ser facilmente substituída por uma rede multiestágio 32x32, uma vez que satisfaz a condição do número de portas igual a uma potência de 2.

O uso das redes de entrada multiestágios ocasionou uma redução média de 20% da área total da UFR. Foram usadas três redes Omega 32x32 para interligar os 32 registros do Barramento principal às 24 ALUs, 6 *Load/Store* e 2 Multiplicadores de cada nível da unidade reconfigurável. Vale ressaltar que cada nível da UFR é composto por 3 linhas, onde a primeira e a segunda possuem 8 ALUs cada e a terceira possui 8 ALUs, 6 Ld/Sd e 2 multiplicadores.

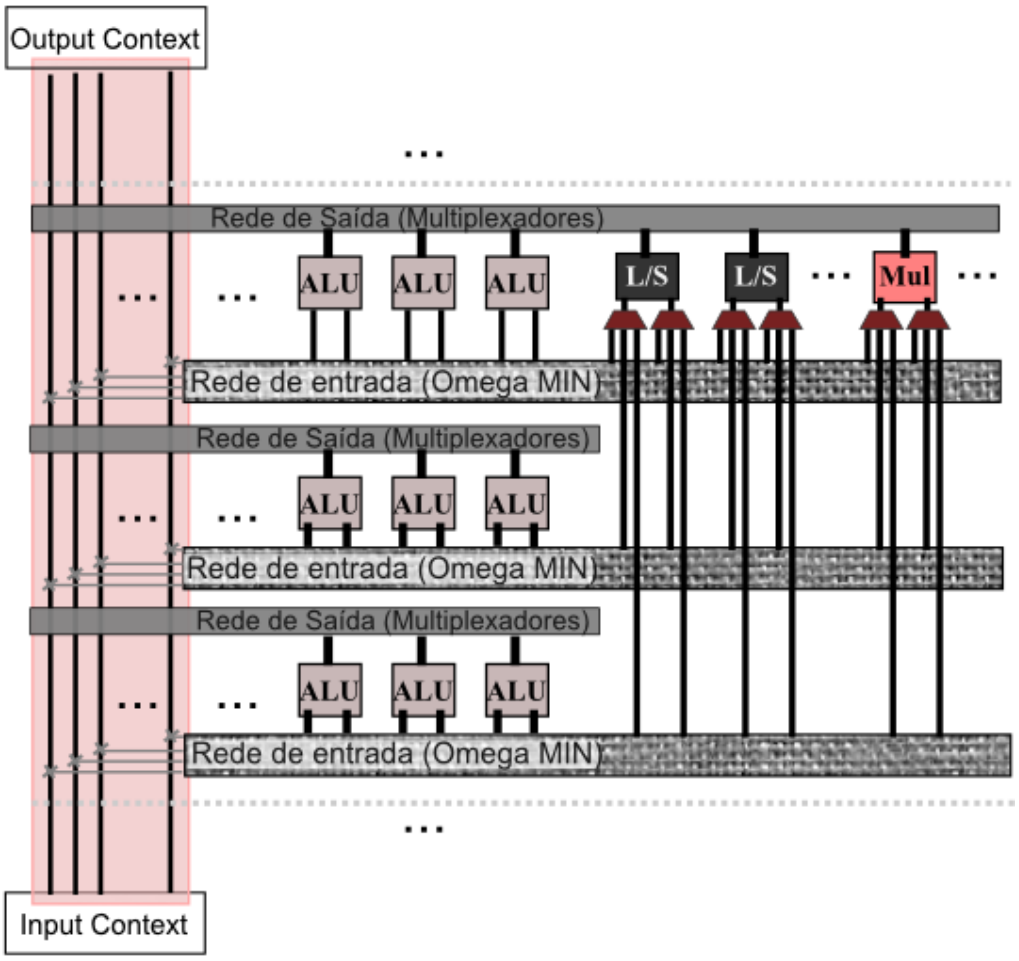


Figura 4.1. Um nível do Arranjo bidimensional com redes Omega

A Figura 4.1 apresenta um esquema genérico de um nível do arranjo bidimensional com redes Omega. Observe que as redes de entrada, antes com multiplexadores (Figura 3.3), foram trocadas por redes multiestágios. As portas de entrada destas redes estão conectadas diretamente aos circuitos do barramento principal, e as portas de saída estão conectadas aos operandos das unidades funcionais.

Para manter o mesmo número de unidades funcionais e satisfazer a restrição do número de portas igual a uma potência de 2, todas as redes multiestágios tem o mesmo número de entradas. Para ocupar as portas vagas nessas redes, cada operando de uma unidade de *Load/Store* ou multiplicador tem 3 alternativas de conexão, via multiplexadores 3:1, em cada uma das redes de entrada. Unidades lógicas aritméticas (ALU) executam em 1/3 do ciclo e estão presentes nas 3 linhas do nível de execução. A própria política de alocação das unidades pelo tradutor binário evita que unidades

de *Load/Store* ou multiplicadores recebam dados alterados pelas ALUs do mesmo nível.

4.2 Tradução Binária

A lógica de tradução binária do modelo bidimensional com redes Omega é essencialmente o mesmo do modelo com redes de multiplexadores. A reconfiguração é transparente e as tabelas de bits de cada linha são utilizadas pelo tradutor binário para montar o grafo das dependências entre as instruções. A única diferença, é que a tradução deve ser capaz de realocar instruções que não puderam ser alocadas na unidade ou na linha intencionada, devido a conflitos de roteamento, o que pode aumentar latência de reconfiguração. A cache idealizada neste trabalho é responsável em armazenar os bits de configuração das redes de interconexão (sejam elas redes de multiplexadores ou multiestágios) e das unidades funcionais, para cada configuração armazenada.

4.2.1 Posicionamento e Roteamento

A arquitetura bidimensional foi projetada para suportar diferentes cenários de execução, nos quais um grafo de dependências com grande profundidade exige mais linhas por parte do arranjo e, portanto, seu crescimento vertical. Quanto maior for o número de instruções num mesmo nível, maior deve ser a largura da linha no arranjo, a fim de maximizar o paralelismo na execução.

Para se adaptar à ocorrência de conflitos nas redes de interconexão, o algoritmo de tradução binária, em tempo de execução, faz a alocação de cada instrução levando em conta o sucesso no roteamento dos dados, além das dependências entre as instruções. Na abordagem baseada em multiplexadores, a tradução binária aloca a primeira unidade livre (e compatível) da linha. Enquanto que, na versão com multiestágios, a primeira unidade roteável é a que será alocada (não necessariamente a primeira livre). Se não houver uma unidade roteável naquela linha, as linhas seguintes serão avaliadas. Como existem muitas unidades por linha, geralmente não ocorre perda de desempenho. Eventualmente, a unidade pode ser alocada numa linha superior devido a conflitos de roteamento. Neste caso, somente se a unidade estiver no caminho crítico que determina a profundidade da configuração, o número de linhas utilizadas irá aumentar e, consequentemente, o desempenho será reduzido. Nos casos de testes da tabela 6.4 do capítulo 6, a aplicação Rijndaeld foi a que teve a

maior redução da aceleração quando executada na UFR com redes bloqueantes. Essa diferença de acelerações é reduzida nos testes em que estágios extras são acrescidos às redes Omega.

A Figura 4.2 (d) exemplifica o caso em que a impossibilidade de alocar a instrução **d** na quarta unidade da terceira linha, devido a um conflito de roteamento, não altera a profundidade de configuração em relação à abordagem com redes não bloqueantes, *vide* Figura 4.2 (c). A existência de outra unidade de mesmo tipo, livre e roteável é o que possibilita esse fato. Outro caso interessante acontece quando a instrução **e** é alocada na linha 2 ao invés da linha 1, por causa de um conflito de roteamento e da ausência de unidades livres e roteáveis na linha 1. Neste caso, a profundidade de configuração não é alterada, uma vez que essa mudança de linha não interfere no caminho crítico. Somente se **a**, **c**, **d** ou **f** mudarem de linha, ocorrerá o aumento da profundidade da configuração.

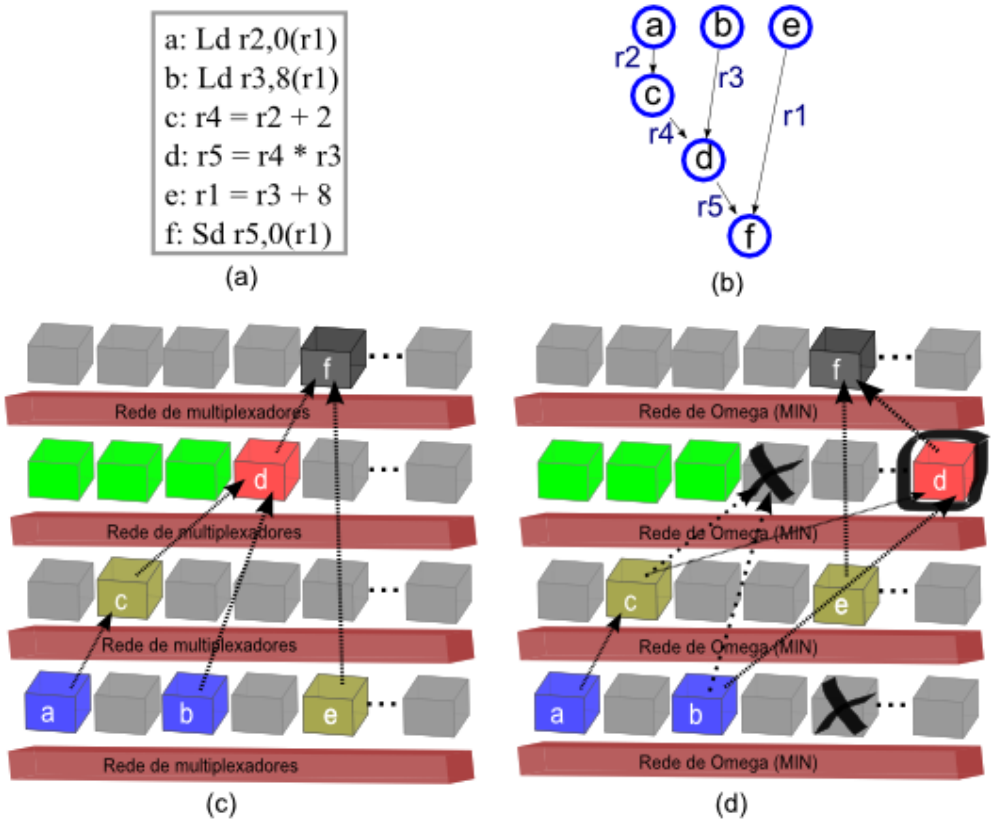


Figura 4.2. Mapeamento de um bloco de instruções nas UFRs bidimensionais.

4.2.2 Redes multiestágios na cache de Reconfiguração

Com a substituição das redes de multiplexadores por redes Omega multiestágios, a cache de configuração sofrerá algumas mudanças. Os custos em bits para a reconfiguração das unidades, dos registros e imediatos do contexto de entrada continuam sendo os mesmos. Porém, agora deverão ser armazenados os bits de configuração de cada comutador das redes de entrada multiestágio, ao invés dos bits que configuravam os multiplexadores.

A implementação em *hardware* da rede Omega com *multicast* exige dois bits de configuração para definir o estado de cada comutador (que pode estar entre os dois estágios completos ou entre os dois de broadcast). Como existem N/2 $\log_2(N)$ comutadores em uma rede Omega NxN, são necessários N $\log_2(N)$ bits para configurar a rede Omega. Uma rede de multiplexadores, por sua vez, que conecta X unidades ao barramento principal de 32 bits gasta 2X multiplexadores 32:1, e consequentemente, 10X bits de configuração (cada multiplexador 32:1 gasta 5 bits). Para N = 32 e X = 16, temos o mesmo número de unidades funcionais por linha e o mesmo custo de configuração, correspondente a 160 bits, para os modelos bidimensionais com rede de Omega ou de multiplexadores. Cada estágio extra nas redes Omega implica no acréscimo de N/2 comutadores à rede e, portanto, o custo em bits de configuração aumentará em N bits por rede. A tabela 4.1 resume os custos em bits de configuração para os dois tipos de rede, onde N é o número de portas de entrada e saída da rede multiestágios, X é o número de unidades funcionais (com dois operandos cada) ligadas à rede de multiplexadores, e K é o número de estágios extras.

Tabela 4.1. Custos de armazenamento da configuração de uma rede Omega ou de multiplexadores

Rede	Custo de configuração (em bits)
Omega	$N(\log_2 N + K)$
Multiplexadores	10X

5 ARQUITETURA UNIDIMENSIONAL COM REDES MULTIESTÁGIOS

A maioria das arquiteturas reconfiguráveis de grão grosso utiliza a topologia bidimensional de distribuição de suas unidades (HARTENSTEIN, 2001a). No entanto, os modelos convencionais bidimensionais, com unidades funcionais homogêneas, tem um alto custo para executar as várias operações de adição, multiplicação, *loads*, etc. Outra opção é o uso de unidades heterogêneas, que por sua vez, estão sujeitas a uma extensa área de interconexão e subutilização das unidades, devido às posições predeterminadas das unidades.

Com o intuito de prover uma arquitetura de grão grosso flexível e escalável, este trabalho propõe um novo modelo de arquitetura unidimensional, compacta e heterogênea. Este novo modelo não está limitado à alocação de unidades por níveis prefixados, evitando a ocorrência de sobrecarga de alguns níveis, enquanto outros estão com muitas unidades ociosas. É análogo a uma estrutura multinível, na qual o número de unidades por nível é ajustado em tempo de execução. As unidades são heterogêneas e, portanto, de menor custo e menos complexas do que unidades homogêneas. Dentre as principais vantagens do nosso modelo estão: a utilização de redes multiestágios como alternativas de interconexão global de baixo custo e o suporte do mecanismo de tradução binária para a reconfiguração transparente da aplicação. Tal como a UFR bidimensional descrita por Beck *et al.* (2008), a UFR unidimensional é fortemente acoplada ao processador e utiliza um *hardware* de tradução binária para traduzir dinamicamente as instruções para a lógica reconfigurável.

5.1 Arquitetura Proposta

A arquitetura proposta utiliza um modelo de arranjo linear (uma dimensão), que distribui as unidades funcionais em fila única, em oposição a estrutura em matriz, com linhas e colunas, dos modelos 2D em linha. Entretanto, como veremos, o

mapeamento de instruções é multinível. Isso implica que as instruções podem ser alocadas em diferentes profundidades de execução.

Para melhor compreender as diferenças entre os modelos, consideremos a sequência de instruções e seu grafo de dependências nas figuras 5.1(a) e 5.1(b). A Figura 5.1(c) apresenta um modelo 2D em linha, tal como é utilizado por diferentes abordagens de arranjos, como as descritas por Goldstein *et al.* (2000) e por Beck *et al.* (2008). A alocação de FUs se dá em diferentes linhas do arranjo, de acordo com a profundidade em que as instruções aparecem no grafo de dependências. As caixas que representam unidades utilizadas são marcadas com as letras das instruções alocadas, enquanto as demais representam as unidades livres. Em primeiro lugar, muitas unidades podem estar ociosas em cada linha, como resultado do mapeamento das dependências. Em segundo lugar, é necessária a transferência de alguns dados em vários níveis, como ocorre com os dados de r1 e r3 na Figura 5.1(c), exigindo que as estruturas de interconexão tenham a habilidade de transpassar dados entre linhas diferentes (GOLDSTEIN *et al.*, 2000; TANIGAWA *et al.*, 2008), ou façam o uso de unidades de roteamento (MEHTA, 2008).

