ALEX DAMIANY ASSIS

UM ALGORITMO DE POSICIONAMENTO E ROTEAMENTO POLINOMIAL PARA ARQUITETURAS RECONFIGURÁVEIS DE GRÃO GROSSO COM REDES MULTIESTÁGIO

VIÇOSA
MINAS GERAIS - BRASIL
2010

ALEX DAMIANY ASSIS

UM ALGORITMO DE POSICIONAMENTO E ROTEAMENTO POLINOMIAL PARA ARQUITETURAS RECONFIGURÁVEIS DE GRÃO GROSSO COM REDES MULTIESTÁGIO

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

Dedico essa dissertação aos meus pais
João e Benedita

A minha irmã
Aline

E a minha amada noiva
Deyse

AGRADECIMENTOS

- Agradeço a Deus pela oportunidade de realizar este trabalho e ser o porto seguro nos momentos felizes e difíceis desta caminhada;

- Agradeço aos meus pais *João Bosco* e *Benedita* pelo Amor e apoio incondicional em todos os momentos;

- Agradeço a minha irmã *Aline* pela boa convivência e pelo apoio;

- Agradeço a minha noiva *Deyse* pela paciência, compreensão e apoio neste trabalho;

- Agradeço ao professor *Ricardo* pela sua dedicação, paciência, persistência e alta disponibilidade para conclusão deste trabalho;

- Agradeço a todos os professores do DPI por compartilhar seus conhecimentos, durante esses anos de formação;

- Agradeço a todos os funcionários do DPI por sempre se mostrarem dispostos a ajudar;

- Agradeço a equipe de serviços gerais por deixarem os laboratórios e todas as dependências organizadas;

- Agradeço a Universidade Federal de Viçosa pela oportunidade de estudar;

- Agradeço ao suporte financeiro do CNPq para realização desta pesquisa.

.

BIOGRAFIA

ALEX DAMIANY ASSIS, filho de João Bosco de Assis e Benedita Fernandes de Carvalho Assis, brasileiro, nascido em 15 de maio de 1979 na cidade de Coronel Fabriciano, no estado de Minas Gerais.

No ano de 1997, concluiu o ensino médio Técnico em Informática na cidade de Coronel Fabriciano, realizou o ensino profissionalizante SENAI concluído em julho de 1997 e trabalhou na Cia. Aços Especiais Acesita S.A., hoje empresa do grupo Arcelor Mittal, no período de agosto de 1997 a outubro de 2000, Em 2002 ingressou no curso de graduação em Ciência da Computação na Universidade Federal de Viçosa, concluído no ano de 2006.

Em 2007, foi aprovado na seleção do programa de pós-graduação do Departamento de Informática – DPI, onde, em março de 2007, iniciou o mestrado em Ciência da Computação na Universidade Federal de Viçosa – UFV, defendendo sua dissertação em março de 2010.

Trabalhou como professor substituto da Universidade Federal de Ouro Preto – UFOP de abril de 2007 a julho de 2008. Atualmente, é Analista de Sistemas na Universidade Federal de Minas Gerais - UFMG, desde fevereiro de 2010.

SUMÁRIO

LISTA DE FIGURAS vii

LISTA DE TABELAS x

RESUMO xi

ABSTRACT xii

1 INTRODUÇÃO 1

1.1 Motivação 3
1.2 Objetivos 4
1.3 Contribuições 4
1.4 Estrutura da dissertação 5

2 MAPEAMENTO EM ARQUITETURAS RECONFIGURÁVEIS 6

2.1 Posicionamento e Roteamento 6
2.1.1 Algoritmo de posicionamento *Simulated Annealing* 6
2.1.2 Algoritmo de roteamento *Pathfinder* 7
2.1.3 *Versatile Place and Route* (VPR) 8
2.2 FPGA 9
2.3 Novas Arquiteturas de FPGA 11
2.4 Algoritmos de mapeamento para CGRA 15
2.5 Compilador *just-in-time* para FPGA 26
2.6 Considerações finais 26

3 ARQUITETURA HÍBRIDA 28

3.1 Decomposição do grafo antes do mapeamento 30
3.2 Multiestágio 31
3.2.1 Uso total e parcial da multiestágio 34
3.2.2 Implementação das MIN em FPGA 36

4 POSICIONAMENTO E ROTEAMENTO NAS CGRA 39

4.1 Introdução 39
4.2 Posicionamento e roteamento local 40
4.2.1 Busca em Profundidade 40
4.2.2 Busca em profundidade priorizando nodos no caminho crítico 45
4.2.3 Busca em profundidade priorizando apenas nodos no caminho crítico 48
4.2.4 Roteamento Global 51

5 RESULTADOS 56

5.1 Roteamento na rede Omega 56

5.2 Capacidade de roteamento da arquitetura e tempo de execução P&R..... 60
5.3 Caminho Crítico..... 65

6 CONCLUSÕES E TRABALHOS FUTUROS 69

REFERÊNCIAS BIBLIOGRÁFICAS 73

LISTA DE FIGURAS

Figura 1.1 Desempenho e flexibilidade das arquiteturas 2

Figura 2.1. Congestionamento de primeira ordem. (EBELING et al., 1995) 8

Figura 2.2. Estilo ilha FPGA (POON; WILTON; YAN, 2005)..... 9

Figura 2.3. Circuito somador A) porta lógica e B) LUTs. 10

Figura 2.4. A) LUTs com 4 variáveis e em B) P&R na arquitetura reconfigurável. 11

Figura 2.5. Arquitetura de FPGA baseada em Árvore (ZIED et al., 2008) 12

Figura 2.6. *Butterfly fat-tree* rearranjada e compactada. (DEHON, 2000) 13

Figura 2.7. Topologia da rede HSRA (HUANG; WAWRZYNEK; DEHON, 2003) 14

Figura 2.8. Grafo de fluxo de dados (DFG) (x+y)*z 15

Figura 2.9. Arquitetura CGRA..... 16

Figura 2.10. Posicionamento e roteamento do DFG 17

Figura 2.11. Arquitetura rDPA (LAI; LAI; YEH, 2005) 18

Figura 2.12. Mapeamento da árvore geradora mínima no rDPA 19

Figura 2.13. Estrutura composta por ALU e rede de conexão (MEHTA et al., 2009) 20

Figura 2.14. Esquema de multiplexação 5:1 (MEHTA et al., 2009) 20

Figura 2.15. Arquitetura DS-HIE (TANIGAWA et al., 2008) 21

Figura 2.16. Grade 2D de PE 22

Figura 2.17. DFG com ordem de posicionamento das arestas 22

Figura 2.18. PE com capacidade de roteamento. 23

Figura 2.19. P&R na grade 2D de PE. 24

Figura 2.20. Roteamento através dos PE, aresta n4→n9 25

Figura 2.21. Mapeamento da árvore geradora mínima na grade 2D..... 26

Figura 3.1. Vizinho local e global do PE3 28

Figura 3.2. Conversão expressão em DFG e mapeamento CGRA com MIN..... 29

Figura 3.3. Conversão expressão para DFG e mapeamento na CGRA com duas MIN 30

Figura 3.4. Decomposição entrada a) nodo com n entradas b) nodo decomposto31

Figura 3.5 Decomposição saída a) nodo com n saídas b) nodo decomposto 31

Figura 3.6. MIN genérica (DUATO; YALAMANCHILI; NI, 2003)..... 32

Figura 3.7. Padrão *Perfect Shuffle* (DUATO; YALAMANCHILI; NI, 2003).... 32

Figura 3.8. Permutação *butterfly* a) segunda ordem b) primeira ordem c)zero ordem (DUATO; YAMALANCHI; NI, 2003) 33

Figura 3.9. Permutação *cube* a) segunda ordem b) primeira ordem c) ordem zero (DUATO; YALAMANCHILI; NI, 2003)..... 33

Figura 3.10. Omega MIN com N=8 e 100% (8) dos terminais usados 35

Figura 3.11. Omega MIN com N=8 e 50% (4) dos terminais usados 36

Figura 3.12. Omega MIN com N=8 e 25% (2) dos terminais usados 36

Figura 3.13. Ocupação LUTs pela rede Omega com radix-2 de 32 bits (FERREIRA, 2010) 38

Figura 3.14. Atraso máximo na rede Omega com radix-2 de 32 bits (FERREIRA, 2010) 38

Figura 4.1. CGRA 40

Figura 4.2. Grafo de entrada DFG 40

Figura 4.3. Elementos de processamento (PE) vizinhos 41

Figura 4.4. Primeira etapa P&R busca em profundidade..... 43

Figura 4.5. Segunda etapa P&R busca em profundidade 44

Figura 4.6. Etapa final P&R busca em profundidade 45

Figura 4.7. DFG nodos do caminho crítico marcados 46

Figura 4.8. Primeira etapa P&R busca em profundidade priorizando o caminho crítico 47

Figura 4.9. Segunda etapa P&R busca em profundidade priorizando caminho crítico 47

Figura 4.10. Etapa final P&R busca em profundidade priorizando o caminho crítico 48

Figura 4.11. Primeiro passo P&R somente nodos no caminho crítico 49

Figura 4.12. Segundo passo P&R somente nodos no caminho crítico 50

Figura 4.13. Etapa final P&R busca em profundidade somente caminho crítico51

Figura 4.14. Rede Omega MIN 16x16 52

Figura 4.15. Conexão 9→12 na rede Omega 16x16 53

Figura 4.16. Conflito na MIN conexões 9→12 e 11→13 54

Figura 4.17. MIN N=16 com estágio extra, sem conflito 55

Figura 5.1. Pares de conexões aleatórios para rede com N=16. 57

Figura 5.2. Percentual de conexões realizadas com sucesso na Omega MIN, máximo de 100.000.000 de permutações considerando N entradas e N saídas .. 59