No arranjo unidimensional proposto, o grafo de dependência é mapeado na estrutura linear composta de unidades funcionais heterogêneas e uma rede global de interconexão, conforme o exemplo da Figura 5.1(d). A rede de interconexão global deve ser capaz de satisfazer as conexões de dependências de dados entre as instruções que aparecem no grafo mapeadas nas unidades funcionais do arranjo durante a tradução binária. Todas as instruções de uma configuração são mapeadas em única linha e, ao mesmo tempo, são mantidas as relações de dependências entre elas. Uma vez que a instrução pode ser alocada em qualquer unidade livre do arranjo (desde que seja roteável e de tipo compatível), sem restrições de linhas, reduz-se o número de unidades ociosas em comparação com o arranjo bidimensional tradicional.

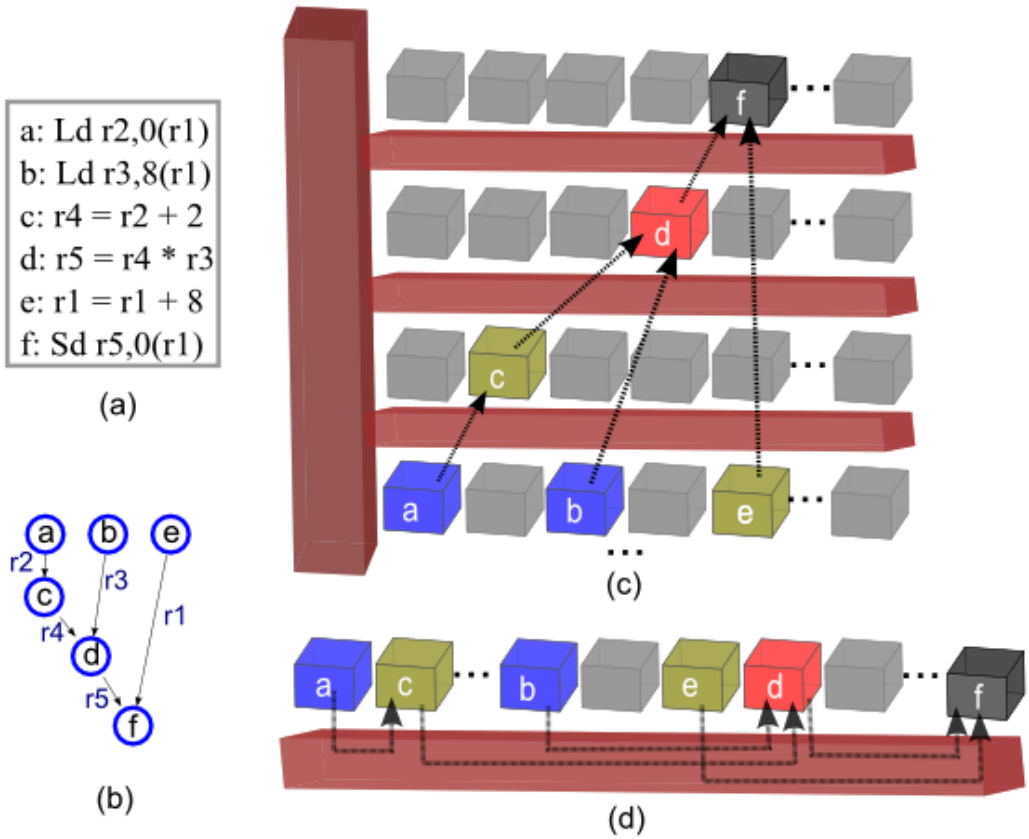


Figura 5.1. (a) Trace de instruções. (b) Grafo de dependências. (c) Mapeamento das dependências no modelo 2D. (d) Mapeamento das dependências no modelo 1D

Em todo caso, uma vez que se economiza área devido à redução do número de unidades ociosas, concentra-se a carga de ligações sobre uma única rede de interconexão. Desta forma, essa rede deve ser eficiente na transferência de dados entre as unidades funcionais e ter um custo aceitável. Neste trabalho optamos por uma rede Omega multiestágios, por sua baixa complexidade e boa conectividade.

A Figura 5.2 apresenta uma visão geral da arquitetura proposta, composta por uma unidade funcional reconfigurável (UFR) fortemente acoplada a um processador RISC (de forma semelhante à UFR bidimensional da Figura 3.1).

Basicamente, as diferenças entre a nova arquitetura 1D e a arquitetura 2D se resumem à estrutura da unidade reconfigurável e às pequenas modificações que esta acarreta na cache de reconfiguração e no processo de tradução binária.

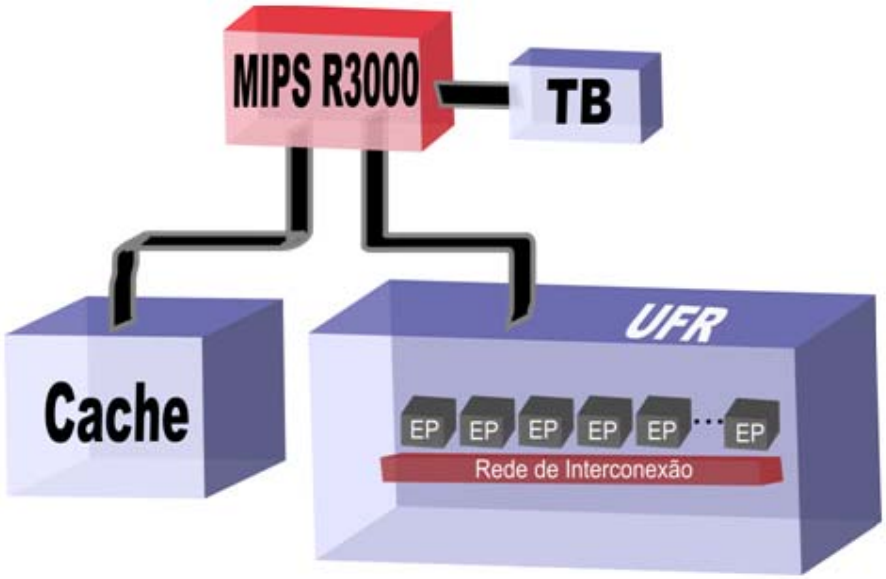


Figura 5.2. Unidade Reconfigurável Unidimensional fortemente acoplada ao processador MIPS R3000

5.2 Estrutura da UFR

Conforme descrito na Seção 5.1, o modelo de unidade funcional reconfigurável proposto é um conjunto de unidades funcionais heterogêneas dispostas em uma única linha, interligadas por uma rede global multiestágios. A UFR se comunica com o processador RISC por meio do contexto de entrada e do contexto de saída. Em tempo de reconfiguração, quando um bloco básico (*vide* Seção 3.2.4) é configurado para executar na UFR, os valores dos operandos presentes no banco de registradores da máquina RISC e os imediatos de cada instrução são copiados para os registros do contexto de entrada. Quando a execução do bloco básico finaliza, os valores modificados dos registros, presentes no contexto de saída, sobrescrevem os registros originais do banco de registradores.

5.3 Contextos de Entrada e Saída

O contexto de entrada do arranjo 1D, semelhante ao do arranjo 2D, consiste em um vetor de registradores preenchidos com os valores copiados em tempo de execução do banco de registradores da máquina RISC, ou com os imediatos copiados para a cache de reconfiguração durante o tempo de detecção. A alocação dos dados é sequencial. Quando não há mais registradores disponíveis para novos dados, ocorre a quebra do bloco de configuração atual. Caso um registro ou um imediato já esteja

alocado no contexto de entrada, o mesmo registro será utilizado pela instrução corrente. Caso contrário, um novo registro deverá ser alocado.

Cada um dos registradores do contexto de entrada está conectado a uma porta de entrada da rede de entrada do arranjo, ao lado das saídas das unidades funcionais, e por meio da rede podem ser conectados aos operandos das unidades funcionais (*vide* Figura 5.3).

O contexto de saída é um vetor de registradores preenchidos com os dados das saídas das unidades durante a execução de uma configuração. Ao término da execução na UFR, os valores dos registros do contexto de saída são copiados para o banco de registros do processador, atualizando-os. Cada um dos registradores do contexto de saída está ligado a pelo menos uma porta da rede de saída, para receber os resultados das unidades (*vide* Figura 5.3).

5.4 Interconexão

O número de portas disponíveis na rede de interconexão NxN deve ser suficiente para interligar todas as unidades funcionais e o contexto de entrada. A entrada da rede recebe as saídas das unidades funcionais e os valores do contexto de entrada. Os dados recebidos são enviados para as entradas das unidades funcionais. No modelo da UFR existe ainda uma segunda rede de interconexão multiestágios, com N' portas (sendo N' igual ao número de unidades funcionais) utilizada para interligar as saídas das unidades funcionais aos registros do contexto de saída, denominada "rede de saída". O uso da rede de saída tem por objetivo evitar toda a carga de ligações sobre uma única rede global e diminuir o número de possíveis conflitos entre as ligações de dados. A Figura 5.3 apresenta um esboço do modelo de UFR proposto para N = 256, N' = 128 e todas as unidades funcionais contendo dois operandos.

A rede de entrada, com NxN portas conecta até N/2 unidades funcionais. Por outro lado, a rede necessita somente de N/2 portas para receber as saídas das unidades (*vide* Figura 5.3). As demais portas são ligadas aos X registros do contexto de entrada, sendo assim N deve ser tal que $N \geq (N/2 + X)$. Se utilizarmos apenas uma rede no lugar das redes de entrada e de saída, seria preciso reservar também portas de saída para o contexto de saída. Para o exemplo da Figura 5.3, temos que N=256 e X=64, isso significa que somente 128 unidades funcionais podem ser usadas, contudo são utilizadas 128 + 64 = 192 portas de entrada, restando 256 - 192 = 64

portas vagas. Para obter maior vantagem da rede, utilizamos as portas vagas para criar caminhos alternativos para o contexto de entrada. Como existem N unidades funcionais e apenas 32 registros no contexto de saída (N > 32), as portas ociosas da rede de saída podem ser utilizadas como acessos alternativos ao contexto de saída, por meio do uso de multiplexadores, evitando-se o uso de estágios extras. No exemplo da Figura 5.3 cada registro do contexto de saída está ligado a quatro portas da rede via multiplexador. O encaminhamento (ou *forward*) de dados é realizado pela rede de entrada na medida em que os dados se tornam disponíveis.

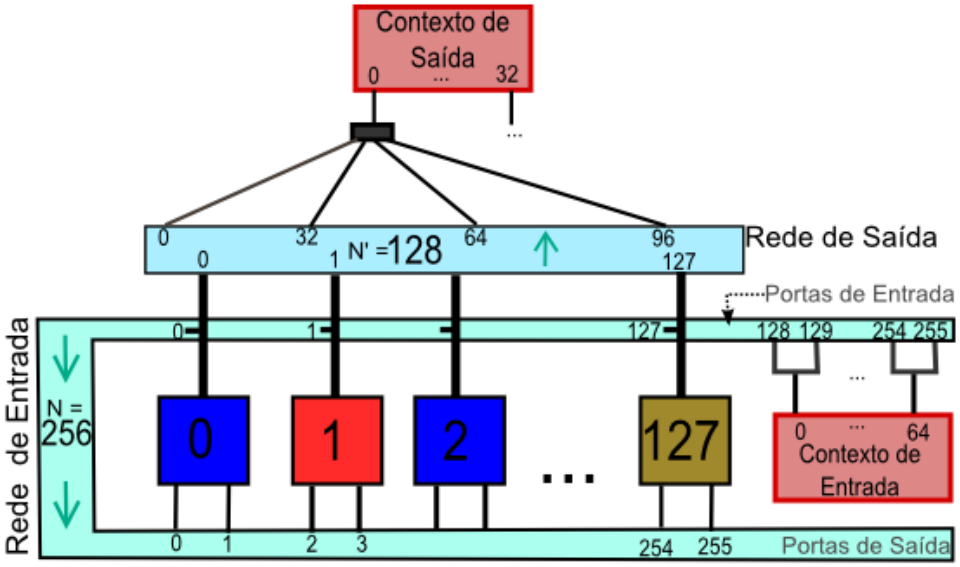


Figura 5.3. UFR unidimensional com 128 unidades funcionais

5.5 Profundidade de instrução

No modelo bidimensional as instruções são ordenadas dinamicamente em um grafo de dependências pelo tradutor binário, e alocadas na linha física do arranjo que corresponde à profundidade de execução da instrução corrente. A profundidade da instrução na UFR leva em conta as dependências com outras instruções e a disponibilidade de unidades funcionais em uma determinada linha. O tradutor binário do arranjo unidimensional, por sua vez, mapeia instruções com diferentes profundidades. A maneira como as unidades funcionais estão interconectadas pela rede de entrada é que determina a profundidade da instrução. A Figura 5.4 apresenta um exemplo da alocação de duas instruções com dependência. A instrução de

profundidade 1 ao ser alocada no arranjo, recebe o valor produzido pela unidade correspondente à instrução pai na profundidade 0.

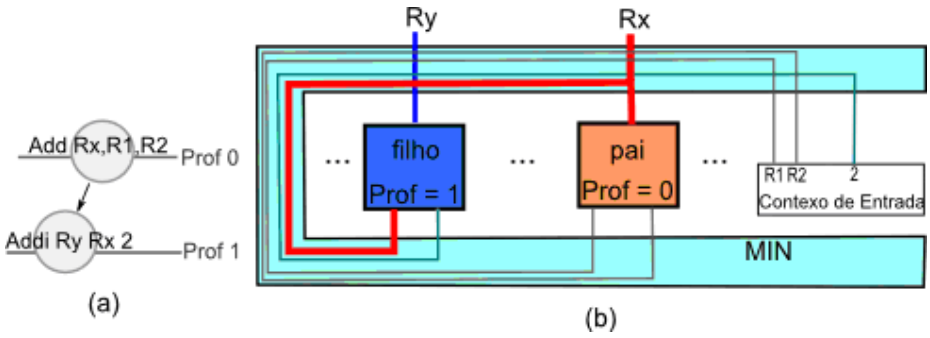


Figura 5.4. Alocação de instruções dependentes no arranjo unidimensional

5.6 Tradução binária

A lógica do mecanismo de tradução binária é a semelhante à descrita na Seção 4.2. O processo de detecção de uma nova instrução é exemplificado no fluxograma da Figura 5.5. A configuração das redes se dá com a adição de instrução por instrução, durante o tempo de detecção. Os operandos de cada instrução devem ser ligados em uma unidade funcional livre e compatível com o tipo de operação. Para tanto, as unidades são marcadas como ocupadas na medida em que são utilizadas. O subprocesso de busca de uma unidade roteável é descrito no fluxograma da Figura 5.6. Observe que os testes para saber se uma unidade é roteável acontecem em paralelo.

Durante o tempo de reconfiguração, a configuração proveniente da cache é mapeada para a UFR. Em paralelo carregam-se os valores do contexto de entrada, os bits de configuração das unidades funcionais e os bits de configuração dos comutadores das redes de interconexão. Após estes passos, a configuração está pronta para executar na UFR por meio do encaminhamento dos dados do contexto de entrada para as unidades, ou de dados entre unidades. Ao fim da execução, os registradores do banco indicados no mapa de escrita são atualizados com os respectivos registros do contexto de saída.

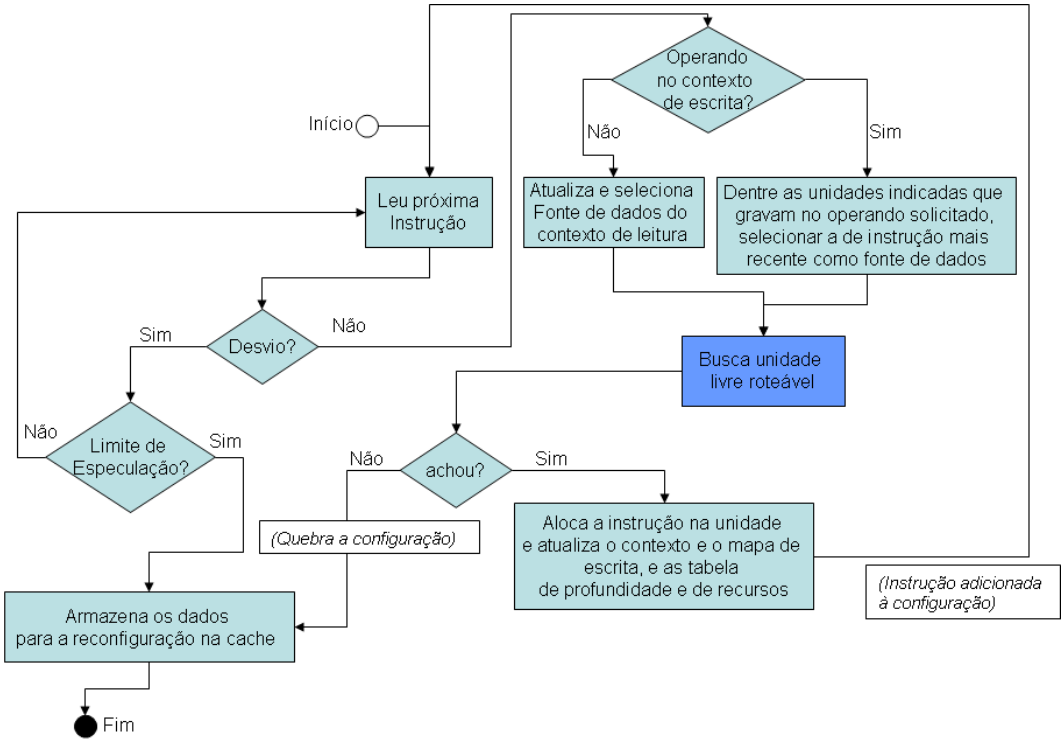


Figura 5.5. Detecção de instruções pelo tradutor binário durante a montagem de uma configuração

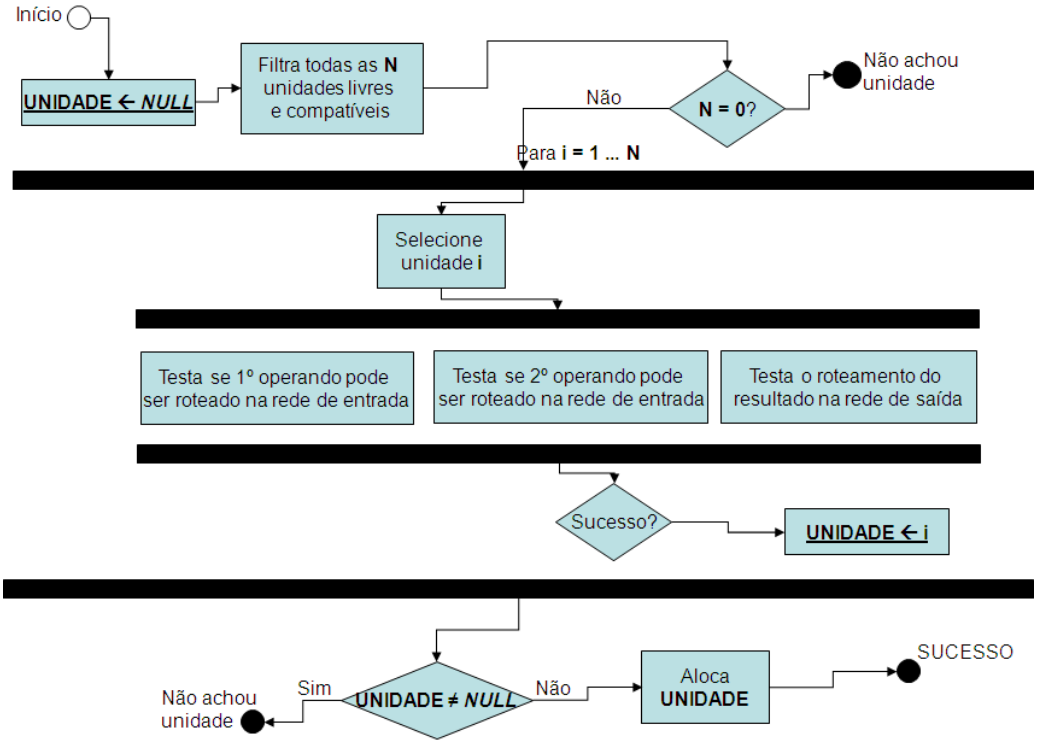


Figura 5.6. Subprocesso de busca e alocação de uma unidade funcional no arranjo 1D.

A Figura 5.7 demonstra o mapeamento de um bloco de instruções (Figura 5.7(a)) na UFR unidimensional (Figura 5.7(b)). A instrução **a** é alocada na primeira unidade funcional e recebe os valores de r2 e r1 do contexto de entrada, provenientes do banco de registradores. O resultado é enviado para o registrador correspondente a r3 no contexto de saída. A instrução **b** depende do valor de r1 proveniente do contexto de entrada e do resultado de r3 obtido da instrução **a** usando a rede de entrada. A instrução **b** é alocada na terceira unidade funcional, livre de conflitos de roteamento com a instrução **a**, e envia o resultado para r0 no contexto de saída. A instrução **c** é alocada na segunda unidade, recebe suas entradas do contexto de entrada e envia o resultado para o registrador correspondente a r0 no contexto de saída, anulando a ligação do resultado produzido pela instrução **b**. Por último, a instrução **d** recebe os valores de suas entradas do mesmo registro do contexto de entrada via *multicast* na rede de entrada, é alocada na quarta unidade e grava no registrador do contexto de saída correspondente a r0. Ao fim da execução o banco de registradores do processador é atualizado com os valores de r0, r1 e r3 do contexto de saída.

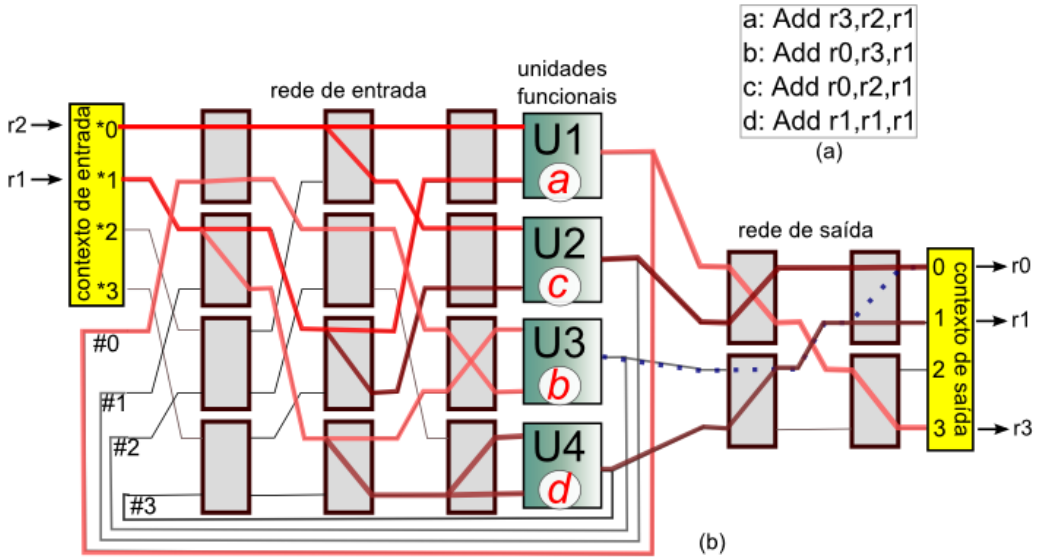


Figura 5.7. Configuração de um bloco de instruções no arranjo 1D.

5.7 Cache de Reconfiguração

De maneira análoga à cache projetada para o modelo bidimensional com redes multiestágios, a cache utilizada para a arquitetura unidimensional armazena os bits

de configuração dos comutadores das redes multiestágios, no lugar dos bits de configuração dos multiplexadores. Além, é claro, dos bits de configuração das unidades funcionais, dos valores de imediatos, dos índices dos registros a serem carregados para o contexto de entrada, e a informação de quais registradores devem ser atualizados com registros do contexto de saída. Como consideramos ainda a propagação de dados em *multicast*, dois bits de configuração são usados para definir o estado de cada comutador.

A cache continua a ser totalmente associativa, com política de alocação FIFO. E para os casos de execução especulativa, a mesma política de substituição de configurações na cache, utilizada pela arquitetura bidimensional, se aplica à arquitetura unidimensional.

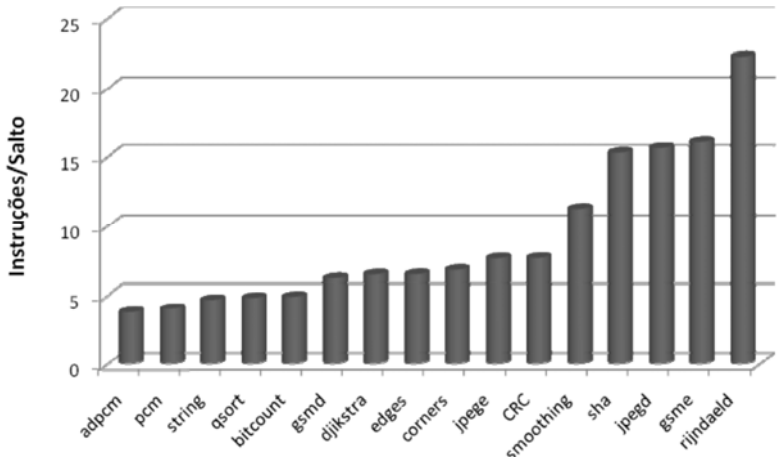
6 RESULTADOS

Neste capítulo, iremos apresentar e avaliar os resultados das simulações de execuções de um conjunto de aplicações sobre os modelos de arquiteturas propostos: bidimensional em linha com redes multiestágios e unidimensional com redes multiestágios. Como referência para comparações, utilizaremos o modelo de arquitetura proposto por Beck *et al* (2008).

6.1 Benchmarks

Para comparar os modelos de arquitetura reconfigurável, escolhemos um subconjunto de aplicações heterogêneas do *benchmark* MiBench (GUTHAUS, 2001), para comparações com trabalhos anteriores. O MiBench se divide em nichos de aplicação que incluem conjuntos específicos para os cenários: automotivo, consumidor, rede, escritório, segurança e telecomunicações. Outros dois exemplos de *benchmarks* muito utilizados na avaliação de sistemas de computação são o Mediabench (LEE *et al.*, 1997) e o SPEC 2000 (HENNING, 2000). Estes e outros benchmarks também poderiam ser utilizados para avaliar as arquiteturas.

De acordo com as peculiaridades de seus comportamentos, as operações de um benchmark podem ser classificadas em orientadas a controle ou orientadas a dados. Na Figura 6.1, é possível observar que o número médio de instruções de dados executadas por instrução de controle, para diferentes aplicações. A aplicação ADPCM (do inglês *Adaptative Differential Pulse-Code Modulation*), usada para a conversão de dados de som em informação binária, possui um comportamento mais orientado a controle. Em outro extremo, o algoritmo de criptografia Rijndaeld é o que possui um comportamento mais orientado a dados, devido a um número maior de instruções executadas entre saltos (RUTZIG, 2008). Quanto menor o número de instruções de controle, maior será o tamanho médio dos blocos básicos (*vide* seção 3.2.6).



Fonte: (RUTZIG, 2008, p. 42)

Figura 6.1. Número de instruções de dados entre instruções de desvio em diferentes aplicações do Mibench

6.2 Simulação

Para avaliar as diferentes aplicações, optamos por uma abordagem de simulação baseada no rastreio das aplicações. O rastreio é utilizado como entrada para uma ferramenta de simulação em Java, baseada no simulador utilizado em (RUTZIG, 2008) para a arquitetura bidimensional de Beck *et al.* (2008). A modelagem e simulação do comportamento de arquitetura de alto nível oferecem vantagens evidentes ao desenvolvimento de projetos de sistemas embarcados e de *hardware* complexo em geral. A ferramenta de simulação consiste na descrição em alto nível da arquitetura, modelando o comportamento do *hardware* de tradução binária, da cache de reconfiguração e da unidade reconfigurável.

Para efeito de comparação, os resultados da aceleração de cada aplicação com o uso da UFR são relativos ao desempenho do processador MIPS R3000, quando utilizado sozinho. Ou seja, uma aplicação que possui fator de aceleração 2.0 quando executa na arquitetura reconfigurável, significa que executa duas vezes mais rápida do que no MIPS escalar.

A especulação de desvios utilizada como base é de dois níveis de especulação, e o fator de confiança na especulação de uma configuração é ajustado para descartá-la sempre que houver duas falhas nas suas duas primeiras especulações, ou quando ocorrer três falhas interpoladas por um acerto, dentre as quatro últimas especulações. É importante ainda ressaltar que o tradutor binário utilizado na simulação não faz a

alocação de uma operação de *Load* e uma operação *Store* quaisquer na mesma linha, para evitar possíveis dependências de nomes na memória.

6.3 Redes Multiestágios na Arquitetura Bidimensional em Linha

No Capítulo 4, apresentamos o modelo de arquitetura bidimensional em linha utilizando redes multiestágios como rede de interconexão de entrada, entre as unidades funcionais e o barramento principal. Vimos que a utilização da rede Omega reduz significativamente a complexidade de área de interconexão em relação às redes *crossbar* ou de multiplexadores. Nesta seção, iremos discutir e avaliar os resultados dos modelos bidimensionais, objetos deste estudo.

6.3.1 Modelo de Referência

Conforme dito antes, a arquitetura reconfigurável bidimensional proposta por BECK *et al.* (2008) é utilizada como modelo de referência para este trabalho.

É utilizada a configuração 2, descrita em (BECK *et al.*, 2008; RUTIZIG; CARRO, 2008), como arranjo padrão. Este arranjo conta com 16 níveis de execução e 16 colunas, sendo que cada nível contém 2 multiplicadores, 6 unidades de *Load/Store* e 24 ALUs. A escolha desta configuração está associada à facilidade de substituição das redes de entrada por redes multiestágios. Cada nível do arranjo bidimensional corresponde a um ciclo de execução. Uma ALU consome apenas 1/3 do *clock*, logo, três linhas de ALUs são utilizadas num mesmo nível, para igualarem o atraso de execução das unidades de *Load/Store* ou de multiplicação, equivalente a um ciclo de *clock*.