Figura 5.3. Comparação de tempo de execução do P&R entre ABP e VPR 63

Figura 5.4. DFG com atraso do caminho crítico de 4 unidades 66

Figura 5.5. DFG com conexão global a) atraso de 1 unidade b) atraso de 2 unidades 66

Figura 5.6. DFG com atraso total de 4 unidades 67

Figura 6.1. DFG "if-else" 72

LISTA DE TABELAS

Tabela 2.1. Resultado *Modulo Schedule*..... 18

Tabela 3.1. Conexões realizadas na *SW-banyan* e adição de estágio extra 35

Tabela 3.2. Custo das redes LUTs, redes com largura de 32 bits (VENDRAMINI, 2009) 37

Tabela 3.3. Área de ocupação da arquitetura (VENDRAMINI, 2009)..... 37

Tabela 5.1. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=16 e 32..... 58

Tabela 5.2. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=64 e 128..... 58

Tabela 5.3. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=256..... 58

Tabela 5.4. P&R dos algoritmos na arquitetura em grade com MIN 61

Tabela 5.5. Tempo de execução do P&R local e na MIN..... 62

Tabela 5.6. Tempo de mapeamento no CGRA ABP, ABP-P, ABP-F e ABP-F2.64

Tabela 5.6. Percentual de latência nos algoritmos de posicionamento..... 67

RESUMO

ASSIS, Alex Damiany, M.Sc., Universidade Federal de Viçosa, março de 2010. **Um algoritmo de posicionamento e roteamento polinomial para arquiteturas reconfiguráveis de grão grosso com redes multiestágio**. Orientador: Ricardo dos Santos Ferreira. Co-Orientadores: Mauro Nacif Rocha e Carlos de Castro Goulart.

Diferentes arquiteturas estão sendo utilizadas para o desenvolvimento de sistemas dedicados. Uma arquitetura reconfigurável muito difundida no mercado são os FPGAs (*Field Programmable Gate Arrays*), eles são uma estrutura flexível e eficiente, mas que exigem um grande esforço de configuração, mapeamento, pois são reconfiguráveis no nível de bits. Esta dissertação propõe utilizar uma arquitetura hibrida de grão grosso em duas dimensões sobre o FPGA de forma a reduzir a complexidade do mapeamento. A arquitetura hibrida é baseada em conexões locais e globais. As conexões locais são entre vizinhos (leste, oeste, norte e sul). As conexões globais são feitas por redes multiestágios. Além disso, são avaliados três algoritmos de posicionamento e roteamento (P&R) para as conexões locais com complexidade polinomial. Os algoritmos de posicionamento são baseados na busca em profundidade no grafo, priorizando ou não o caminho crítico.

ABSTRACT

ASSIS, Alex Damiany, M.Sc., Universidade Federal de Viçosa, March, 2010. **A placement and routing polynomial algorithm for coarse grain reconfigurable architecture with multistage networks.** Adviser: Ricardo dos Santos Ferreira. Co-Advisers: Mauro Nacif Rocha and Carlos de Castro Goulart.

Differents architectures have been proposed for embedded systems. The FPGAs (Field Programmable Gate Array) has an efficient and flexible structure. However it requires a great effort of configuration and mapping at bit level. This work proposes an hybrid 2-dimension architecture coarse grain, as a logic layer upon the FPGAs. This approach reduces the mapping complexity. The hybrid architecture are based on local and global connections. The local connections consist of neighbour links (east, west, north and south). The global connections are done by multistage interconnection network (MIN). Furthermore, three placement and routing polynomial algorithms are evaluated to perform the local connections. These algorithms are depth first search based and prioritize the critical path.

.

1 INTRODUÇÃO

A busca pelo aumento do poder de processamento e a exploração contínua dos recursos do sistema tem criado muitos desafios, motivando o desenvolvimento de técnicas e métodos para criação de compiladores e arquiteturas. As principais restrições para sistemas embarcados são: desempenho do sistema, a quantidade de memória, o tamanho do código, o consumo de energia, o peso e tamanho do dispositivo, o comportamento em tempo-real, a segurança, o *time-to-market* e o custo. Diferentes arquiteturas têm sido utilizadas, incluindo processadores de propósito geral (GPPs – *General Purpose Processors*), plataformas de computação reconfigurável e circuitos integrados para aplicações específicas (ASIC - *Application Specific Integrated Circuits*) (MOREANO, 2005).

Com o conjunto de instruções definidas no processador GPP, é possível executar diversas aplicações, sendo necessário apenas mudar o *software*, ou seja, o GPP é uma arquitetura de alta flexibilidade. O ASIC é uma arquitetura especializada, na qual toda a aplicação é implementada no *hardware*, desta forma conseguindo ter um desempenho melhor para uma determinada aplicação. As operações são implementadas de forma eficiente. Entretanto os ASICs não permitem alterações do circuito e caso alguma alteração seja necessária, deve-se reprojetar o circuito e refabricá-lo. Essa solução é interessante para aplicações bem definidas e utilizadas em larga escala para compensar os custos de projeto e fabricação (MOREANO, 2005).

A arquitetura reconfigurável surgiu como uma solução que faz o balanceamento entre a flexibilidade e o desempenho. Na Figura 1.1, a arquitetura reconfigurável apresenta maior desempenho em relação ao GPP e maior flexibilidade que o ASIC.

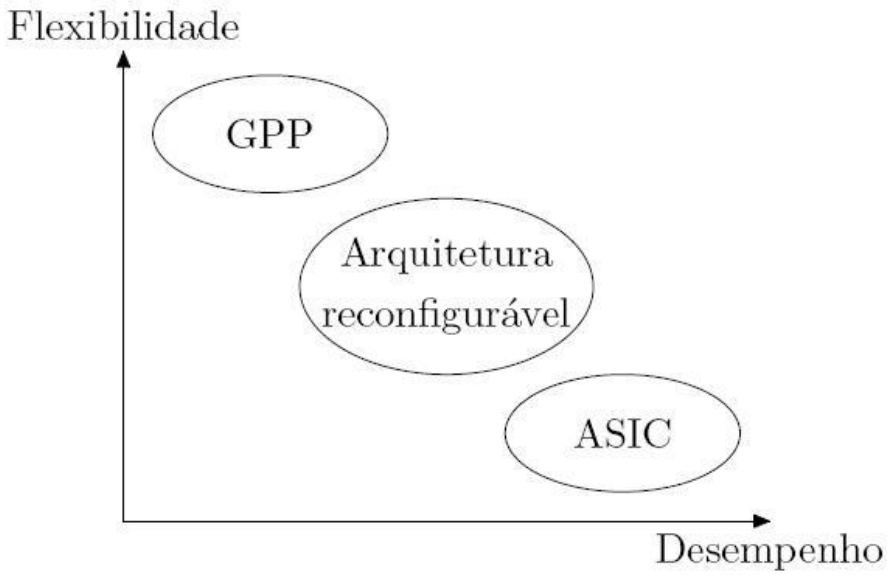


Figura 1.1 Desempenho e flexibilidade das arquiteturas

A arquitetura reconfigurável (AR) permite que o hardware seja modificado para atender às necessidades da aplicação e explorar o paralelismo do código. A configuração da arquitetura para uma determinada aplicação pode executar o algoritmo de forma mais eficiente que os GPPs, evitando a ativação de circuitos mais genéricos para a resolução do problema. Mostra-se também mais flexível em relação aos ASICs por permitir atualizações e correções (OTERO, 2005).

O *hardware* reconfigurável pode ser programado para implementar um algoritmo de criptografia e mais tarde ser usado para implementar um algoritmo de compactação de dados, esta flexibilidade permite implementar os algoritmos diretamente no hardware. Neste contexto, é importante fornecer métodos e ferramentas para modelagem rápida e avaliação de diferentes arquiteturas de *hardware* para implementar um dado algoritmo (HARTENSTEIN, 2003).

Em uma AR, os blocos lógicos e suas conexões são estruturas reconfiguráveis. O tamanho e a complexidade de cada bloco lógico definem a granularidade do sistema (MOREANO, 2005). Os componentes reconfiguráveis de granularidade fina possuem blocos lógicos capazes de manipular dados no nível de bit e possuem maior flexibilidade. Porém têm um elevado custo de reconfiguração. O FPGA (*Field Programmable Gate Array*) é um dispositivo de granularidade muito fina, composto por uma matriz de blocos lógicos muito simples e regulares.

As ARs com componentes de grão grosso possuem blocos lógicos que formam unidades funcionais (por exemplo, somador de 16 bits) e implementam *datapaths* de forma mais simples e eficiente que as arquiteturas de granularidade fina.

1.1 Motivação

As arquiteturas reconfiguráveis têm ganhado importância na construção de sistemas embarcados e na computação de alto desempenho (HARTENSTEIN, 2003). O mercado das arquiteturas reconfiguráveis é dominado pelos FPGAs muito utilizados em prototipação de circuitos.

A alta flexibilidade penaliza os FPGAs aumentando a complexidade de síntese e mapeamento das aplicações. Além disso, os FPGAs dedicam uma grande área ao roteamento (HARTENSTEIN, 2001b). Uma arquitetura de grão grosso pode ser uma alternativa viável, pois aumentando a granularidade pode-se simplificar o mapeamento das aplicações. Porém existem poucas arquiteturas de grão grosso comerciais.

Ao invés de fabricar uma nova arquitetura de grão grosso, pode-se usar os *soft-cores*, onde elas são implementadas em uma camada lógica acima da arquitetura física dos FPGAs. Esta é uma opção interessante que aproveita o FPGA existente e simplifica a complexidade do mapeamento. Apesar da estrutura de grão grosso perder um pouco em flexibilidade, esta solução consegue se adaptar as aplicações e demandas dos sistemas embarcados.