6.3.2 Desempenho

O uso da UFR acoplada ao processador MIPS implica na aceleração da execução normal da aplicação, pela exploração do paralelismo entre instruções. A Figura 6.2 apresenta os valores de aceleração obtidos com as simulações da execução de aplicações do MiBench com cache de 512 blocos para três modelos: a arquitetura de referência; e as arquitetura com redes Omega sem estágios extras e com 2 estágios extras. Lembrando que os valores de aceleração se referem ao MIPS escalar e que o *software* de tradução binária atua com dois níveis de especulação de desvios.

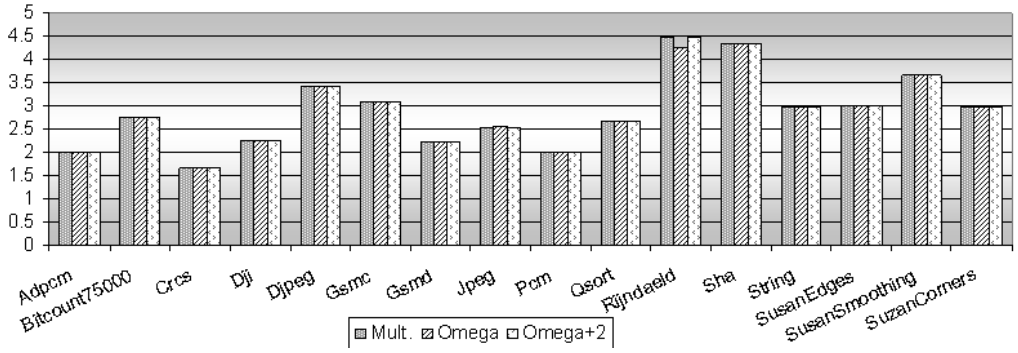


Figura 6.2. Aceleração relativa de aplicações do MiBench nos modelos de arquitetura bidimensionais em linha com diferentes redes de entrada.

É interessante perceber que embora a rede Omega seja um modelo de rede bloqueante, com capacidade de roteamento limitada, para a maioria das aplicações da Figura 6.2 apenas a utilização de redes Omega sem estágios extras é suficiente para manter o desempenho equivalente ao do modelo de referência. O número médio de unidades funcionais utilizadas no mapeamento das instruções de um bloco de configuração é menor do que 4% do total de unidades do arranjo. As unidades ociosas nas linhas do arranjo são exploradas pelo algoritmo de posicionamento e roteamento dinâmico para evitar ou reduzir o impacto causado por conflitos de roteamento. Mesmo para aplicações com comportamento mais orientado a dados como o Rijndaeld, a queda na aceleração pelo uso de redes Omega 32x32 sem estágios extras é de apenas 5%, sendo que não há redução de desempenho quando são utilizadas redes Omega com apenas 2 estágios extras.

6.3.3 Área

A Tabela 6.1, apresenta o valor estimado de área, em portas lógicas, e do atraso de propagação, sem algoritmo de roteamento, obtidos da implementação em VHDL da rede Omega 32x32 com até três estágios extras, e da rede de multiplexadores, usando o software Leonardo Spectrum (MENTOR GRAPHICS, 2009) com uma tecnologia de 180nm (FERREIRA *et al.*, 2009). Podemos observar que a rede de multiplexadores consome quatro vezes mais área do que a rede Omega sem estágios extras.

O atraso na propagação das diferentes redes de interconexão exibidos na tabela 6.1 é o maior tempo que um sinal de dados leva para atravessar a rede de uma origem a um destino, em uma rota sem conflitos. Mesmo uma rede Omega com três estágios extras é mais rápida do que a rede original de multiplexadores, enquanto que uma

rede Omega sem estágios extras chega a ser até 1,8 vezes mais rápida do que a rede de multiplexadores em condições ideais.

O atraso devido à rede Omega está relacionado ao número de estágios que os dados devem percorrer até chegarem ao destino. Assim é fácil perceber que quanto maior for a dimensão da rede ou maior o número de estágios extras, maior será o atraso. Como a rede Omega é uma estrutura regular, podemos ainda inferir que o atraso em cada estágio é equivalente a 0,16 ns. Com esse valor podemos estimar o atraso para redes maiores, como a rede de 128x128 ou 256x256 utilizadas no modelo de arquitetura 1D. Lembrando que esses são os atrasos de propagação do sinal das entradas para as saídas da rede multiestágios. A configuração dos comutadores é realizada em conjunto com a carga dos registradores no contexto de entrada e a configuração das unidades funcionais. Esta configuração gasta 3 ciclos para ser carregada da cache.

Tabela 6.1. Área (em portas lógicas) e Atraso (em nanosegundos) para redes 32x32 Omega com estágios extras, ou de multiplexadores

Rede 32x32	Omega (mais estágios extra)				MUXes
	0	1	2	3	
Área (portas)	9.619	11.543	13.467	15.391	42.642
Atraso (ns)	0,80	0,96	1,12	1,28	1,44

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

A tabela 6.2 apresenta os custos em área para as UFRs do modelo bidimensional de referência e do modelo com redes de entrada Omega 32x32, em números de portas lógicas. A área em portas lógicas obtida da descrição em VHDL do processador MIPS R3000 é de 26.882 portas. Já a descrição em VHDL do tradutor binário usado em (RUTZIG, 2008) resultou em apenas 1.024 portas. A partir destes dados, é fácil deduzir que, desconsiderando a área gasta pela cache de reconfiguração, a maior parte da área ocupada em ambas as arquiteturas é devida ao componente reconfigurável e, em especial, às redes de interconexão. A área total da UFR com redes Omega chega a ser 26% menor do que a tradicional, ao mesmo tempo em que apresentou uma redução média no desempenho de apenas 0,5%. Em adição, o hardware que faz o posicionamento e roteamento dos dados pela rede Omega tem o custo de apenas 21.000 portas e atraso de 5 ns durante a detecção de cada instrução a ser mapeada na lógica reconfigurável (FERREIRA *et al.*, 2009).

Esse atraso é bem menor do que a latência de operação do MIPS (cerca de 25 ns), e não interfere no tempo de execução do processador.

Tabela 6.2. Área (em portas lógicas) de diferentes configurações de unidades reconfiguráveis

	2D com MUXes		2D com Omega	
	número	Area	número	Area
ALU	384	642.048	384	642.048
Multiplicadores	32	214.016	32	214.016
Load/Store	96	21.888	96	21.888
Tipos de Redes	multiplexadores	Area	Omega & multiplexadores	Area
Conectores de Entrada	1024x[32:1]	1.343.488	48x[32x32] + 256x[3:1]	484.752
Conectores de Saída	512x[17:1] + 1024x[9:1]	737.280	512x[17:1] + 1024x[9:1]	737.280
Total		**2.958.720**		**2.099.984**

6.4 Arquitetura Unidimensional com redes multiestágios

O componente reconfigurável bidimensional apresenta um interessante fator de aceleração para as aplicações da Tabela 6.1, no entanto a um custo de área maior do que 100 vezes a área do MIPS escalar. Existem ainda muitas unidades ociosas por linha, não alocadas durante a reconfiguração, ocasionando subutilização dos recursos da UFR. O modelo unidimensional de arquitetura que propomos tem a principal vantagem de ser uma estrutura compacta, com a redução do número de unidades funcionais ociosas e a utilização de redes Omega como alternativa de interconexão, sem perda de desempenho.

6.4.1 Configurações de teste

O número de portas de entrada/saída das redes de interconexão devem estar de acordo com o tamanho do contexto de entrada, com o número total de entradas das unidades funcionais , com o tamanho do contexto de saída e com o número total de saídas das unidades. Para evitar toda carga de ligações sobre uma única rede, o arranjo unidimensional utiliza duas redes separadas, uma rede de entrada e uma rede de saída, tal como na Figura 5.3.

Vamos concentrar nossa atenção sobre duas configurações principais de arranjos, ambas podendo ou não conter estágios extras, com contexto de entrada de 64 registros e 32 registros de contexto de saída:

- **configuração pequena**, com rede de entrada 128x128, 64 unidades funcionais (23 *Load/Store*; 33 ALUs e 8 multiplicadores) e rede de saída 64x64;
- **configuração grande**, com rede de entrada 256x256, 128 unidades funcionais (48 *Load/Store*; 70 ALUs e 10 multiplicadores) e rede de saída 128x128;

6.4.2 Distribuição das unidades funcionais

As unidades funcionais do modelo unidimensional estão distribuídas uma ao lado da outra e dispostas em sequência alternada, ou seja, uma ALU seguida de um *Load/Store* e de um multiplicador, e assim por diante enquanto houver unidades heterogenias para alternar.

6.4.3 Quantidade de instruções nas configurações

Conforme dito antes, um bloco básico é um conjunto de instruções de dados que se encontram entre duas instruções de desvio. No entanto, devido às restrições de *hardware* da unidade reconfigurável, como o número de unidades funcionais e as limitações de roteamento das redes, o bloco básico pode finalizar antes que uma instrução de desvio seja encontrada. Em uma arquitetura ideal, com número infinito de unidades funcionais roteáveis, o bloco básico é ideal. Isso significa que nesta arquitetura, o tamanho máximo ou médio de uma configuração tende a ser maior, dependendo da aplicação e da política de especulação de desvios. Configurações maiores normalmente implicam em taxas de aceleração mais altas ao executar na UFR, o que vai depender da política de especulação e do número de reusos.

Como estamos utilizando dois níveis de especulação de desvios, teremos no máximo três blocos básicos por configuração: o bloco inicial e dois especulados.

O número médio ou máximo de instruções de uma configuração para um conjunto de aplicações nos permite avaliar se os recursos de *hardware* estão sendo bem utilizados. Busca-se avaliar o custo/benefício para evitar o desperdício de recursos, ao mapear poucas instruções, em arquiteturas grandes. Arquiteturas pequenas, por outro lado, suportam poucas instruções e poderiam apresentar melhores desempenhos se fossem maiores.

O gráfico da Figura 6.3 (a) apresenta os valores de tamanho máximo de uma configuração, em número de instruções, para diferentes aplicações do MiBench mapeadas sobre os modelos bidimensionais de referência (2D) e unidimensional

pequeno (1D 128), com rede ideal e com diferentes valores de estágios extras. A Figura 6.3 (b) apresenta análise semelhante com o modelo unidimensional grande (1D 256). Uma rede ideal é aquela em que não ocorrem conflitos de roteamento, portanto uma arquitetura com rede ideal somente depende do potencial de suas unidades funcionais e não da topologia de interconexão. Outros resultados de números máximos ou médios de instruções podem ser observados no Apêndice A. É fácil perceber que os valores de tamanho máximo de uma configuração, de uma dada aplicação, dependem da quantidade de unidades funcionais e da capacidade de roteamento do arranjo. Assim, para uma aplicação que apresenta poucas instruções por configuração, como a ADPCM, o modelo 1D com apenas 128 unidades funcionais e rede Omega global 256x256 é o suficiente para se equiparar a uma configuração grande como a do modelo 2D. Mesmo em um modelo menor, com rede de entrada 128x128 e 64 unidades funcionais, podemos obter um número máximo de instruções bem próximo do modelo 2D, adicionando 4 estágios extras à rede. Outras aplicações como o Suzan Edges e o Rinjidaeld já possuem um número máximo de instruções no modelo 2D que excede as expectativas do pequeno número de unidades funcionais ou da capacidade limitada de roteamento dos modelos 1D.

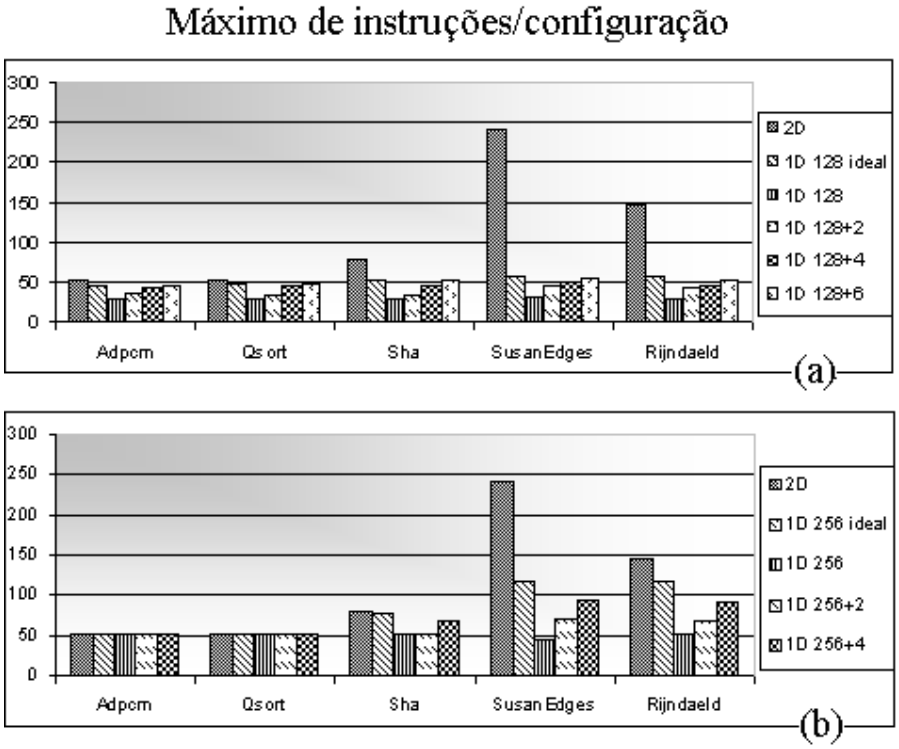


Figura 6.3. Número máximo de instruções em um bloco de configuração mapeado na UFR durante a execução da aplicação.

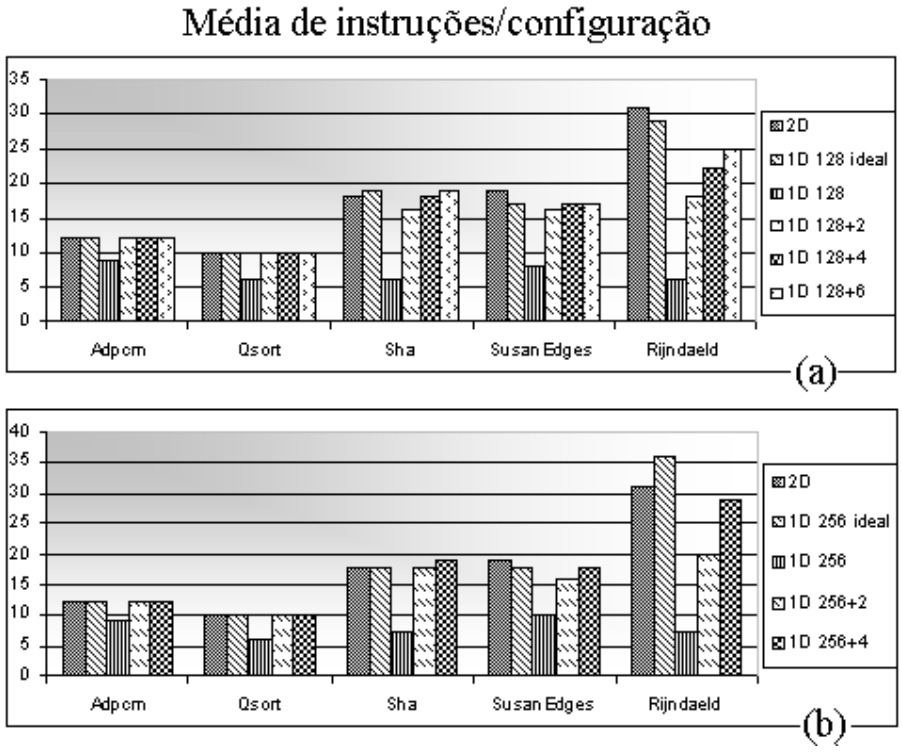


Figura 6.4. Número máximo de instruções em um bloco de configuração mapeado na UFR durante a execução da aplicação.