Mais especificamente, com relação à complexidade e capacidade de roteamento, uma estrutura em malha é interessante, pois pode aproveitar a própria estrutura regular do posicionamento e roteamento existente no FPGA.

A possibilidade de poder fazer isso de maneira eficiente (baixo tempo de execução durante a síntese) sobre os FPGAs será abordada nesta dissertação com o foco na simplificação do mapeamento para uma arquitetura de grão grosso. Um mapeamento eficiente em termos de tempo de execução pode ser incorporado em compiladores JIT (*Just-in-time*). Esta alternativa é interessante para garantir portabilidade em sistemas embarcados.

Existe um compromisso entre o tempo de execução do mapeamento e o número de recursos de interconexões. O mapeamento envolve dois passos: posicionamento e roteamento. Em uma arquitetura onde todos os elementos possuem uma ligação com todos os outros, ou seja, um grafo completo, o mapeamento é

trivial. Podemos posicionar em qualquer elemento que sempre será possível conectar diretamente os elementos. Entretanto, o custo em *hardware* de uma rede de interconexões completa é inviável, mesmo para arquitetura com poucos elementos de processamento.

Analisando o algoritmo de posicionamento. Se o posicionamento é ruim, serão necessários muitos recursos para efetuar o roteamento e um longo tempo de execução. Algumas vezes, um posicionamento ruim não possibilita que o roteamento de todas as conexões seja realizado. É importante buscar um custo-benefício na complexidade das redes de interconexão e dos algoritmos de posicionamento e roteamento.

Uma alternativa pouco explorada em ambientes reconfiguráveis são as redes de interconexão multiestágios. Estas redes apresentam um custo polinomial $O(n\log_2 n)$, boa conectividade e algoritmos de roteamento polinomiais.

1.2 Objetivos

O objetivo deste trabalho é avaliar algoritmos de mapeamento em tempo polinomial para arquiteturas reconfiguráveis de grão grosso em malha organizada em duas dimensões (2D) para serem usados em compiladores JIT. Para alcançar o objetivo geral, é necessário cumprir os seguintes objetivos específicos:

- simplificar as arquiteturas reconfiguráveis de grão grosso com ligações locais;
- avaliar uma rede global multiestágio como alternativa para conexão dos elementos de processamento não vizinhos;
- avaliar algoritmos de posicionamento e roteamento polinomiais para arquitetura de grão grosso.

1.3 Contribuições

Conforme mencionado na Seção 1.1, a alta complexidade de mapeamento nos FPGAs é um fator motivador para busca de alternativas. Este trabalho apresenta uma solução baseada em CGRA (*Coarse Grain Reconfigurable Architecture*) que utiliza uma MIN (*Multistage Interconnection Network*), para reduzir a complexidade do mapeamento. As principais contribuições foram:

- O custo da arquitetura foi reduzido a partir da inclusão de uma arquitetura híbrida com conexões locais em malha e redes multiestágio;

- simplificação do algoritmo de posicionamento e roteamento na nova arquitetura, com custo polinomial;
- derivações do algoritmo de posicionamento para priorizar o caminho crítico;
- baixo tempo de execução, 14x mais rápido que o VPR (*Versatile Place and Route*).

1.4 Estrutura da dissertação

O restante desta dissertação está estruturado da seguinte forma: o Capítulo 2 apresenta o estado da arte do mapeamento em arquiteturas reconfiguráveis. A arquitetura híbrida proposta neste trabalho é apresentada no Capítulo 3. No Capítulo 4 são descritos os algoritmos de posicionamento e roteamento. Os resultados serão apresentados no Capítulo 5. O Capítulo 6 discute as principais conclusões deste trabalho.

2 MAPEAMENTO EM ARQUITETURAS RECONFIGURÁVEIS

Neste capítulo, serão apresentadas algumas arquiteturas reconfiguráveis de grão grosso (CGRA) e de FPGA, envolvendo o problema de posicionamento e roteamento. Analisaremos os seguintes aspectos das arquiteturas: ganho de desempenho, área ocupada, escalabilidade, regularidade, etc. A maior parte das abordagens desacoplam o posicionamento do roteamento. Primeiro, os elementos são posicionados na arquitetura baseados em estimativas de custo. Depois, em um segundo passo, o roteamento é realizado, sem mudar os elementos de posição.

Nas próximas seções serão descritos os algoritmos de posicionamento e roteamento e a ferramenta VPR. Novas arquiteturas em FPGA propostas pela literatura e também algoritmos utilizados no mapeamento para CGRA.

2.1 Posicionamento e Roteamento

Nesta seção serão apresentados os algoritmos de *Simulated Annealing* e *Pathfinder* que são os mais utilizados para posicionamento e roteamento, respectivamente (HARTENSTEIN, 2001b). Posteriormente, a ferramenta VPR (BETZ; ROSE, 1997), utilizada como referência em vários trabalhos, será descrita.

2.1.1 Algoritmo de posicionamento *Simulated Annealing*

O *Simulated Annealing* é uma heurística muito utilizada para o posicionamento em FPGA devido a sua capacidade de produzir resultados satisfatórios ao ajustar funções objetivo complexas (VORWERK et al., 2007). No posicionamento busca-se, em geral, minimizar o comprimento dos fios e o atraso no caminho crítico. As primeiras abordagens de posicionamento usando *Simulated Annealing* surgiram no início da década de 80 em (KIRKPATRICK; GELATT; VECCHI, 1983) e continuam a ser usadas em trabalhos recentes (VORWERK et al., 2007).

Vários estudos têm sido feitos para melhorar a qualidade do posicionamento e o tempo de execução (VORWERK et al., 2007). Para minimizar o tempo de

execução, que podem ser muito longos, as principais técnicas são: melhorar posicionamento inicial, melhorar o empacotamento com posições mais próximas, realizar a duplicação de lógica, usar funções objetivos mais precisas, incorporar estimativas de peso e congestionamento e/ou alterar os critérios de aceitação.

2.1.2 Algoritmo de roteamento *Pathfinder*

O *Pathfinder* é um algoritmo iterativo baseado na proposta de (NAIR, 1987), mas difere na definição da função de custo e no tratamento do atraso. A cada iteração foi adicionada a capacidade de retirar os sinais já roteados e roteá-los novamente (*rip-up and retry*) (EBELING et al., 1995).

O roteamento é realizado através da negociação dos recursos entre os sinais, a fim de verificar qual deles mais necessita do recurso. Para diminuição do atraso, os sinais mais críticos têm prioridade na negociação.

O algoritmo pode ser divido em duas partes (MCMURCHIE; EBELING, 1995): um roteador de sinal (RS) que roteia um sinal por vez usando o algoritmo de menor caminho e um roteador global (RG) que chama o RS para rotear todos os sinais, ajustando os custos do recurso (CR) a fim de atingir o roteamento completo. O RS usa a busca em largura para encontrar o menor caminho, atribuindo um custo de congestionamento e atraso para cada recurso. O RG ajusta dinamicamente o custo de cada recurso baseado nas demandas dos sinais.

Durante a primeira iteração do RG não existe custo para o compartilhamento do recurso. Nas próximas interações o custo é gradualmente incrementado em função do número de sinais que compartilham o recurso. Assim os sinais entram em negociação pelos recursos. Quando os sinais têm grande demanda por um determinado recurso, aumenta-se o custo do recurso. Então alguns sinais buscarão outros recursos com menor demanda. O RG faz novamente o roteamento dos sinais usando o RS até que não existam mais recursos sendo compartilhados.

A função que define o custo do uso de um dado nodo n na arquitetura durante o roteamento é dada por:

$$c_n = (b_n + h_n) * p_n \qquad (1)$$

onde b_n é o custo base para o uso do nodo n, h_n histórico de congestionamento desse nodo acumulado em iterações anteriores do roteador global e p_n número de sinais usando o nodo. Uma escolha razoável para b_n é o próprio atraso do nodo n, no

intuito de minimizar o atraso do sinal a ser roteado em sua totalidade. Os termos p_n e h_n serão explicados a seguir.

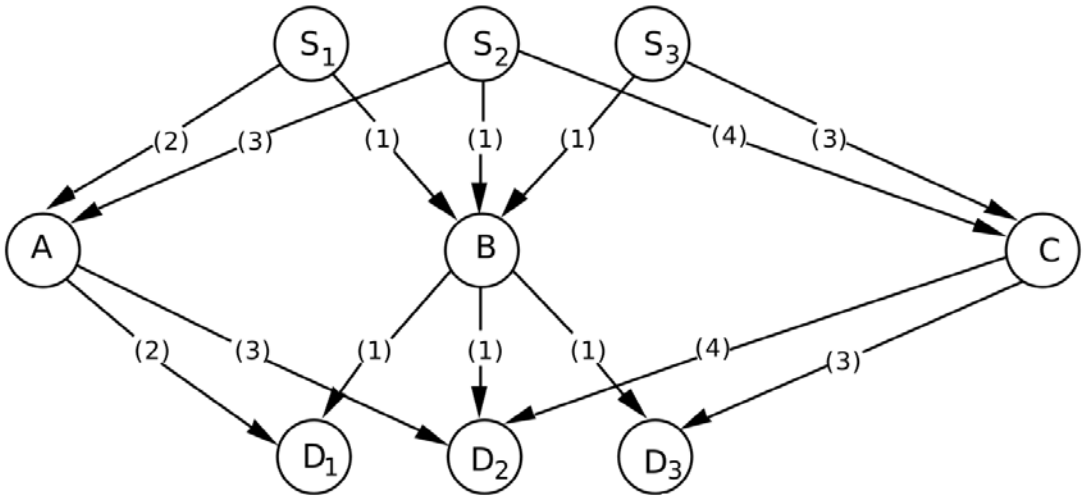


Figura 2.1. Congestionamento de primeira ordem. (EBELING et al., 1995)

Ao considerar o exemplo da Figura 2.1 para rotear os sinais 1, 2 e 3, parte-se das suas respectivas fontes S1, S2 e S3 e incide-se sobre seus receptores D1, D2 e D3. Os arcos no grafo representam os caminhos parciais fornecidos pela arquitetura com seus respectivos custos entre parênteses. Sem considerar o congestionamento, o caminho de menor custo para cada uma das fontes, passa pelo nodo B. O termo p_n é iniciado com 1 e h_n com 0. Nenhuma penalidade recai sobre o nodo na primeira iteração. À medida que o nodo n se tornar mais congestionado, p_n será gradualmente aumentado de acordo com o número de sinais que compartilham o nodo. Desta forma, os caminhos que não utilizem n tornam-se mais vantajosos. Na primeira iteração, os sinais congestionam o recurso B. Nas iterações subsequentes, o algoritmo decidirá que o menor custo para o sinal de S1 é através do recurso A, diminuindo o congestionamento em B. Em outra iteração, o sinal de S3 terá menor custo através do recurso C. Para finalizar, o sinal de S2 passa por B.

2.1.3 *Versatile Place and Route* (VPR)

O VPR (BETZ; ROSE, 1997) é uma ferramenta CAD (*Computer-Aided-Design*) muito utilizada no mapeamento em arquiteturas FPGA. O algoritmo de posicionamento é baseado no *Simulated Annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) e o roteamento é baseado no *Pathfinder* (MCMURCHIE; EBELING, 1995). O VPR é uma referência na área devido aos seus recursos e eficiência. O usuário tem várias opções de critérios de custo como tempo de

execução, menor caminho crítico, redução do número de barramentos, limitação do número máximo de iterações do *Simulated Annealing* ou *PathFinder*. Atualmente o VPR está na versão 5.0, lançada recente em 2008.

A arquitetura do FPGA no VPR é baseada no estilo ilha, composta por blocos lógicos, blocos de chaves, blocos de conexões e roteamento, como mostra a Figura 2.2. Os blocos lógicos são os quadrados cinza e em geral tem quatro ou cinco terminais. Estes terminais são ligados aos blocos de conexões, que são segmentos de barramentos, que por sua vez ligam-se aos elementos de chaveamento que irão interconectar os barramentos.

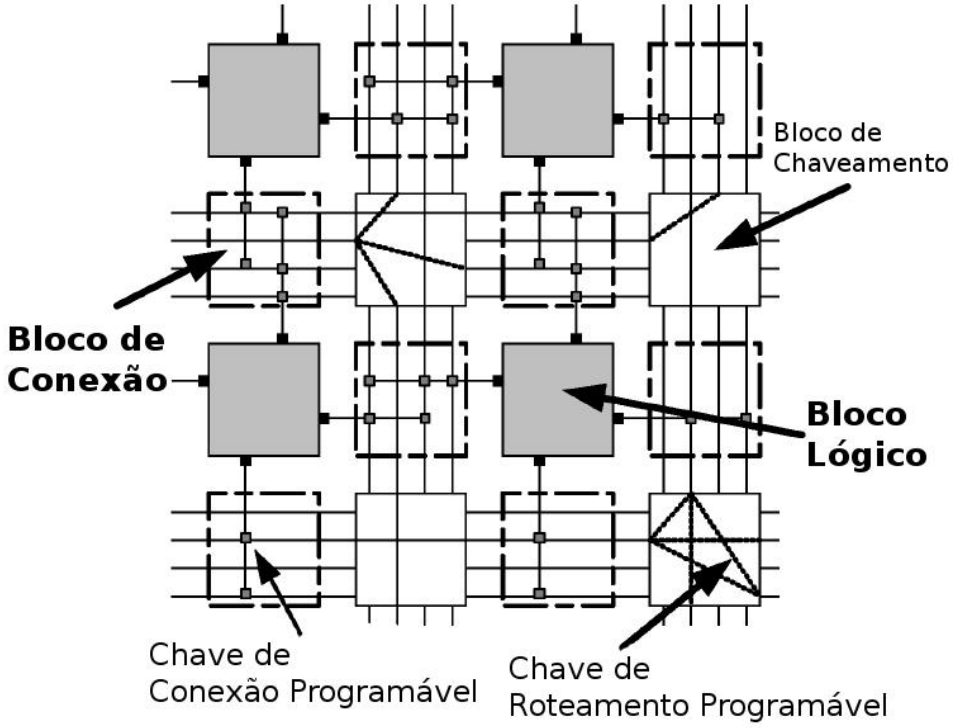


Figura 2.2. Estilo ilha FPGA (POON; WILTON; YAN, 2005)

2.2 FPGA

Os *Field Programmable Gate Arrays* (FPGAs) são circuitos reconfiguráveis, podem ser modificados internamente no nível de *hardware* após sua fabricação. Em contraste com os circuitos integrados VLSI (*Very Large Scale Integration*) que são circuitos projetados e uma vez fabricados não podem mais ser alterados a nível de *hardware*.

O FPGA é composto por um conjunto de blocos lógicos e um conjunto de interconexões. Os blocos lógicos são reconfiguráveis, podem implementar funções simples de alguns bits. Em geral, os blocos lógicos são LUTs (Look Up Tables) que

executam uma função binária de n variáveis, na Figura 2.3 mostra um somador com portas lógicas em A) e a mesma representação com 2 blocos LUTs em B) para implementar a lógica de “Soma” e “Vai-um”. O número de variáveis booleanas por LUT varia em torno de 4 a 5.

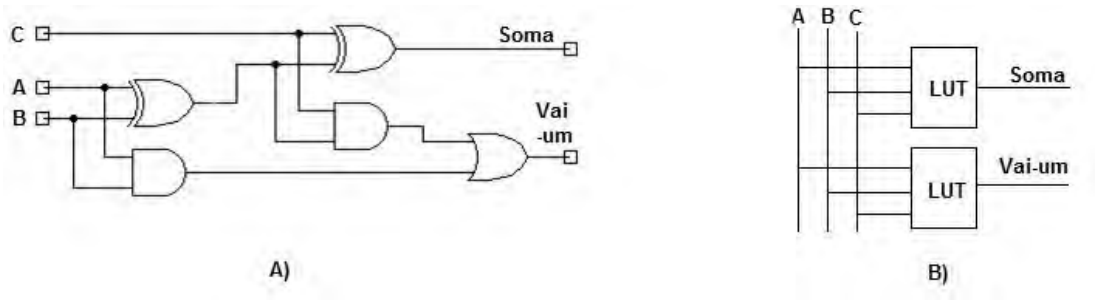


Figura 2.3. Circuito somador A) porta lógica e B) LUTs.

Os blocos lógicos se comunicam por meio de uma rede de interconexão reconfigurável. A maioria dos FPGAs comerciais usam uma arquitetura de interconexão estilo-ilha, como ilustrado na Figura 2.2. Esta arquitetura é composta por um conjunto de barramentos segmentados interligados por blocos de roteamento (SW). Os blocos lógicos e as entradas/saídas (I/O) são conectadas aos barramentos e através dos blocos de roteamento se interligam. A operação de A(a1a0)+B(b1b0) é executada no nível de bits, pelos LUTs com a função de “Soma” e “Vai-um”, representado na Figura 2.4A, já na Figura 2.4B temos o P&R desta soma na arquitetura reconfigurável. A granularidade é definida pelo número de variáveis das LUTs. Por exemplo, as arquiteturas de grão fino são compostas por n-LUT, onde n < 8. Já as arquiteturas de grão grosso são formadas por conjunto de elementos de processamento como ALUs (*Arithmetic Logic Unit*) de 8, 16 ou 32 bits. De acordo com a descrição e do tipo de circuito o posicionamento e roteamento no FPGA pode ser complexo devido a alta flexibilidade e o grande espaço de solução.

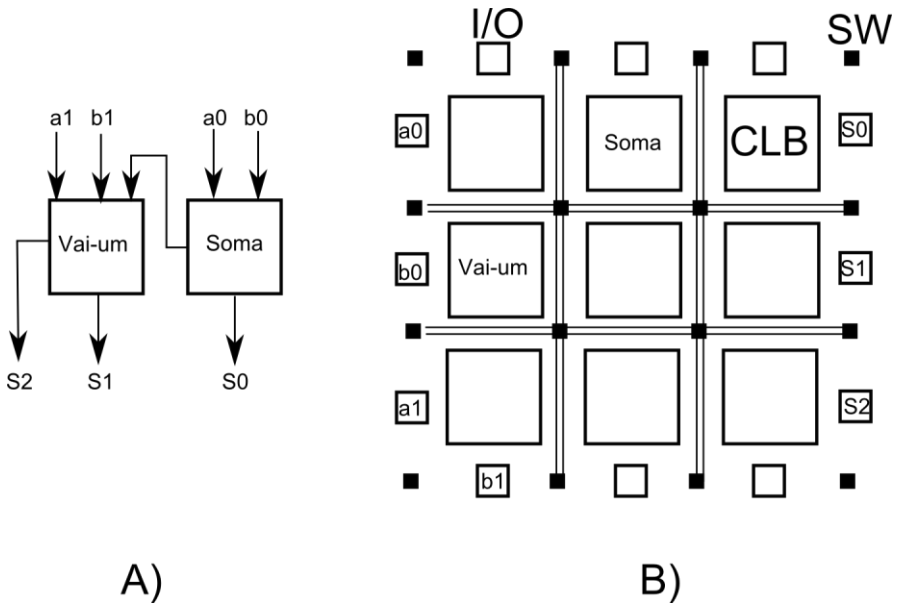


Figura 2.4. A) LUTs com 4 variáveis e em B) P&R na arquitetura reconfigurável.