Os gráficos das figura 6.4 exibem os tamanhos médios de configuração das aplicações e arranjos descritos na figura. A partir desses valores é possível classificar o comportamento da aplicação. Aplicações com um número médio maior de instruções por configuração tendem a ser orientadas a dados, enquanto as que apresentam menor tamanho médio de configuração tendem a ser orientadas a controle. Uma configuração 1D grande com quatro estágios extras é o suficiente para alcançar o mesmo tamanho médio de bloco do modelo com rede ideal, para a maioria das aplicações avaliadas.

6.4.4 Desempenho

A arquitetura unidimensional tem desempenho próximo ao do modelo de referência, que tem cerca de oito vezes e quatro vezes mais unidades funcionais do que os arranjos unidimensionais, pequeno e grande, respectivamente. Esse desempenho é devido a flexibilidade do modelo 1D, e ao posicionamento e

roteamento dinâmicos, que aumenta a densidade do número de unidades funcionais utilizadas no arranjo.

Os gráficos da Figura 6.5 apresentam os valores de aceleração obtidos com as simulações da execução de aplicações do Mibench, com uma cache de 512 blocos de configurações, nos modelos de arquiteturas reconfiguráveis com arranjo bidimensional de referência; e com arranjos unidimensionais, pequeno ou grande, com ou sem estágios extras. Lembrando novamente que, em ambas as arquiteturas, os valores de aceleração são relativos ao tempo de execução no MIPS escalar, e que o software de tradução binária atua com dois níveis de especulação de desvios e fator de confiança de especulação. A aceleração é devida ao paralelismo no aproveitamento das configurações obtidas da cache.

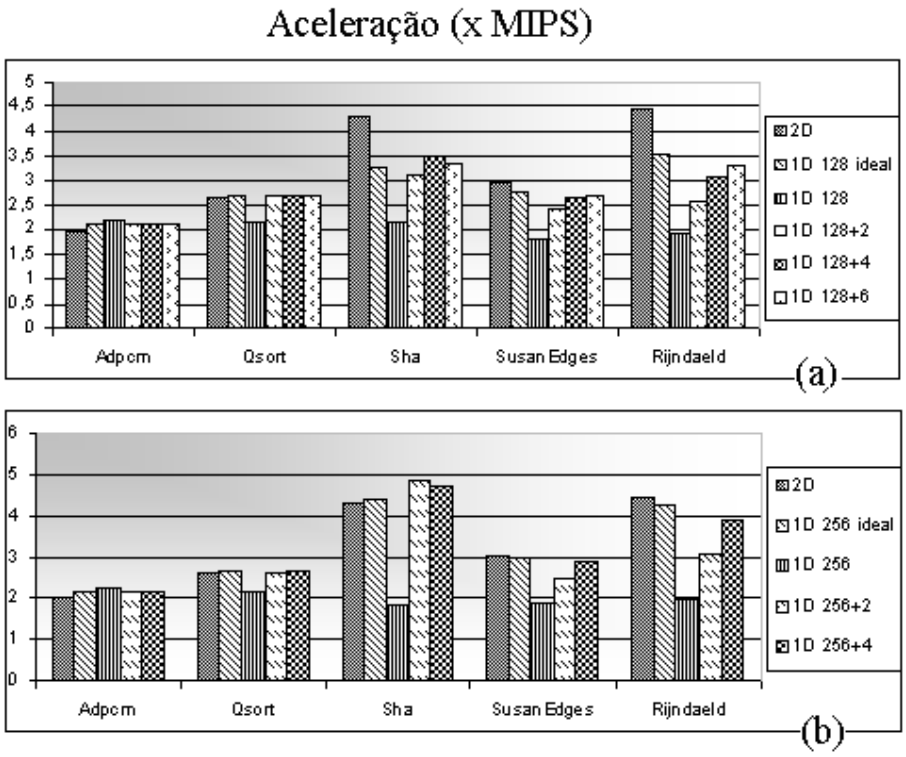


Figura 6.5. Aceleração de algumas aplicações nos arranjo bidimensional (2D), ou unidimensional com rede 128x128 (1D 128) ou 256x256 (1D 256) com diferente valores de estágios extras.

Analisando valores descritos nos gráficos da Figura 6.5, percebe-se que a arquitetura 1D pequena, com apenas 64 unidades, tem desempenho similar ao modelo 2D em aplicações orientadas a controle, como o ADPCM e o Quicksort. No entanto, o modelo pequeno apresenta uma perda de desempenho em aplicações mais orientadas a dados (Sha, Susan Edges e Rijndaeld) quando comparado ao modelo

bidimensional. Para estas aplicações, o modelo 1D grande apresenta um desempenho satisfatório e bem próximo do 2D, a medida que foram acrescentados estágios extras na rede multiestágios, resolvendo conflitos de roteamento. Um fato interessante acontece com as aplicações ADPCM e Sha, nas quais a aceleração obtida com algumas configurações do arranjo unidimensional com rede bloqueante, ultrapassou o valor da aceleração no arranjo com rede ideal. Isso ocorre porque as quebras por conflitos de roteamento fazem com que os blocos de configurações sejam diferentes do conjunto de blocos gerado para o arranjo ideal. Se os blocos são utilizados com maior frequência ao longo da execução da aplicação, uma aceleração maior é obtida.

As Tabelas 8.2 e 8.3 do Apêndice A descrevem os valores de aceleração das aplicações em diferentes configurações do arranjo 1D, utilizando caches de reconfiguração de 512 e 32 blocos respectivamente, bem como a média de aceleração de cada arranjo. No pior caso, as reduções médias de aceleração do modelo 1D com redes bloqueantes, diante do modelo 2D, foram de 34% e 32%, respectivamente. Enquanto que, no melhor caso, não houve perdas de aceleração nos arranjos com cache de 512 blocos, e uma redução de apenas 10% nos arranjos com cache de 32 blocos.

6.4.5 Área

A vantagem do modelo 1D perante o modelo bidimensional é o seu baixo custo em área. Como dito antes, a área extra do *chip* torna-se disponível para outros recursos.

Tabela 6.3. Custo em área de diferentes arranjos reconfiguráveis

	2D com MUXes		1D com 128x128		1D com 256x256	
	número	Area	número	area	número	Area
ALU	384	642.048	33	55.176	70	117.040
Multiplicadores	32	214.016	8	53.504	10	66.880
Load/Store	96	21.888	23	5.244	48	10.944
Tipos de Redes	multiplexadores	Area	Omega	Area	Omega	Area
Conectores de Entrada	1024x[32:1]	1.343.488	[128x128]	53.869	[256x256]	122.880
Conectores de Saída	512x[17:1] + 1024x[9:1]	737.280	[64x64] + 32x[2:1]	24.527	[128x128]+ 32x[4:1]	58.189
Total		**2.958.720**		**192.320**		**375.933**

O apêndice B apresenta os custos estimados de área para diferentes configurações de arquiteturas em números de portas lógicas. A Tabela 6.3 apresenta

um resumo dos dados de algumas dessas configurações: bidimensional de referência; unidimensional pequeno e unidimensional grande, ambas sem estágios extras. Os dados utilizados nesses cálculos foram sintetizados usando-se o *software* Leonardo Spectrum (MENTOR GRAPHICS, 2009), utilizando tecnologia de 180nm. Observe que UFR bidimensional é quinze vezes maior do que a unidimensional pequena, e oito vezes maior do que a unidimensional grande. Cada estágio extra que for acrescentado à rede Omega global aumenta a área da rede em 7.680 e 15.360 portas, nos modelos unidimensionais, pequeno e grande, respectivamente.

6.4.6 Tamanho da cache de reconfiguração

O tamanho da cache de reconfiguração depende da sua capacidade, em linhas (ou blocos de configuração), e do tamanho da configuração da arquitetura utilizada. A Tabela 6.4 apresenta o tamanho da cache de reconfiguração para diferentes arquiteturas e com diferentes capacidades.

Tabela 6.4. Tamanho da cache de reconfiguração para diferentes modelos de arquiteturas e diferentes capacidades de cache

Capacidade (Linhas)	Tamanho (Bytes)			
	2D com MUXes	2D com Omega	1D Pequeno	1D Grande
2	3.712	4.480	571	1.060
4	7.424	8.960	1.141	2.119
8	14.848	17.920	2.282	4.238
16	29.696	35.840	4.564	8.476
32	59.392	71.680	9.128	16.952
64	118.784	143.360	18.256	33.904
128	237.568	286.720	36.512	67.808
256	475.136	573.440	73.024	135.616
512	950.272	1.146.880	146.048	271.232

A cache de configuração utilizada para o arranjo bidimensional é semelhante a utilizada pelo modelo unidimensional. Conforme dito na Seção 5.7, a cache armazena os bits necessários para a reconfiguração das estruturas de roteamento; das unidades funcionais; dos imediatos contidos nas instruções; e das informações sobre os registros a serem carregados no contexto de entrada, e aqueles a serem enviados do contexto de saída para o banco de registradores.

A Tabela 6.5 apresenta o custo (em bits) necessário para a execução de uma configuração em diferentes modelos de unidades reconfiguráveis: bidimensional de referência, unidimensional pequeno e unidimensional grande sem estágios extras. Observe que o custo de configuração das redes de multiplexadores na arquitetura bidimensional tradicional é menor do que o custo de configuração das redes Omega

no modelo com redes multiestágios, o que explica porque o arranjo 2D com multiestágios tem um custo, em cache, 20% maior do que o tradicional. Ao compararmos os custos de configuração do modelo bidimensional tradicional com os dos modelos unidimensionais com redes multiestágios, a situação se altera. Os modelos 1D pequeno e grande tem custos de configuração 85% e 71% menores do que o custo do arranjo bidimensional tradicional, respectivamente.

Tabela 6.5. Tamanho do bloco de configuração (em bits) para diferentes modelos de unidades reconfiguráveis.

	2d com muxes		2d com 32x32		1D com 128x128		1D com 256x256	
	Número	Bits	Número	Bits	Número	Bits	Número	Bits
Contexto Entrada	32	160	32	160	32	160	32	160
imediatos	32	512	32	512	32	512	32	512
Mapa de Escrita	32	160	32	160	32	32	32	32
ALU	384	1.920	384	1.920	33	165	70	350
Multiplicadores	32	0	32	0	8	0	10	0
Load/Store	96	288	96	288	23	69	48	144
Tipos de Redes	muxes		mista		Omega		Omega	
Conectores deEntrada	1024 x [32:1]	5.120	48x[32x32] + 256x[3:1]	8.192	[128x128]	896	[256x256]	2.048
conectores de Saída	512x[17:1] + 1024x[9:1]	6.656	512x[17:1] + 1024x[9:1]	6.656	[64x64] + 32x[2:1]	416	[128x128] + 32x[4:1]	960
Total		**14.816**		**17.888**		**2.250**		**4.206**

O acréscimo de estágios extras em uma rede Omega significa um acréscimo de dois bits de configuração para cada novo comutador (FERREIRA *et al.*, 2009). Com dois estágios extras, por exemplo, o custo de configuração sobe 2.506 bits e 4.718 bits para os arranjos 1D pequeno e 1D grande, respectivamente.

6.4.7 Quebras

O arranjo unidimensional utiliza redes Omega multiestágios como estruturas de interconexão entre as unidades e os contextos de entrada e saída. Apesar do baixo custo em área em comparação com as redes *crossbar* e de multiplexadores, a rede Omega é uma rede bloqueante e, portanto, irão existir permutações de ligações que não poderão ser realizadas. O número de instruções em uma configuração pode ser reduzido em decorrência de faltas de roteamento.

Uma forma de minimizar os conflitos das redes é usar K estágios extras para obter 2^k alternativas de roteamento. Outra possibilidade é usar portas que estão sobrando na rede como fontes de caminhos alternativos, utilizando multiplexadores, tal como é demonstrado na rede de saída do arranjo da Figura 5.2, na qual, quatro

diferentes portas de saída são usadas para interligar as unidades a um dado registro do contexto de saída. Nesse mesmo modelo a rede de entrada ainda possui 64 portas de entrada disponíveis que são utilizadas como portas alternativas de acesso aos dados dos registros do contexto de entrada. O aumento da densidade de conexões na rede tende a aumentar a ocorrência de conflitos de roteamento.

O algoritmo de alocação do tradutor binário busca em paralelo por uma unidade livre e capaz de ser roteada. Se não existe unidade disponível e roteável para receber a instrução corrente, ocorre a quebra da configuração, conforme o processo descrito na Figura 5.3.

Três alternativas foram utilizadas para minimizar a quebra de uma configuração devido a conflitos de roteamento:

i. **Portas extras:** As portas que eventualmente estariam disponíveis nas redes Omega são utilizadas para prover caminhos alternativos a um recurso, que pode ser um registro ou uma unidade funcional.

ii. **Rede com estágios extras:** Para algumas aplicações o uso de uma rede de entrada sem estágios extras não é suficiente, por tanto, adicionamos novos estágios para prover novas alternativas de roteamento.

iii. **Mudança de unidade funcional:** O algoritmo de alocação procura e tenta alocar uma outra unidade funcional caso não seja possível utilizar uma dada unidade devido à conflitos de roteamento.

As tabelas 6.6(a) e 6.6(b) demonstram o número de quebras de configurações na execução das aplicações Rijindaeld e Sha na lógica reconfigurável de diferentes arranjos da arquitetura unidimensional. As quebras são divididas em dois tipos:

- **Quebras por falta de recursos:** Este tipo de quebra tem origem no número limitado de recursos do arranjo. Podem ser por falta de unidade funcional, que ocorre quando todas as unidades do tipo solicitado pela instrução corrente já foram alocadas; ou por falta de entradas, que ocorre quando não há mais registradores do contexto de entrada disponíveis para alocar um novo valor de imediato.

- **Quebras por falha de Roteamento:** Ocorre quando não há unidades roteáveis, embora ainda existam unidades livres, capazes de receber a instrução corrente. Este tipo de quebra se caracteriza pela falha em encontrar alguma unidade livre do arranjo que seja roteável.

É fácil perceber na Tabela 6.6, que quanto mais restrita for a rede, mais conflitos de roteamento tendem a ocorrer. Uma quebra por falha de roteamento pode antecipar a quebra por falta de unidade, esse fato explica o porquê de o número de quebras por falta de unidades ser tão elevado nas redes ideais, que são livres de conflitos de roteamento. Lembrando que aplicações mais orientadas a dados possuem maior densidade de instruções por configuração e, portando, estão mais suscetíveis às quebras por falha de roteamento.