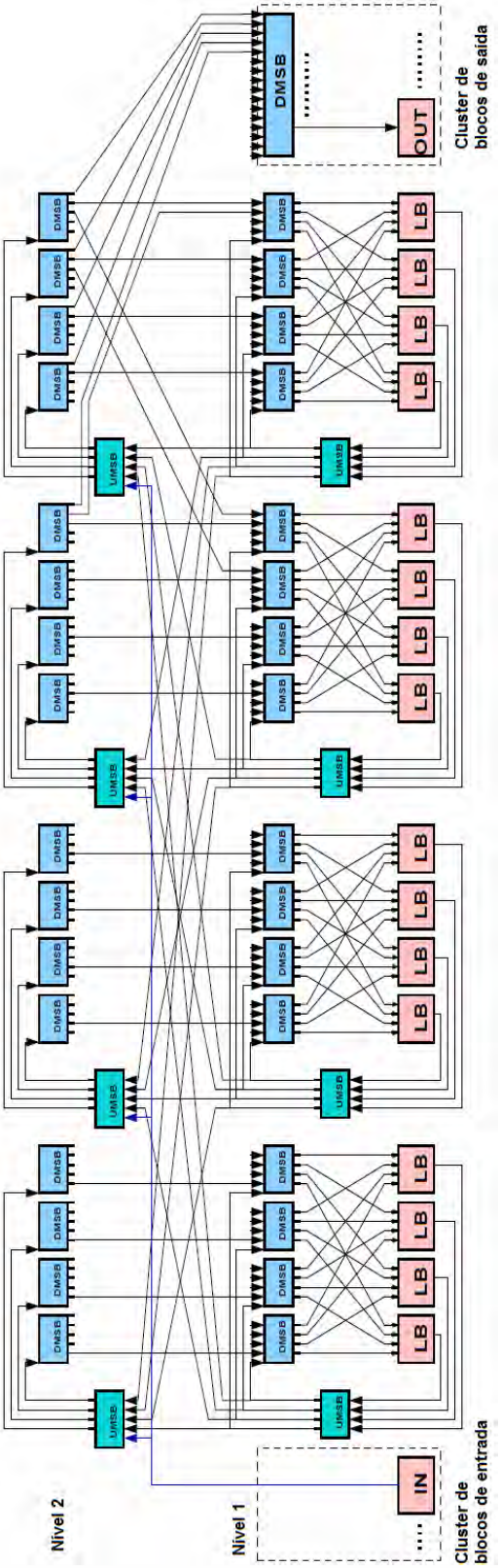
Coarse Grain Reconfigurable Architecture (CGRA) são unidades funcionais complexas como Unidades Lógica Aritméticas (ULA) e multiplicadores que processa dados a nível de palavras (8, 16 ou 32 bits) (CHANG,2008). São compostos por barramentos de interconexão e elementos de processamento para realizar operações a nível de palavra como um *datapath*. A configuração do FPGA para a realização de operações regulares no nível de palavra pode ser complexa em relação ao CGRA, que a priori é definido para trabalhar neste nível, pois os FPGAs atuam em nível de bits. Portanto com o CGRA ocorre a simplificação de configuração, poucos bits são necessários, para realizar a mesma configuração que uma arquitetura de grão fino baseada em LUTs.

2.3 Novas Arquiteturas de FPGA

Nesta seção, são apresentados exemplos de arquitetura FPGA com estrutura física baseada em redes multiestágio. Estes exemplos mostram as vantagens e viabilidade do uso das redes na substituição no nível físico do estilo ilha (Figura 2.2) pela topologia multiestágio. Neste trabalho usaremos redes multiestágios em nível lógico sobre o FPGA com estrutura física estilo ilha.

Em (ZIED et al., 2008) foi proposta uma nova arquitetura de FPGA com a utilização de redes multiestágio. A arquitetura é baseada em Árvore que unifica duas redes reconfiguráveis unidirecionais. É uma hierarquia em multi-nível onde a rede inferior usa a topologia *Butterfly Fat Tree* para conectar os micro chaveadores

(DMSB) as entradas dos blocos lógicos (LB) e a rede superior conecta a saída dos LB aos comutadores de cada estágio. A Figura 2.5 ilustra a arquitetura proposta.



Arquitetura de FPGA baseada em Árvore (ZIED et al., 2008)

Figura 2.5. Arquitetura de FPGA baseada em Árvore (ZIED et al., 2008)

Comparando com arquiteturas tradicionais de FPGA baseadas no estilo ilha com unidades lógicas em malhas 2D, mapeadas usando o VPR (BETZ; ROSE, 1997), obteve-se uma redução de mais de 50% em área de ocupação. Apesar de a multiestágio ganhar em roteamento, a malha 2D é mais escalável devido a sua regularidade. Foi sugerido por (ZIED, 2008) como trabalho futuro para melhorar a escalabilidade da multiestágio mesclar as topologias de malha e árvore. Entretanto, um novo FPGA deve ser construído para implementar a nova arquitetura.

Uma outra proposta de implementação de uma nova arquitetura FPGA usando multiestágio, *fat-tree* (LEISERSON, 1985), com topologia *butterfly*, foi apresentada por (DEHON, 2000). Processos modernos de construção de circuitos integrados podem explorar o modelo de ligações em multi-camadas, para implementar de forma eficiente as ligações multiestágios.

A Figura 2.6 mostra a *butterfly fat-tree* em um *layout* compactado.

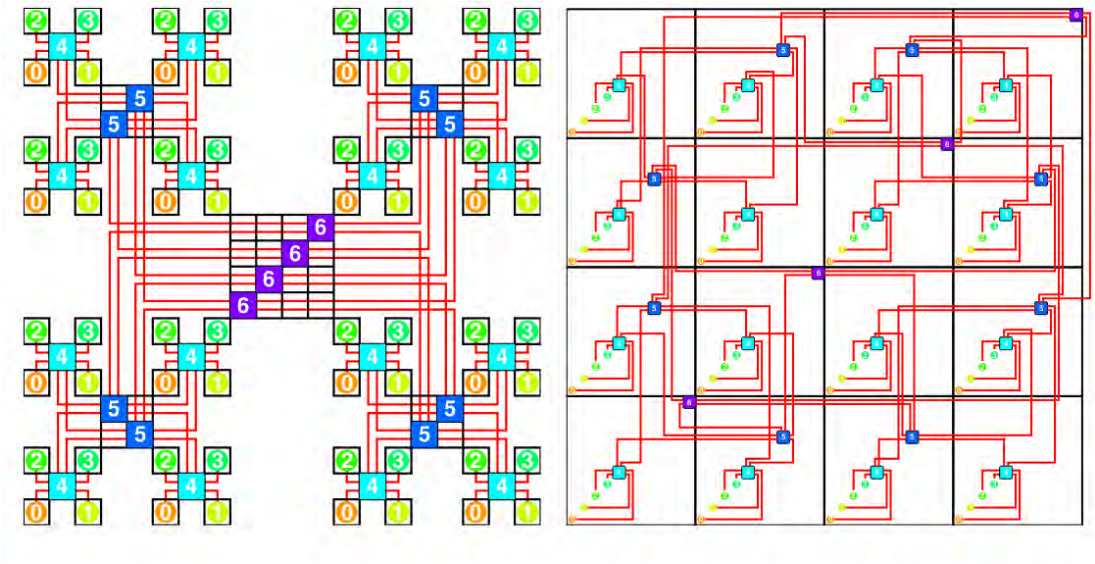


Figura 2.6. *Butterfly fat-tree* rearranjada e compactada. (DEHON, 2000)

O layout compacto da *fat-tree* foi possível graças à capacidade dos chaveadores se arranjarem para cada nodo de processamento e as ligações não saturarem as vias de conexões das camadas.

Posteriormente, uma nova abordagem foi apresentada em (HUANG; WAWRZYNEK; DEHON, 2003) para acelerar o processo de roteamento no FPGA. A proposta é baseada no roteamento auxiliado por *hardware* implementado em uma rede *fat-tree*.

Os elementos de chaveamento da rede possuem uma unidade de controle que é configurada durante o roteamento. A Figura 2.7 ilustra a rede, onde P são extremos

da rede (ex. *LookUp-Table* - LUT), os círculos e ovais são pontos de chaveamento e os X, são as caixas de conexões das chaves.

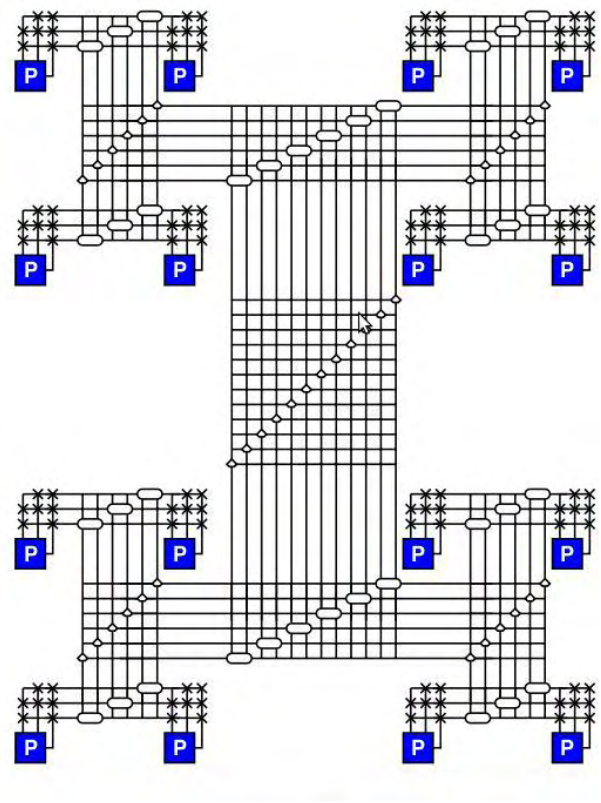


Figura 2.7. Topologia da rede HSRA (HUANG; WAWRZYNEK; DEHON, 2003)

A busca por uma rota viável na rede é feita inserindo sinais na origem e no destino. Os sinais percorrem as chaves. Se os sinais se encontrarem em uma caixa de chaveamento, o caminho será marcado entre origem-destino.