As aplicações Rijindaeld e Sha apresentam um grande número de quebras por roteamento nas redes mais restritas. Isso implica na redução dos blocos de configuração e explica as perdas de aceleração descritas no gráfico da Figura 6.5, rodando sobre diferentes arquiteturas com mesmo número de unidades funcionais. O apêndice A apresenta os números de quebras dessas e de outras aplicações do MiBench.

Tabela 6.6. Números de quebras de configurações ao executar as aplicações Rijindaeld (a) e Sha (b).

Rijindaeld					
array/Quebras	Roteamento	Entradas	ALU	MULT	LD/ST
1D pequeno Ideal	0	0	79	0	8
1D pequeno	497	0	0	0	0
1D pequeno + 4	58	0	0	0	0
1D grande Ideal	0	0	36	0	3
1D grande	441	0	0	0	0
1D grande + 4	39	0	0	0	0

(a)

Sha					
array/Quebras	Roteamento	Entradas	ALU	LD/ST	MULT
1D pequeno Ideal	0	0	25	0	4
1D pequeno	313	0	0	0	0
1D pequeno + 4	16	0	3	0	3
1DGrande Ideal	0	0	1	0	0
1D grande	281	0	0	0	0
1Dgrande + 4	4	0	0	0	0

(b)

Observação: As quebras são devidas às falhas de roteamento nas redes de interconexões, ou pela ausência de unidades livres de um dado tipo.

As quebras por falta de registradores de entradas quase não ocorreram dentro do conjunto de aplicações de teste. Somente a aplicação GSMC montada sobre um arranjo com dobro das unidades funcionais da configuração grande e roteamento ideal apresentou esse tipo de quebra.

7 CONCLUSÕES

Este trabalho apresentou o uso de redes multiestágios como alternativa de interconexão de baixo custo em arquiteturas dinamicamente reconfiguráveis. Utilizando uma arquitetura semelhante à proposta por Beck *et al.* (2008) foi demonstrado que a substituição de redes de multiplexadores por redes multiestágios ocasionou uma redução considerável da área da UFR, cerca de 26%, ao mesmo tempo em que houve uma perda de apenas 0,5% no desempenho. Além disso, foi proposto um novo modelo de arquitetura, também com reconfiguração dinâmica e transparente, muito mais compacta que o modelo original, um dos arranjos apresentados chega a ser 85% menor do que o original, e com uma política de alocação de unidades mais eficiente, enquanto apresenta um bom desempenho, com uma perda média de 34% no pior caso e de apenas 10% no melhor caso (*vide* Tabela 8.3 do Apêndice A), diante do modelo 2D com redes bloqueantes.

No capítulo 3, foram discutidos os benefícios de uma arquitetura com reconfiguração dinâmica e transparente. A arquitetura de referência (BECK *et al.*, 2008) aparece nesse contexto como uma opção que alia desempenho na execução de aplicações heterogêneas e compatibilidade de software, a partir do mecanismo de tradução binária aclopado a um processador RISC.

No capítulo 4, foram propostas algumas alterações no modelo de arquitetura apresentada no capítulo 3. Essas alterações estão relacionadas à substituição de parte das redes de multiplexadores por redes multiestágios Omega. O uso das redes Omega ocasionou uma redução considerável na área total da UFR, ao mesmo tempo em que o desempenho foi mantido. No capítulo 5, foi aprofundada a combinação dos benefícios da tradução binária com o baixo custo das redes multiestágios, com a proposta de um novo modelo de arquitetura, bem menor do que a arquitetura original com multiplexadores. Mais do que isso, a possibilidade de adicionar estágios extras às redes Omega, utilizadas na interconexão, surge como opção para melhorar a conectividade e, como consequência, o desempenho do arranjo reconfigurável. O capítulo 6 descreveu o processo de simulação e os resultados das avaliações das

arquiteturas apresentadas nos capítulos 4 e 5 executando várias aplicações de natureza heterogênea.

7.1 Trabalhos futuros

7.1.1 Tolerância a Falhas

A nanotecnologia evoluiu até um patamar de circuitos com níveis de integração tão altos quanto os limites do silício. Contudo, a tecnologia em nanoescala é sujeita a grandes números de defeitos de imprecisão na fabricação. A taxa de falhas prevista chega a ser maior do que 15%, contra taxas inferiores a 0,1% das tecnologias atuais (DEHON; NAEIMI, 2005). Dessa forma, além das necessidades tradicionais por flexibilidade, desempenho e baixo consumo de energia; os dispositivos em nanoescala devem prover algum mecanismo de tolerância a falhas que viabilize sua produção. Nesse contexto, as arquitetas com redes multiestágios apresentadas neste trabalho, que já agregam o desempenho e a flexibilidade de uma reconfiguração dinâmica e transparente, podem ainda ser adaptadas para que algumas falhas em unidades funcionais ou em comutadores das redes de interconexão não prejudiquem o desempenho satisfatório do dispositivo e, viabilizem a fabricação em nanoescala (FERREIRA et al., 2010).

A chave para a tolerância a falhas nos arranjos deste trabalho estão no mecanismo de posicionamento e roteamento dinâmicos, e na flexibilidade das redes multiestágios. Embora o roteamento dos operando de uma unidade funcional ocorram na simulação de forma sequencial, o algoritmo em *hardware* da rede Omega utilizado em Ferreira *et al*. (2009) propõe as ligações de ambos os operandos *on-the-fly* em broadcast da entrada para as saída, e escolhe um dos caminhos que obtiveram sucesso de roteamento, para cada operando. Durante o sinal de *broadcast*, é possível identificar, além dos comutadores que já se encontram utilizados, unidades ou comutadores defeituosos, e automaticamente isolá-los, e em paralelo, selecionar outra unidade ou outro caminho (fornecido com a adição de estágios extras).

7.1.2 Avaliação de potência consumida

Como parte deste trabalho, foram descritos os benefícios da flexibilidade de uma reconfiguração dinâmica e transparente, capaz de beneficiar aplicações heterogêneas. Também foram apresentados os resultados em termos de desempenho para as arquiteturas avaliadas. No entanto, para que a fabricação de um dispositivo

reconfigurável seja viável à indústria de sistemas embarcados, é essencial que o dispositivo tenha um consumo de energia baixo, que se adeque à realidade da tecnologia das baterias. Rutizig (2008) apresenta a avaliação dos resultados de potência e energia consumida pelo arranjo bidimensional com redes de multiplexadores. Uma análise semelhante deve ser feita para os modelos com redes multiestágios. Técnicas como o rearranjo das unidades funcionais e a *Sleep Transistor*, apresentadas no trabalho anterior, também poderiam ser adaptadas para esses arranjos, a fim de se obter um desempenho energético mais adequado aos dispositivos a bateria.

7.1.3 Melhorias na cache de reconfiguração

Na maioria dos projetos de sistemas reconfiguráveis, os requisitos de área, potência e aceleração da unidade reconfigurável costumam ofuscar a importância de otimizações na cache de contexto. A energia e o tamanho da cache também são fatores críticos para a fabricação de sistemas embarcados.

A energia consumida pela cache pode representar até 89% da energia total consumida pelo sistema (LO *et al.*, 2010). Ao mesmo tempo, grande parte dessa energia é dissipada em função do número de bits de entrada/saída. O tamanho de palavra na cache depende da lógica e do número de recursos que definem uma configuração. Em Ló *et al.* (2010) é apresentada uma técnica que reduz a potência dissipada pelo sistema reconfigurável.

A cache utilizada nas arquiteturas deste trabalho armazena em cada palavra de configuração os bits de configuração das unidades e das redes. Esse valor se torna extenso quando levamos em conta que grande parte dos recursos do arranjo não são utilizados pela configuração, contudo ocupam bits na cache. Uma alternativa para minimizar esse problema e reduzir o tamanho da cache, reside na desfragmentação dos bits de configuração das unidades dos arranjos em partes conectadas sob a forma de uma lista encadeada. Cada partição teria sua própria rede de interconexão, que se conectaria com partições vizinhas por meio de portas extras. Essa técnica se assemelha à utilizada no armazenamento de dados em sistema de arquivos FAT (*File Allocation Table*) (LO *et al.*, 2010).

7.1.4 Exploração do Paralelismo entre configurações

Os trabalhos sobre arquiteturas reconfiguráveis atuam na aceleração de uma dada porção da aplicação por vez. Não existem abordagens que realizam a exploração de paralelismo sobre várias configurações independentes, ao mesmo tempo em várias UFRs. Uma proposta de trabalho futuro seria agregar o paralelismo no nível de instruções, que existe em cada UFR, com o paralelismo no nível de *trhead* entre UFRs (RUTIZIG, 2008).

Uma outra idéia de paralelismo entre configurações, que se aplica aos modelos 2D, é o *pipeline* de configurações em uma mesma UFR. A UFR é composta por linhas com diferentes níveis de execução. Uma configuração pode ser configurada e executar nível a nível. Isso permite que uma segunda configuração, sem dependência de dados, possa executar ao mesmo tempo, tão logo a primeira libere um nível já computado, sob a forma de um *pipeline* entre níveis de diferentes configurações. Um trabalho sobre como identificar e tratar dinamicamente as dependências de dados entre as configurações seria interessante nesse sentido.

REFERÊNCIAS BIBLIOGRÁFICAS

BACON, D. F.; GRAHAM, S. L.; SHARP O. J.: **Compiler Transformations for High-Performance Computing**, *ACM Computing Surveys*, New York, v. 26, n. 4, p. 345-420, Dez. 1994.

BAUGH L.; ZILLES C.: **Decomposing the Load-Store Queue by Function for Power Reduction and Scalability**. *In P=ac¾ Conference, IBM Research*, Out. 2004.

BECK, A. C. S.: **Transparent Reconfigurable Accelerator for Heterogeneous Behavior Systems**. *Doctoral thesis, PPGC, UFRGS*, 2008.

BECK, A. C. S.; RUTZIG, M. B.; GAYDADJIEV, G. N.; CARRO L.: **Transparent reconfigurable acceleration for heterogeneous embedded applications**. *In proceedings of Design, Automation and Test in Europe 2008 (DATE 08)*, Mar. 2008.

BECK, A.S. GOMES, V.F. CARRO, L.: **Automatic Dataflow Execution with Reconfiguration and Dynamic Instruction Merging**. *In Very Large Scale Integration, 2006 IFIP International Conference*, p. 30-35, Out. 2006.

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..

BENES, V.: **Mathematical Theory of Connecting Networks**, Academic Press, N.Y., 1965

BENINI, L.; MICHELI, G.; **Networks on Chips: A New SoC Paradigm**. *IEE Computer*, Jan. 2002.

CARDOSO, J.; VESTIAS, M.: **Architectures and compilers to support reconfigurable computing**. *ACM Crossroads*, 5(3):15-22, Mar. 1999.

CARDOSO, J. M. P.: **Self Loop Pipelining and Reconfigurable Dataflow Arrays.** *In International Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS IV)*, Samos, Grecia, Jul. 19-21, 2004

ÇAM, H.; FORTES, J. A. B.: **Work-Efficient Routing Algorithms for Rearrangeable Symmetrical Networks**. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 7, p. 733-741, Jul. 1999.

CHEN, D. C..: **Programmable arithmetic devices for high speed digital signal processing**. *Electronic ResearchLaboratory,* Memorandum No. UCB/ERL M92/49, University of California, Berkeley, May 1992

CHEN, D. C.; RABAEY, J.: **Reconfigurable Multi-processor IC for Rapid Prototyping of Algorithmic-Specific High-Speed Datapaths**. *IEEE Journal of Solid-State Circuits*, V. 27, No. 12, Dec 1992.

CLARK, N.; KUDLUR, M.; PARK, H.; MAHLKE, S.; FLAUTNER, K.: **Application-specific processing on a general-purpose core via transparent instruction set customization**. *In MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, p. 30–40, 2004.

CLOS, C.: **A Study of Nonblocking Switching Networks**. *Bell System Tech. J*., Vol. 32, pp. 406-424, 1953.

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

DEHON, A.: **DPGA utilization and application**. *In Proceedings of the ACM/SIGDA International Symposium on FPGAs*, p.115-121, 1996.

DEHON, A. and NAEIMI, H. **Seven Strategies for Tolerating Highly Defective Fabrication**. *IEEE Design and Test* 22(4), 306315(2005)

EBELING, C.; CRONQUIST, D.C.; FRANKLIN, P.: **Rapid - reconfigurable pipelined datapath**. *In: FPL '96: Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, London, UK, Springer-Verlag, p. 126–135, 1996

ERNST, R.: **Codesign of embedded systems: Status and trends**. *Design and Test of Computers*, Santa Barbara, v. 15, n. 2, p. 45 – 53, Abr. 1998.

FENG, T-Y.: **A Survey of Interconnection Networks**. *Computer*, vol. 14, no. 12, p. 12-27, Dec. 1981, doi:10.1109/C-M.1981.220290, 1981.

FENG, T.-Y.; SEO, S.-W.: **A new routing algorithm for a class of rearrangeable networks**. *IEEE Transactions on Computers*, vol. 43, no. 11, p. 1270–1280, 1994.

FERREIRA, R.; BUENO, C.; LAURE, M.; PEREIRA, M.; CARRO, L.: **A Dynamic Reconfigurable Super-VLIW Architecture for a Faul Tolerant Nanoscale Design.** In: *4th HiPEAC Workshop on Reconfigurable Computing (WCR),* Italy, January, 2010.

FERREIRA, R.; LAURE, M.; BECK, A. C.; THIAGO LO; RUTZIG, M.; CARRO, L: **A low cost and adaptable routing network for reconfigurable systems**. *In Proceedings of the 2009 IEEE international Symposium on Parallel&Distributed Processing*, 2009.

FERREIRA, R.; LAURE, M.; RUTZIG, M.; BECK, A. C.; CARRO, L.: **Reducing Interconnection Cost in Coarse-Grained Dynamic Computing Through Multistage Network**, *Proceedings of Field-Programmable Logic (FPL08)*, Heidelberg, Alemanha, Set. 2008.

FLYNN M. J.; HUNG P.: **Microprocessor design issues: thoughts on the road ahead**, *in IEEE Micro*. vol. 25, pp. 16-31, 2005.

FREITAS, M. E.:**Arquiteturas Superescalares**. *Fundação CPqD Centro de Pesquisa e Desenvolvimento em Telecomunicações*, Novembro de 2005.

GAJSKI D.; VAHID F., NARAYAN S.; GONG J.; **SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design**. *In IEEE Trans. on VLSI Systems*. vol. 6, 1998, pp. 84-100.

GAZIT, I.: **On the number of permutations performable by extra-stage multistage interconnection networks**. *IEEE Trans. Comput.*, vol. 38, no. 2, pp. 297–302, 1989.

GOKE, L. R.; LIPOVSKI, G. J.: **Banyan Networks for Partitioning Multiprocessing Systems**. *Proc. First Annual Computer Architecture Conf.*, pp. 21-28, Dec. 1973.

GOLDSTEIN, S. C.; SCHMIT, H.; BUDIU, M.; CADAMBI, S.; MOE, M.; TAYLOR, R. **PipeRench: A Reconfigurable Architecture and Compiler**. *IEEE Computer*, vol. 33, No. 4, 2000.