A implementação no FPGA da lógica de roteamento foi avaliada em relação ao VPR (*Pathfinder*), resultando em um ganho de duas ordens de grandeza na execução. Entretanto, a lógica de roteamento proposta em (HUANG; WAWRZYNEK; DEHON, 2003) demanda a construção de um novo FPGA, que inclui a capacidade de ter o roteamento em *hardware* implementados nos elementos de chaveamento.

Foram realizados vários estudos que tratam da viabilidade de implementação das redes multiestágio diretamente em VLSI (*Very Large Scale Integration*). O trabalho apresentado por (YEH; VARVARIGOS; PARHAMI, 1999) mostrou que as redes multiestágio podem ser sintetizadas em VLSI com estrutura proporcional ao quadrado, podendo ser usadas em duas dimensões desde que tenha recursos de multi-camada. Um outro trabalho (MUTHUKRISHNAN; et al, 1999) também mostra a

viabilidade do *layout* das redes multiestágio em duas dimensões com $O(N^2)$, sendo N o número de nodos. Esta proposta foi aplicada em comutadores de redes ATM (*Asynchronous Transfer Mode*).

Nesta seção, foram apresentadas arquiteturas de FPGA com redes multiestágio no nível físico. Entretanto novos circuitos têm que ser fabricados. Na próxima seção, iremos apresentar arquiteturas de grão grosso que também demandam a construção de novos circuitos.

2.4 Algoritmos de mapeamento para CGRA

Nesta seção iremos apresentar arquiteturas de grão grosso e suas abordagens para interconexão, posicionamento e roteamento.

Primeiro iremos adotar uma notação padrão para descrever os algoritmos. O mapeamento envolve um grafo com a descrição da computação. Os vértices representam as operações e as arestas as dependências entre elas. Um exemplo de grafo é ilustrado na Figura 2.8 para computação de (x+y)*z. Para simplificar, o grafo será denominado DFG (*Data Flow Graph*).

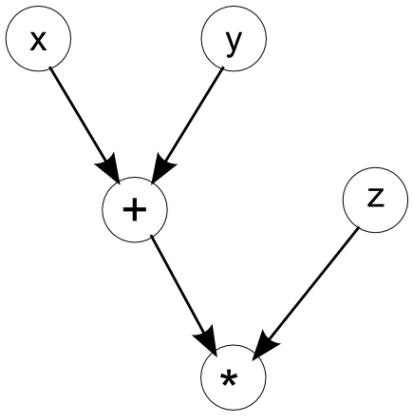


Figura 2.8. Grafo de fluxo de dados (DFG) (x+y)*z

A arquitetura onde o mapeamento será executado é formada por um conjunto de PE (*Processor Element*) dispostos em grade de duas dimensões (2D). Na grade os PEs possuem conexões com seus vizinhos. Entenda como vizinho de um PE os elementos localizados ao norte, sul, leste e oeste. Os PEs podem executar uma operação (adição, multiplicação, divisão, subtração, etc) ou armazenar um valor. A Figura 2.9 ilustra a arquitetura, e esta será denominada arquitetura reconfigurável de grão grosso (CGRA).

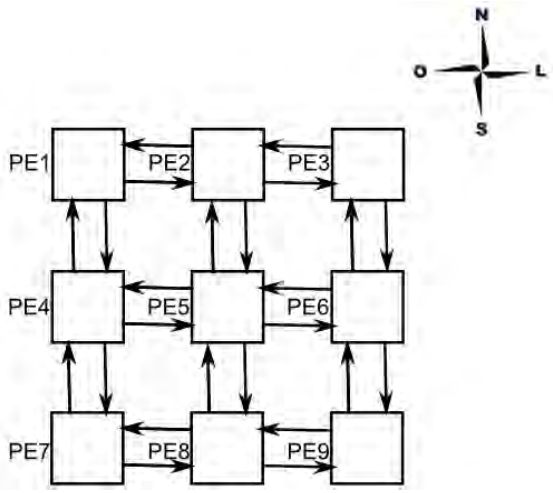


Figura 2.9. Arquitetura CGRA

Um algoritmo de *Modulo Scheduling* para o mapeamento de um DFG em um CGRA foi proposto por (MEI et al., 2003). O algoritmo trabalha com três dimensões, na qual a arquitetura é formada por uma malha de PEs em duas dimensões. O eixo do tempo é a terceira dimensão. O problema de mapeamento foi dividido em três sub-problemas: posicionamento, roteamento e escalonamento.

O posicionamento determina em qual PE da malha a operação será alocada. O escalonamento determina o ciclo em que a operação é executada. E o roteamento conecta os PE alocados de acordo com o escalonamento das operações respeitando as dependências dos dados.

A arquitetura usada nos testes deste algoritmo assemelharam-se ao Morphosys (SINGH et al., 2000) e possui 64 FU (*Functional Unit*), uma em cada PE. Os PEs foram agrupados em quatro quadrantes, e cada quadrante é um grupo de 4x4. Cada PE possui um banco de registradores dedicados que armazena valores intermediários. Além das conexões com os vizinhos, ligações locais, cada PE pode conectar diretamente a outro PE dentro do mesmo quadrante deste que estejam na mesma linha ou coluna. A capacidade de conexão dentro do quadrante aumenta o custo da arquitetura e a flexibilidade, mesmo assim o roteamento é lento, da ordem de segundos para grafo com dezenas de vértices.

O mapeamento de um DFG é apresentado na Figura 2.10. O processo começa com o posicionamento do grafo (Figura 2.10a) na arquitetura (Figura 2.10b) onde os nodos **a**, **b**, **c**, **d**, e **e** são alocados em PE1, PE2, PE4, PE5 e PE6 respectivamente. O próximo passo é o roteamento em três intervalos de tempo. Durante o intervalo t=1 (Figura 2.10c), o algoritmo de roteamento conectará os PEs vizinhos PE1→PE2 e

PE1→PE4. As conexões realizadas em t=1 são liberadas. No intervalo t=2, será feita a conexão do PE4→PE5 que corresponde a aresta do grafo c→d e a conexão de PE2→PE3, que propaga b de PE2 para PE3, temporariamente. O PE3 recebe e armazena o valor da computação do nodo **b**, ou seja, realiza um *store-and-forward*. Finalmente no t=3, a computação do PE3, elemento intermediário, é transmitida para o PE6 que tem o nodo e alocado, PE3→PE6, e a computação do PE5 é transmitida para PE6, representa d→e.

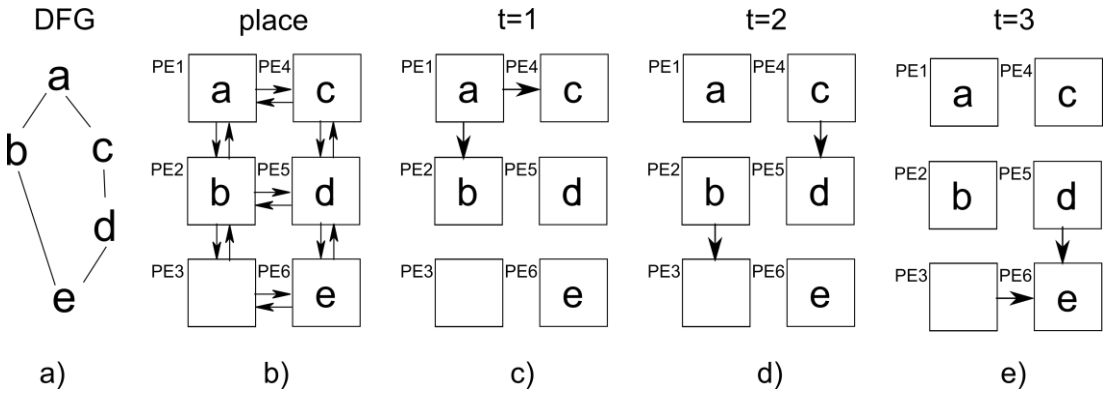


Figura 2.10. Posicionamento e roteamento do DFG

O algoritmo de posicionamento é baseado no *Simulated Annealing*, cuja função objetivo é minimizar a função de custo dos recursos de posicionamento e o roteamento semelhante ao *Pathfinder*. Nesta arquitetura pode-se utilizar a técnica de *pipeline* para grafos do tipo *dataflow*. Entretanto, o algoritmo de *Modulo Scheduling* tem custo elevado de implementação, onde o tempo de execução é da ordem de 100 a 1000 segundos para grafos com 100 vértices conforme os algoritmos usados como *benchmark* são aplicações de processamento de sinais digitais e multimídia com alto potencial de paralelismo: *idct* processa o inverso da transformação discreta do cosseno 8x8 (compressão de áudio e vídeo), *fft* refere-se a transformada rápida de Fourier, *corr* processa correlação 3x3 e *latanal* é uma função de análise de rede apresentados na Tabela 2.1 (MEI et al., 2003), que inviabiliza sua utilização em ambientes de compilação JIT que necessitam de tempos da ordem de milissegundos.

Tabela 2.1. Resultado *Modulo Schedule*

Benchmark	Número de operações	Tempo (segundos)
idct	86	162
Fft	70	891
Corr	56	100
latanal	12	5,2

Um algoritmo de posicionamento polinomial foi proposto em (LAI; LAI; YEH, 2005) para a arquitetura reconfigurável rDPA (*Reconfigurable Datapath Architecture*) (HARTENSTEIN; KRESS, 1995). Os recursos de roteamento do rDPA são simples, permitindo ligações locais apenas com os vizinhos a frente ou abaixo do elemento de processamento (DPU). Para as ligações globais utilizou-se um barramento global, como ilustrado na Figura 2.11. Na etapa de decomposição do DFG, o número de vértices (V) inseridos no grafo é menor que o número de arestas (E). O número de arestas deve ser menor que o quadrado do número de vértices do grafo, $E < V^2$. A seleção das arestas para encontrar a maior árvore geradora utiliza o algoritmo de *Kruskal*, que possui complexidade proporcional ao número de arestas (E) (O(E.logE)). Assim a complexidade do tempo de decomposição é limitada por $O(|V|^2 LogV)$, onde V é o número de vértices do grafo.

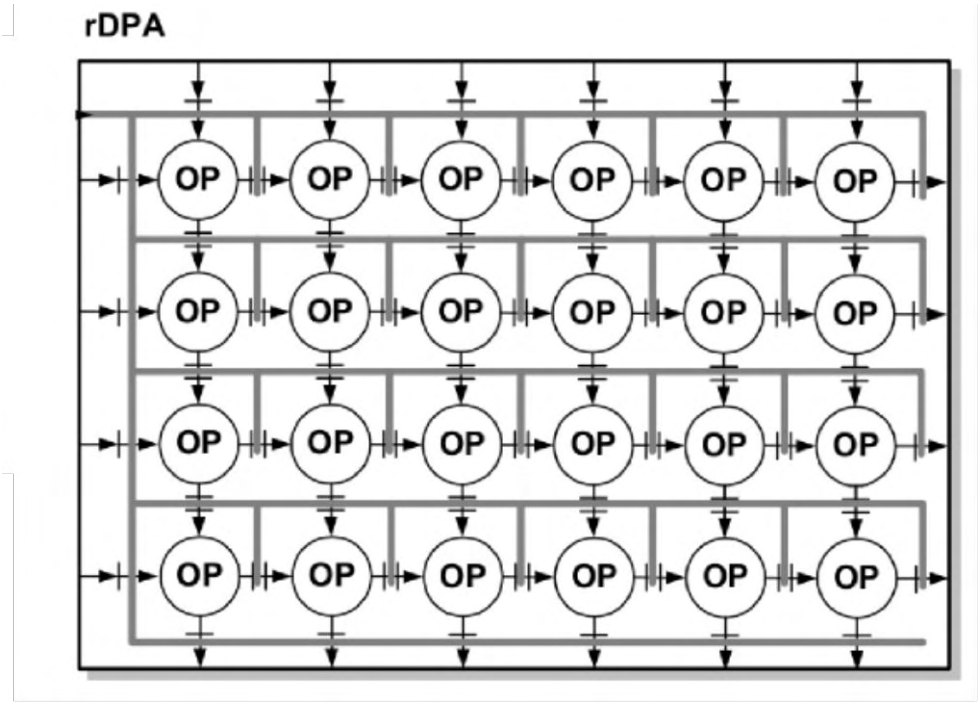


Figura 2.11. Arquitetura rDPA (LAI; LAI; YEH, 2005)

O primeiro passo do mapeamento do DFG na arquitetura é a decomposição do grafo, onde os vértices não podem ter mais que duas arestas de entrada e duas de saída, atendendo a restrição da arquitetura. O algoritmo de posicionamento é baseado na árvore geradora mínima (SEDGEWICK, 1998) onde todas as arestas têm pesos iguais. Um exemplo de mapeamento está ilustrado na Figura 2.12. O barramento global foi utilizado para ligar os vértices que não foram conectados localmente.

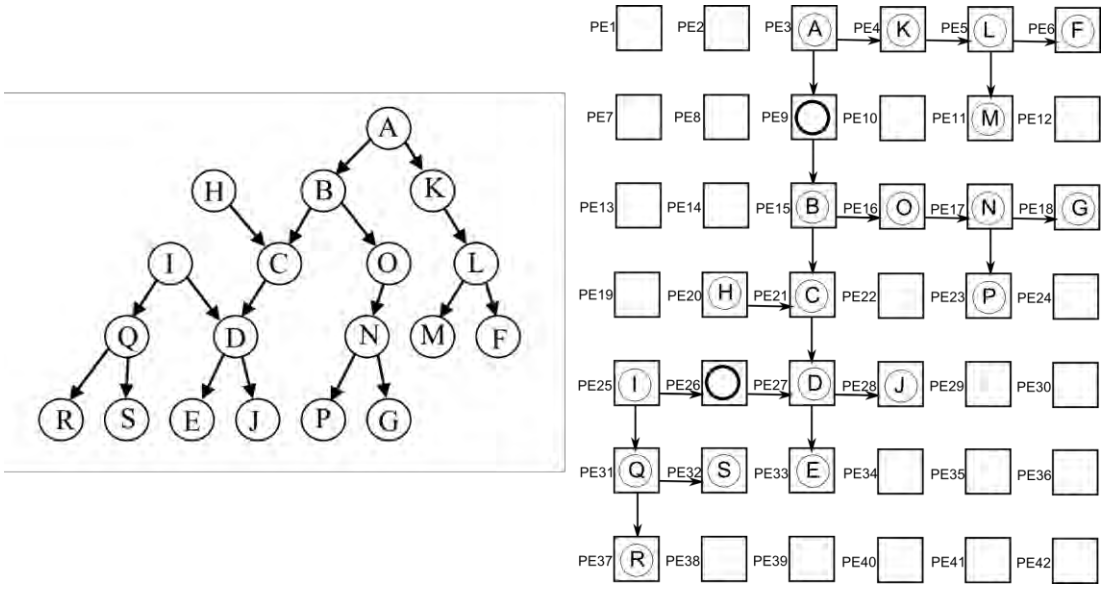


Figura 2.12. Mapeamento da árvore geradora mínima no rDPA

Uma das principais demandas para o desenvolvimento de uma nova arquitetura está relacionada ao balanceamento entre desempenho e o gasto de energia. O trabalho apresentado por (MEHTA et al., 2009) visou reduzir o consumo e os recursos de interconexão para arquiteturas em arranjos de grão-grosso com topologia bidimensional em linha. A Figura 2.13 ilustra a arquitetura proposta. As ALUs (*Arithmetic Logic Unit*) foram organizadas em linha e cada unidade funcional opera independente, desempenhando computação em paralelo, os resultados são passados para a linha seguinte. A comunicação entre as linhas é feita através de uma rede de interconexão completa, ou seja, cada linha possui W ALU(s) e cada ALU da linha inferior irá ligar a todos W elementos de processamento da linha superior. Esta configuração de interconexão tem alto custo com W multiplexadores com W entradas.

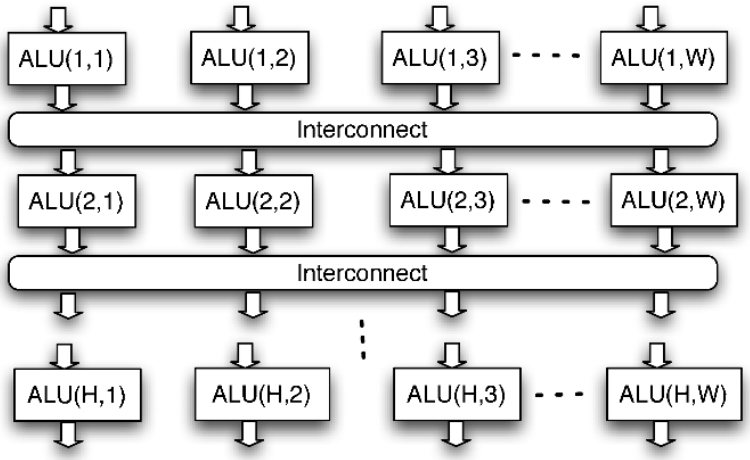


Figura 2.13. Estrutura composta por ALU e rede de conexão (MEHTA et al., 2009)

Para melhorar esta arquitetura (MEHTA et al., 2009) propõe que ao invés de ligar todas ALUs em todas, cada ALU é ligada a um subconjunto de ALUs da linha seguinte. A Figura 2.14 mostra um esquema com multiplexadores de 5:1.

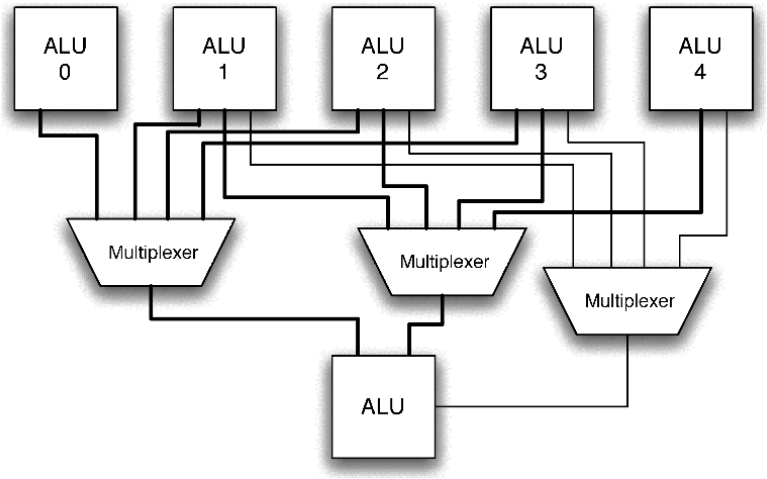


Figura 2.14. Esquema de multiplexação 5:1 (MEHTA et al., 2009)

Multiplexadores com números de entradas variando de 4 a 8 em uma topologia irregular foram analisados. O uso de multiplexadores menores simplifica a complexidade de conexão.