GRAHAM, P.; NELSON, B. **Genetic Algorithms In Software and In Hardware--A Performance Analysis of Workstations and Custom Computing Machine Implementations**. *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 216-225, 1996.

GRAMMATIKAKIS, M. D.; HSU, F.D.; HRAETZL, M.: **Parallel System Interconnection and Communications.** *CRC Press*, 2001.

GUPTA R. K.; MICHELI G. D. **Hardware-software co-synthesis for digital systems**. *In IEEE Design and Test of Computers*. vol. 10, 1993, pp. 29-41.

GUTHAUS, M. R.; RINGENBERG, J. S.; ERNST, D.; AUSTIN, T. M.; MUDGE, T.; BROWN, R. B. **Mibench: A free, commercially representative embedded benchmark suite**. *In WWC '01: Proceedings of the Workload Characterization*, 2001. WWC-4. 2001 IEEE International Workshop. Washington, DC, USA: IEEE Computer Society, pp. 3–14, 2001.

HARTENSTEIN, R.: **A decade of reconfigurable computing: a visionary retrospective**. *Proceedings of the conference on Design, automation and test in Europe (DATE 01)*, Munich, Germany, pp 642-649, 2001a.

HARTENSTEIN, R.: **Coarse Grain Reconfigurable Architectures**. *In Proceedings of the 2001 conference on Asia South Pacific design automation*, Yokohama, Japan, Pages: 564 – 570, 2001b.

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

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, 1994, pp. 404-414.

HAUCK, S.; FRY, T. W.; HOSLER, M. M.; KAO, J. P.: **The Chimaera reconfigurable functional unit**. *IEEE Symposium on Field-Programmable Custom Computing Machines*, p. 87 – 96, 1997.

HAUCK, S. **Configuration prefetch for single context reconfigurable coprocessors**. *In Proceedings of the ACM/SIGDA International Symposium on FPGAs*, pages 65-74, 1998.

HAUCK, S.; LI, Z.; SCHWABE, E.: **Configuration compression for the Xilinx XC6200 FPGA.** *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, p. 138-146, 1998.

HAUSER, J. R.; WAWRZYNEK J. **Garp: A MIPS Processor with a Reconfigurable Coprocessor**. *In: FPGA-BASED CUSTOM COMPUTING MACHINES*, 1997, Napa Valley. Proceedings… Washington: IEEE Computer Society, 1997, 12-21

HUANG, W. J.; SAXENA, N; MCCLUSKEY, E. J.: **A Reliable LZ Data Compressor on Reconfigurable Coprocessors.** *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 249-258, 2000.

HWANG, K.; BRIGGS, F. A.: **Computer Architecture and Parallel Processing**. New York: McGraw-Hill, Inc., 1990.

HENNESSY, J.; PATTERSON, D.: **Computer Architecture: A Quantitative Approach**, 4th edition, Morgan Kauffman, San Francisco, 2007.

HENNING; J. L. **SPEC CPU2000: measuring cpu performance in the new millennium**. *IEEE Computer*, p. 28–35, Jul. 2000.

HUR J.Y.; STEFANOV T.; WONG S.; VASSILIADIS S.: **Customizing Reconfigurable On-Chip Crossbar Scheduler**. *In IEEE 18th International Conference on Application-specific Syst*ems, Architectures and Processors (ASAP07), Montreal, Canada, Jul. 2007

INTEL: **Pentium 4 Homepage**. Disponível em: <http://www.intel.com/products/processor/pentium4/index.htm> Acesso em Dez. 2008.

KHALID, M.A.S.; ROSE, J.: **Experimental Evaluation of Mesh and Partial Crossbar Routing Architectures for Multi-FPGA Systems**. *Proceedings of the Sixth IFIP International Workshop on Logic and Architecture Synthesis (IWLAS'97)*, 1997

KOREN, I. Et al.: **A Data-Driven VLSI Array for Arbitrary Algorithms**. *In IEEE Computer*, p. 30-43, Out. 1988.

LARUS, J.: **Spending Moore's Dividend**. **Microsoft Tech Report MSR-TR-2008-69**, Maio de 2008

LAWRIE D. H., **Access and Alignment of Data in an Array Processor**, *IEEE Trans. Computers*, Vol. C-24, No. 12, pp. 1145-1155, Dec. 1975.

LEE, C.; POTKONJAK, M.; MANGIONE-SMITH, W. H. **MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems**. *In Proceedings of the 30th Annual International Symposium on Microarchitecture (Micro-30)*, dez. 1997.

LI, Z.; COMPTON, K.; HAUCK, S. **Configuration caching management techniques for reconfigurable computing**. *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, p. 22-36, 2000.

LO, T. B.; BECK, A. C. S.; RUTZIG, M. B.; CARRO, L.: **Decreasing the Impact of the Context Memory on Reconfigurable Architectures.** In: *4th HiPEAC Workshop on Reconfigurable Computing (WCR),* Italy, January, 2010.

LUDOVICI, D.; GILABERT, F.; MEDARDONI, S.; REQUENA, C. J.; GÓMEZ, M. E.; LÓPEZ, P.; Bertozzi, D.; Gaydadjiev, G. N.: **Assessing Fat-Tree Topologies for Regular Network-on-Chip Design under Nanoscale Technology Constraints**. *In proceedings of Design, Automation and Test in Europe 2009 (DATE 09)*, pp. 562-565, France 2009.

LYSECKY, R.; STITT, G.; VAHID, F.: **Warp Processors**. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, pp. 659-681, July 2006

MEHTA, G., IHRIG, C., JONES, A.: **Reducing energy by exploring heterogeneity in a coarse-grain fabric**. *In: IEEE RAW Reconfigurable Architecture WorkShop*, 2008.

MEI, B.; LAMBRECHTS, A; MIGNOLET, J-Y.; VERKERST, D.; LAUWEREINS, R.: **Architecture Exploration for a Reconfigurable Architecture Template**. *IEEE CS and the IEEE CASS*, 2005.

MENTOR GRAPHICS. **Leonardo Spectrum**. Disponível em: <http://www.mentor.com/products/fpga/synthesis/leonardo_spectrum>, 2009.

MEFTALI, F.; DEKEYSER, J-L.; SCHERSON, I. D.: **Scalable Multistage Networks for Multiprocessor System-on-Chip Design**. *In Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN),* 2005.

MIYAMORI, T.; OLUKOTUN, K.: **REMARC: Reconfigiurable Multimedia Array Coprocessor**. *In Proc. ACM/SIGDA FPGA '98*, Monterey, 1998.

MOORE, G. E.: **Cramming more components onto integrated circuits**. *Electronics*, 38(8):114–117, April 19, 1965

MOREANO, N. B.: **Algoritmos para Alocação de Recursos em Arquiteturas Reconfiguráveis.** *Doctoral thesis.* Instituto de computação. Universidade Estadual de campinas, Nov. 2005.

OLUKOTUN, K; HAMMOND, L.: **The future of microprocessors**. *ACM Queue*, 3(7):26–29, 2005.

PACT XPP TECHNOLOGIES, INC.: **The XPP White Paper**, release 2.1.1, March 2002, <http://www.pactxpp.com>.

RAMANATHAN, R.: **Intel multi-core processors: Making the move to quad-core and beyond**. White Paper, Intel Corporation, 2006.

RIGO, S.; ARAUJO, D.; BARTHOLOMEU, M.; AZEVEDO, R.: **Archc: A systemc-based architecture description language**. *In SBACPAD'04: Proceedings on Computer Architecture and High Performance Computing*. IEEE Computer Society, 2004.

RUTZIG, M.; BECK, A. C. S.; CARRO, L. **Balancing Rconfigurable Data Path Resources According to Applications Requirements.** *In: RECONFIGURABLE ARCHITECTURE WORKSHOP*, RAW, 2008, Miami.

RUTIZIG, M.: **Gerenciamento Automático de Recursos Reconfiguráveis Visando a Redução de Área e do Consumo de Potência em Dispositivos Embarcados**, *Master thesis, PPGC, UFRGS*, 2008.

SANKARALINGAM, K.; NAGARAJAN, R.; LIU, H; KIM, C.; HUH, J.; RANGANATHAN, N.; BURGER, D; KECKLER, S. W.; MCDONALD, R. G.; MOORE, C. R.: **Trips: A polymorphous architecture for exploiting ilp, tlp, and dlp**, *ACM Transactions on Architecture and Code Optimization*, New York,v.1, n. 1, p. 62 – 93, May 2004.

SCHMIT, H.; WHELIHAN, D.; TSAI, A.; MOE, M.; LEVINE, B.; TAYLOR, R.R.: **Piperench: A virtualized programmable datapath in 0.18 micron technology**. *In: Custom Integrated Circuits Conference*, 2002.

SETHUMANDAVAN Et al.: **Scalable Hardware Memory Disambiguatiofor high ilp processors**. *In Proceedings of the 36th Annual IEEE/ACMInternational Symposium on Microarchitecture*, Dez. 2003.

SHEN, X.; ZHANG, Y., **Partitionability of k-extra-stage omega networks and an optimal task migration algorithm**. *J. Parallel Distrib. Comput.*, vol. 60, no. 3, pp. 334–348, 2000.

SHIRAZI, N.; LUK, W.; CHEUNG, P.: **Automating production of run-time reconfigurable designs**. *In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 147-156, 1998.

SHIMIZU, S.: **Self-routing multistage switching network for fast packet switching system**. United States Oki Electric Industry Co., Ltd. (Tokyo, JP), 1990

SIMA D.; FALK H.: **Decisive aspects in the evolution of microprocessors** , pp. 1896-1926, 2004.

SINGH, H.; LEE, M. H.; LU, G.; BAGHERZADEH, N.; KURDAHI, F. J.; FILHO, E. M. C. **Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications**. *IEEE Trans. Comput.*, vol. 49, no. 5, 2000.

SWANSON, S.; MICHELSON, K.; SCHWERIN, A.; OSKIN, M.: **Wavescalar**. *In MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, p. 291, 2003.

TAU, E.; CHEN, D.; ESLICK, I.; BROWN, J.; DEHON, A.: **A First Generation DPGA Implementation.** *In FPD'95, Canadian Workshop of Field-Prog. Devices*, May 1995

TANIGAWA, K.; ZUYAMA, T.; UCHIDA, T.; HIRONAKA, T.: **Exploring compact design on high throughput coarse grained reconfigurable architectures**, *in FPL* '08: *Proceedings of the International Workshop on Field-Programmable Logic*. London, UK: Springer-Verlag, pp. 126–135, 2008.

TIAN, H.; KATANGUR, A.; ZHONG, J.; PAN, Y.: **A novel multistage network architecture with multicast and broadcast capability**. *The Journal of Supercomputing*, vol. 35, p. 277–300(24), Mar. 2006.

VARGHESE, J., BUTTS, M., AND BATCHELLER, J.: **An efficient logic emulation system**. *IEEE Trans. VLSI Syst*. 1, 2, 171-174, 1993.

VENKATARAMANI G.; NAJJAR W.; KURDAHI F.; BAGHERZADEH N.; BOHM W.: **A Compiler Framework for Mapping Applications to a Coarse-grained Reconfigurable Computer Architecture.** *In Conf. on Compiler, Architecture and Synthesis for Embedded Systems*, 2001.

YEH, Y. M.; FENG, T-Y: **On a class of rearrangeable networks**, *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1361– 1379, 1992.

WU, C.; FENG, T.: **On a Class of Multistage Interconnection Networks**. *IEEE Trans. Computers*, Vol. C-29, No. 8, Ag. 1980, p. 694-702.

APÊNDICE A – COMPORTAMENTO DE EXECUÇÃO

A.1 Total de configurações geradas em diferentes arranjos

Tabela 8.1. Número total de blocos configurações gerados na execução de uma dada aplicação

-	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners
2D	148724	259	163	2003	1761	5622	680	22597	93009	9908	174	168	263	12016	12541	5472
1D 128 ideal	148722	264	164	2003	1919	5693	689	22217	93007	9909	246	186	267	12075	12537	5511
1D 256 ideal	148724	259	163	2003	1644	5607	680	22636	93009	9908	196	168	263	12065	12541	5486
1D 512 ideal	148724	259	163	2003	1638	5638	678	22588	93009	9908	164	168	263	12105	12541	5550
1D 128	105777	468	314	1617	2692	7592	963	8388	101117	7018	510	331	477	8126	8433	4527
1D 128+2	148671	296	174	2856	2154	5845	709	22706	93013	10007	263	182	268	12773	15741	6422
1D 128+4	148725	267	164	2002	2054	5763	691	21897	93010	9911	216	177	260	12280	12924	5707
1D 128+6	148725	268	164	2000	2212	5731	684	22090	93010	9911	219	191	260	12335	12881	5477
1D 256	152830	455	286	1980	2529	7607	1118	10648	101331	6951	468	308	464	8058	12893	4324
1D 256+2	148670	277	167	2854	1559	5689	705	22145	93012	9944	224	178	252	12294	13446	5937
1D 256+4	148724	259	163	2003	1735	5606	680	22662	93009	9908	190	171	263	12122	12542	5450
1D 512	152830	456	286	1980	2518	7579	1107	10641	101331	6951	467	308	463	8000	12357	4242
1D 512+2	148670	277	167	2854	1475	5736	728	22192	93012	9944	193	172	252	12242	13444	6034

Observação: Os nomes das arquiteturas devem ser lidos conforme os exemplos: **1)** 1D 128 ideal → Arquitetura unidimensional com rede de entrada ideal Omega 128x128 (64 unidades funcionais); **2)** 1D 512+2 → Arquitetura unidimensional com rede de entrada Omega 512x512 (256 unidades funcionais) utilizando 2 estágios extras.

A.1 Aceleração em diferentes arranjos

Tabela 8.2. Aceleração das aplicações nos arranjos com cache de 512 blocos

Cache 512	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	2	2,74	1,65	2,26	3,4	3,1	2,23	2,52	2	2,65	4,47	4,33	2,97	3	3,65	2,96	*2,87*
1D 128 ideal	2,14	2,71	1,65	2,36	2,91	2,64	2,36	2,3	2,1	2,7	3,55	3,28	3,03	2,8	3,66	2,78	*2,68*
1D 256 ideal	2,14	2,74	1,65	2,36	3,5	3,01	2,36	2,49	2,1	2,7	4,29	4,4	3,03	2,99	3,66	2,98	*2,9*
1D 512 ideal	2,14	2,74	1,65	2,36	3,72	3,27	2,36	2,6	2,1	2,7	4,85	4,42	3,03	3,05	3,66	3,06	*2,98*
1D 64+4	2,14	2,43	0	2,63	2,14	2,21	2,19	2,09	2,09	3,13	2,17	2,81	3,03	2,26	1,88	2,41	*2,23*
1D 128	2,23	1,4	1,77	1,98	1,91	2,19	2,03	1,74	2,2	2,17	1,94	2,16	2,28	1,83	1,42	2	*1,95*
1D 128+2	2,14	2,53	1,65	2,08	2,26	2,4	2,3	2,18	2,1	2,69	2,56	3,11	3,02	2,43	2,53	2,54	*2,41*
1D 128+4	2,14	2,71	1,65	2,36	2,73	2,63	2,36	2,26	2,1	2,7	3,06	3,51	3,03	2,67	3,34	2,69	*2,62*
1D 128+6	2,14	2,79	1,65	2,36	2,78	2,64	2,36	2,25	2,1	2,7	3,31	3,33	3,03	2,72	3,35	2,77	*2,64*
1D 256	2,24	1,5	1,77	2	1,95	2,13	2,1	1,9	2,23	2,16	1,96	1,83	2,29	1,87	1,42	2,02	*1,96*
1D 256+2	2,14	2,62	1,65	2,08	3,04	2,6	2,35	2,3	2,1	2,66	3,09	4,83	3,05	2,48	3,41	2,6	*2,69*
1D 256+4	2,14	2,74	1,65	2,36	3,32	2,91	2,36	2,41	2,1	2,7	3,89	4,72	3,03	2,9	3,66	2,94	*2,86*
1D 512	2,24	1,47	1,77	2	1,99	2,16	2,1	1,9	2,23	2,16	1,95	1,83	2,29	1,87	1,42	2,03	*1,96*
1D 512+2	2,14	2,62	1,65	2,08	3,25	3	2,36	2,4	2,1	2,66	3,99	4,48	3,05	2,56	3,64	2,63	*2,79*

Observação: Número de unidades nos modelos: **1)** 1D 128 → 23 LD/SD, 33 ALU, 8 MULT; **2)** 1D 256 → 48 LD/SD, 70 ALU, 10 MULT; **3)** 1D 512 → 80 LD/SD, 156 ALU, 20 MULT

Tabela 8.3. Aceleração das aplicações nos arranjos com cache de 32 blocos

Cache 512	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	2	2,74	1,65	2,24	2,92	2,6	2,11	2,46	2	2,62	4,05	4,33	1,95	1,69	3,1	1,74	*2,5*
1D 128 ideal	2,14	2,71	1,65	2,34	2,3	2,39	2,23	1,93	2,1	2,67	1,08	3,25	1,96	1,62	3,11	1,61	*2,2*
1D 256 ideal	2,14	2,74	1,65	2,34	3,04	2,67	2,23	1,95	2,1	2,67	3,89	4,4	1,96	1,6	3,11	1,59	*2,5*
1D 512 ideal	2,14	2,74	1,65	2,34	3,21	2,72	2,23	2,55	2,1	2,67	4,78	4,42	1,96	1,68	3,11	1,8	*2,6*
1D 64+4	2,14	2,43	0	2,61	1,95	1,33	2,07	1,93	2,09	2,09	1,08	2,8	1,7	1,46	1,7	1,55	*1,8*
1D 128	2,23	1,4	1,77	1,83	1,76	1,31	1,93	1,63	2,2	1,7	1,1	2,16	1,52	1,32	1,32	1,37	*1,7*
1D 128+2	2,14	2,53	1,65	2,06	2,02	2,23	2,17	2,08	2,1	2,66	1,07	3,1	1,93	1,46	2,22	1,57	*2,1*
1D 128+4	2,14	2,7	1,65	2,34	2,41	2,24	2,23	2,11	2,1	2,67	1,08	3,49	1,96	1,59	2,87	1,66	*2,2*
1D 128+6	2,14	2,78	1,65	2,34	2,43	2,42	2,23	2,02	2,1	2,67	1,07	1,7	1,96	1,56	2,89	1,67	*2,1*
1D 256	2,24	1,5	1,77	1,86	1,78	1,29	2,01	1,76	2,23	1,7	1,1	1,7	1,51	1,32	1,3	1,36	*1,7*
1D 256+2	2,14	2,62	1,65	2,07	2,46	2,38	2,22	2,04	2,1	2,62	1,07	4,79	1,91	1,5	2,93	1,63	*2,3*
1D 256+4	2,14	2,74	1,65	2,34	2,9	2,56	2,23	2,07	2,1	2,67	1,07	4,71	1,96	1,65	3,11	1,73	*2,4*
1D 512	2,24	1,47	1,77	1,86	1,81	1,29	2,01	1,76	2,23	1,7	1,1	1,7	1,51	1,32	1,31	1,36	*1,7*
1D 512+2	2,14	2,62	1,65	2,07	2,78	2,7	2,23	1,94	2,1	2,62	1,07	4,48	1,91	1,53	3,1	1,62	*2,3*

A.2 Quebras de configurações no modelo 1D

Tabela 8.4. Número de quebras por roteamento

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	148724	259	163	2003	1761	5622	680	22597	93009	9908	174	168	263	12016	12541	5472	*19710*
1D 128 ideal	148722	264	164	2003	1919	5693	689	22217	93007	9909	246	186	267	12075	12537	5511	*19713*
1D 256 ideal	148724	259	163	2003	1644	5607	680	22636	93009	9908	196	168	263	12065	12541	5486	*19710*
1D 512 ideal	148724	259	163	2003	1638	5638	678	22588	93009	9908	164	168	263	12105	12541	5550	*19712*
1D 128	105777	468	314	1617	2692	7592	963	8388	101117	7018	510	331	477	8126	8433	4527	*16147*
1D 128+2	148671	296	174	2856	2154	5845	709	22706	93013	10007	263	182	268	12773	15741	6422	*20130*
1D 128+4	148725	267	164	2002	2054	5763	691	21897	93010	9911	216	177	260	12280	12924	5707	*19753*
1D 128+6	148725	268	164	2000	2212	5731	684	22090	93010	9911	219	191	260	12335	12881	5477	*19760*
1D 256	152830	455	286	1980	2529	7607	1118	10648	101331	6951	468	308	464	8058	12893	4324	*19516*
1D 256+2	148670	277	167	2854	1559	5689	705	22145	93012	9944	224	178	252	12294	13446	5937	*19835*
1D 256+4	148724	259	163	2003	1735	5606	680	22662	93009	9908	190	171	263	12122	12542	5450	*19718*
1D 512	152830	456	286	1980	2518	7579	1107	10641	101331	6951	467	308	463	8000	12357	4242	*19470*
1D 512+2	148670	277	167	2854	1475	5736	728	22192	93012	9944	193	172	252	12242	13444	6034	*19837*

Observação: A quebra de uma configuração por roteamento ocorrem quando não existem mais unidades roteáveis capazes de receber uma dada instrução.

Tabela 8.2. Número de quebras por falta de ALU

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	0	0	0	0	195	92	2	328	0	0	79	25	0	228	2	17	*61*
1D 128 ideal	0	0	0	0	62	10	0	84	0	0	36	1	0	35	0	16	*15*
1D 256 ideal	0	0	0	0	56	1	0	13	0	0	12	0	0	6	0	0	*5,5*
1D 512 ideal	0	0	0	0	0	1	0	0	0	0	1	0	0	17	1	1	*1,3*
1D 128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 128+2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	*0,1*
1D 128+4	0	0	0	0	11	0	0	3	0	0	0	3	0	2	0	0	*1,2*
1D 128+6	0	0	0	0	52	26	0	23	0	0	1	18	0	4	1	9	*8,4*
1D 256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 256+2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 256+4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	*0,1*
1D 512	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 512+2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*

Observação: A quebra por falta de unidades acontece se todas às unidades do tipo especificado já foram utilizadas no arranjo.

Tabela 8.5. Número de quebras por falta de Load/Store

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	1	2	1	1	109	68	16	78	1	1	8	4	5	50	0	290	*40*
1D 128 ideal	0	0	0	0	0	27	7	5	0	0	3	0	0	208	0	242	*31*
1D 256 ideal	0	0	0	0	0	39	4	0	0	0	2	0	0	271	0	104	*26*
1D 512 ideal	7	12	0	12	54	23	23	83	7	11	9	6	9	15	17	17	*19*
1D 128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 128+2	0	0	0	0	14	4	5	20	0	0	0	0	2	1	0	1	*2,9*
1D 128+4	0	1	0	0	43	22	5	47	0	0	0	3	4	1	0	3	*8,1*
1D 128+6	1	2	1	1	86	29	11	51	1	1	1	4	5	6	0	13	*13*
1D 256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 256+2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	*0,1*
1D 256+4	0	0	0	0	0	5	3	4	0	0	0	0	0	0	0	0	*0,8*
1D 512	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 512+2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*

Tabela 8.6. Número de quebras por falta de Multiplicador

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	0	0	0	0	0	27	0	0	0	0	0	0	5	0	0	0	*2*
1D 128 ideal	0	0	0	0	0	75	0	0	0	0	0	0	0	0	0	0	*4,7*
1D 256 ideal	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	*1,1*
1D 512 ideal	0	2	0	0	11	99	0	9	0	29	0	0	9	66	17	14	*16*
1D 128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 128+2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	*0,1*
1D 128+4	0	0	0	0	0	9	0	0	0	0	0	0	4	0	0	0	*0,8*
1D 128+6	0	0	0	0	0	12	0	0	0	0	0	0	5	0	0	0	*1,1*
1D 256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 256+2	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	*0,1*
1D 256+4	0	0	0	0	0	31	0	0	0	0	0	0	0	0	0	0	*1,9*
1D 512	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*
1D 512+2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*0*

A.3 Tamanho máximo e médio de configuração

Tabela 8.7. Tamanho máximo de bloco de configuração (em nº de instruções) em diferentes arranjos

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	51	51	51	51	200	173	85	316	51	51	146	80	51	241	60	130	*112*
1D 128 ideal	44	47	47	47	56	60	55	56	44	47	57	51	47	58	48	58	*51*
1D 256 ideal	51	51	51	51	111	124	90	118	51	51	117	78	51	117	60	119	*81*
1D 512 ideal	51	51	51	51	206	214	155	196	51	51	236	80	51	239	60	169	*120*
1D 64+4	24	24	0	25	29	28	26	28	24	26	25	26	24	31	27	28	*25*
1D 128	30	29	29	29	29	30	31	32	30	29	29	30	29	32	30	30	*30*
1D 128+2	36	39	35	35	44	44	38	43	36	35	41	35	42	44	44	45	*40*
1D 128+4	42	44	44	44	49	55	45	47	42	44	44	44	44	49	46	52	*46*
1D 128+6	44	47	47	47	54	56	49	54	44	47	52	51	47	54	48	56	*50*
1D 256	51	51	51	51	55	42	61	55	51	51	51	51	51	44	42	42	*50*
1D 256+2	51	51	51	51	75	71	72	68	51	51	68	52	51	72	60	67	*60*
1D 256+4	51	51	51	51	88	90	82	88	51	51	90	68	51	93	57	93	*69*
1D 512	51	51	51	51	78	61	74	56	51	51	51	51	51	51	51	66	*56*
1D 512+2	51	51	51	51	110	146	132	117	51	51	132	66	51	127	60	119	*85*

Tabela 8.8. Tamanho médio de bloco de configuração (em nº de instruções) em diferentes arranjos

	Adpcm	Bitcount	Crcs	Dji	Djpeg	Gsmc	Gsmd	Jpeg	Pcm	Qsort	Rijndaeld	Sha	String	Edges	Smoothing	Corners	**Média**
2D	12	16	16	19	36	18	17	18	9	10	31	18	15	19	21	22	*19*
1D 128 ideal	12	16	16	19	27	16	17	18	9	10	29	19	15	17	21	19	*18*
1D 256 ideal	12	16	16	19	27	17	17	18	9	10	36	18	15	18	21	21	*18*
1D 512 ideal	12	16	16	19	30	18	17	18	9	10	38	18	15	20	21	25	*19*
1D 64+4	12	10	0	13	15	10	12	12	9	9	12	11	11	14	14	13	*11*
1D 128	9	6	6	9	8	8	8	8	7	6	6	6	7	8	6	6	*7,1*
1D 128+2	12	15	15	16	23	15	16	16	9	10	18	16	15	16	18	16	*15*
1D 128+4	12	16	16	19	24	16	17	17	9	10	22	18	15	17	21	18	*17*
1D 128+6	12	16	16	19	25	16	17	17	9	10	25	19	15	17	21	18	*17*
1D 256	9	8	7	10	11	8	8	9	8	6	7	7	8	10	11	9	*8,5*
1D 256+2	12	16	16	16	25	16	16	17	9	10	20	18	15	16	22	19	*16*
1D 256+4	12	16	16	19	27	16	17	18	9	10	29	19	15	18	21	19	*18*
1D 512	9	7	7	10	11	8	8	9	8	6	7	7	8	10	11	10	*8,5*
1D 512+2	12	16	16	16	26	17	17	17	9	10	26	18	15	17	22	19	*17*

APÊNDICE B – ÁREA ESTIMADA DA UNIDADE RECONFIGURÁVEL

B.1 Área das UFRs bidimensionais

Tabela 8.9. Área das unidades funcionais no modelo 2D

Unidade	Número	Área
ALU	384	642.048
Multiplicador	32	214.016
Load/Store	96	21.888
Soma:		877.952

Tabela 8.10. Área de interconexão no modelo 2D com multiplexadores

	Tipos de redes	Número	Área
Redes de Entrada:	Muxes 32x32 Muxes 32x16	16 32	1.343.488
Redes de Saida:	Muxes 17x32 Muxes 9x32	16 32	737.280
Total de interconexão:			2.080.768
Área Total do arranjo:			**2.958.720**

Tabela 8.10. Área de interconexão no modelo 2D com redes Omega 32x32

	Tipos de redes	Número	Área
Redes de Entrada:	Omega 32x32 Muxes 3:1	48 256	484.752
Redes de Saida:	Muxes 17x32 Muxes 9x32	16 32	737.280
Total de interconexão:			1.222.032
Área Total do arranjo:			**2.099.984**

Tabela 8.11. Área de interconexão no modelo 2D com redes Omega 32x32 + 2 estágios extras

	Tipos de redes	Número	Área
Redes de Entrada:	Omega 32x32+2 Muxes 3:1	48 256	670.896
Redes de Saida:	Muxes 17x32 Muxes 9x32	16 32	737.280
Total de interconexão:			1.408.176
Área Total do arranjo:			**2.286.128**

B.2 Área das UFRs unidimensionionais

Tabela 8.11. Área no arranjo 1D com rede de entrada128x128

Unidade	Número	Área
ALU	33	55.176
Multiplicador	8	53.504
Load/Store	23	5.244
Soma:		113.924

	Tipos de redes	Número	Área
Redes de Entrada:	Omega 128x128	1	53.869
Redes de Saida:	Omega 64x64 Muxes 2:1	1 32	24.527
Total de interconexão:			78.396
Área Total do arranjo:			**192.320**
Adicional por estágio extra na rede de entrada:			*7.696*

Tabela 8.12. Área no arranjo 1D com rede de entrada 256x256

Unidade	Número	Área
ALU	70	117.040
Multiplicador	10	66.880
Load/Store	48	10.944
Soma:		194.864

	Tipos de redes	Número	Área
Redes de Entrada:	Omega 256x256	1	122.880
Redes de Saida:	Omega 128x128 Muxes 4:1	1 32	58.189
Total de interconexão:			181.069
Área Total do arranjo:			**375.933**
Adicional por estágio extra na rede de entrada:			*15.360*

Tabela 8.12. Área no arranjo 1D com rede de entrada 512x512

Unidade	Número	Área
ALU	156	260.832
Multiplicador	20	133.760
Load/Store	80	18.240
Soma:		412.832

	Tipos de redes	Número	Área
Redes Global:	Omega 512x512	1	276.480
Redes de Saida:	Omega 256x256 Muxes 8:1	1 32	132.960
Total de interconexão:			409.440
Área Total do arranjo:			**822.272**
Adicional por estágio extra na rede de entrada:			*30.720*