Quando não é possível ligar na próxima linha de ALU, faz-se uso dos pontos de passagens. O ponto de passagem não desempenha computação, apenas repassam o valor para prover a conectividade da arquitetura.

Com a redução da complexidade dos multiplexadores, a energia gasta, por exemplo para o algoritmo *adpcm encoder*, foi reduzida em cerca de 50%. Em relação ao tempo de execução do mapeamento, (MEHTA et al., 2007) mostrou que se a rede

estiver totalmente conectada o tempo é menor que 1 segundo, mas para o caso de Mux 5:1, os tempos são da ordem de segundos podendo chegar até 10 segundos.

Uma arquitetura 2D em linha usando redes multiestágios no processador reconfigurável DS-HIE foi proposta por (TANIGAWA et al., 2008) para reduzir os recursos de *hardware* mantendo a performance. O método de computação *bit* serial foi usado para redução de área, apesar da alta latência de execução das operações. Redes multiestágios Benes (BENES, 1962a), entre as linhas foram adotadas para reduzir a área das interconexões. O diagrama da arquitetura DS-HIE foi apresentado na Figura 2.15.

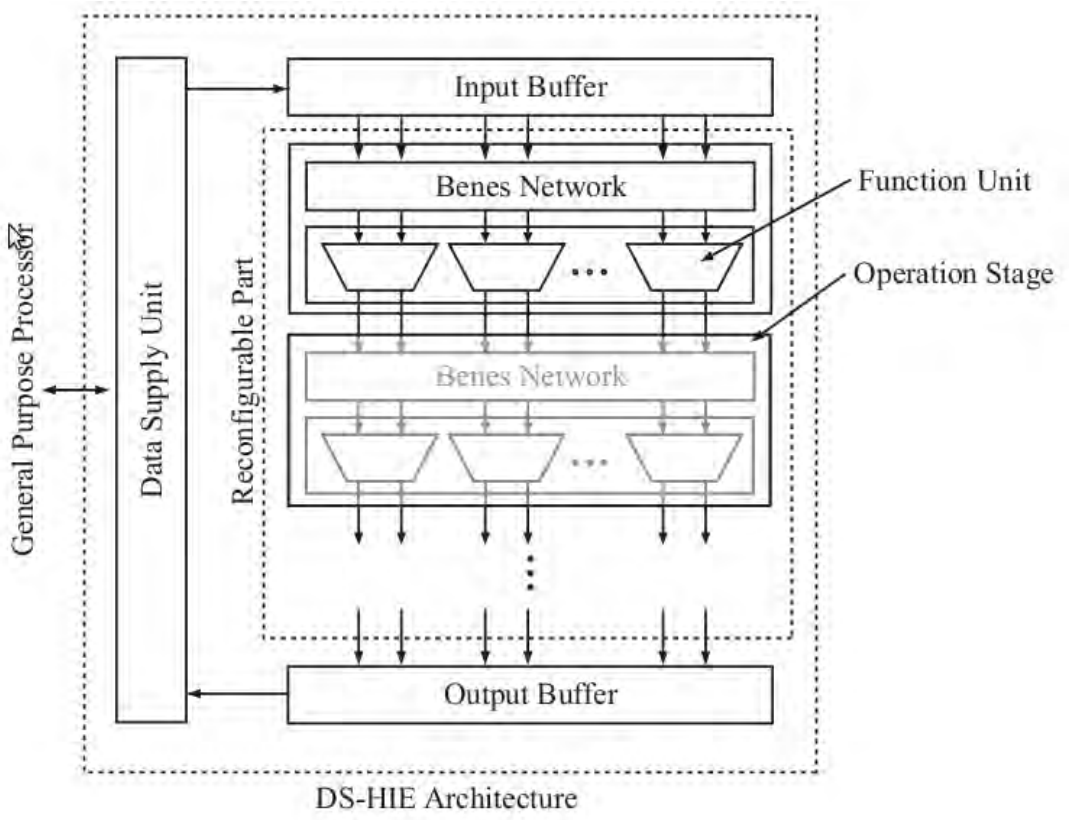


Figura 2.15. Arquitetura DS-HIE (TANIGAWA et al., 2008)

Esta arquitetura usa o mesmo modelo de Metha (MEHTA et al., 2009) com estrutura bidimensional em linha onde a computação de cada linha passa para a linha seguinte.

O mapeamento de quatro exemplos para testes foi feito manualmente e obteve um desempenho semelhante ao processador *Core 2 Duo*. Porém falta desenvolver ferramentas para automatizar o mapeamento. A combinação da rede Benes e computação serial a nível de *bit* garantiu a alta disponibilidade de roteamento dentro

de uma menor área do *chip* e, além disso , demonstrou o potencial das redes multiestágio.

Um outro algoritmo de mapeamento foi proposto em (FERREIRA et al., 2007) com uma heurística de posicionamento com custo polinomial. Para realizar o roteamento usou-se o algoritmo de *Dijsktra*.

A arquitetura é formada por uma grade 2D de PEs com capacidade de roteamento como ilustrado na Figura 2.16, sendo que cada PE possui estrutura interna para conectar cada porta de entrada a um operando da unidade funcional (FU) e cada saída da FU a uma porta de saída do PE, além de rotear conexões para outros PEs.

O mapeamento proposto percorre simultaneamente em profundidade o grafo do DFG e o grafo da arquitetura. No exemplo da Figura 2.17 a ordem na qual as ligações são mapeadas é identificada com um número ao lado de cada aresta do DFG. A busca em profundidade no DFG da Figura 2.17 foi feita aleatoriamente, ou seja, não houve critério para selecionar a próxima aresta a ser percorrida.

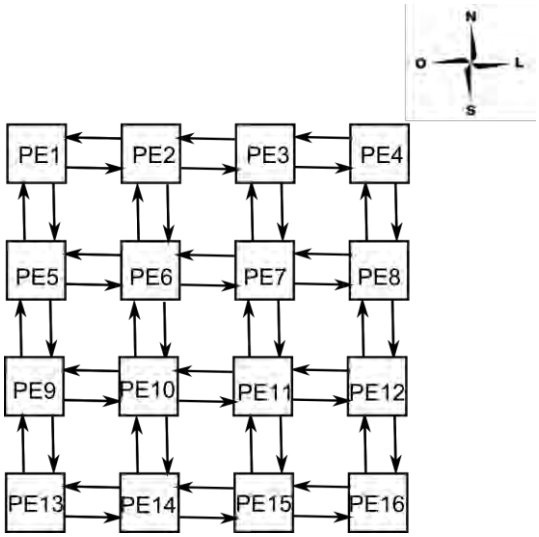


Figura 2.16. Grade 2D de PE

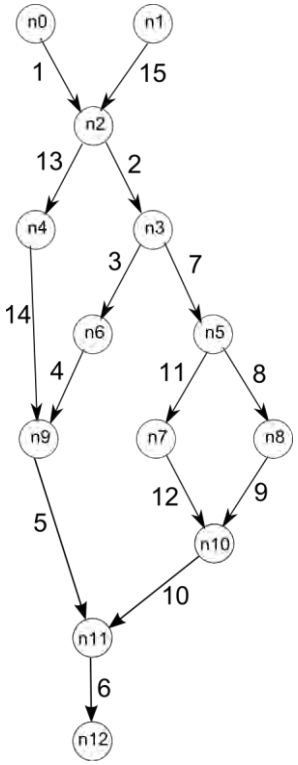


Figura 2.17. DFG com ordem de posicionamento das arestas

A estrutura interna de roteamento dos PEs permite receber sinais do norte, sul, leste e oeste. Estes sinais podem ser ligados a qualquer entrada da FU através de um multiplexador. Cada saída do PE pode receber o sinal da FU ou o sinal de outra entrada. Quando um sinal de entrada é conectado em uma saída sem usar a FU, o PE está realizando uma função de roteamento. Cada saída usa um multiplexador, como ilustra a Figura 2.18. Para um PE com 4 entradas e saídas, são necessários quatro multiplexadores para conseguir todas as possibilidades de roteamento no PE.

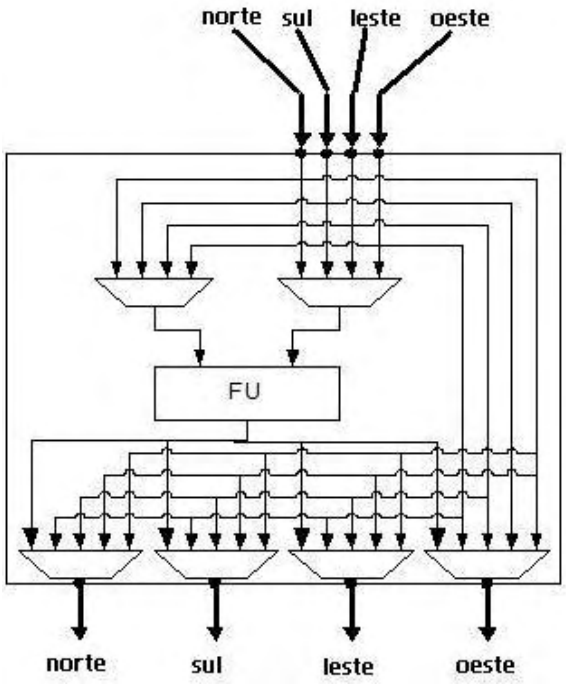


Figura 2.18. PE com capacidade de roteamento.

O algoritmo de posicionamento será detalhado no capítulo 4 e foi utilizado nesta dissertação. Como já foi mencionado, é realizado um caminhamento em profundidade no grafo DFG e no grafo da arquitetura. A maioria das arestas são roteadas localmente, como ilustra a Figura 2.19.

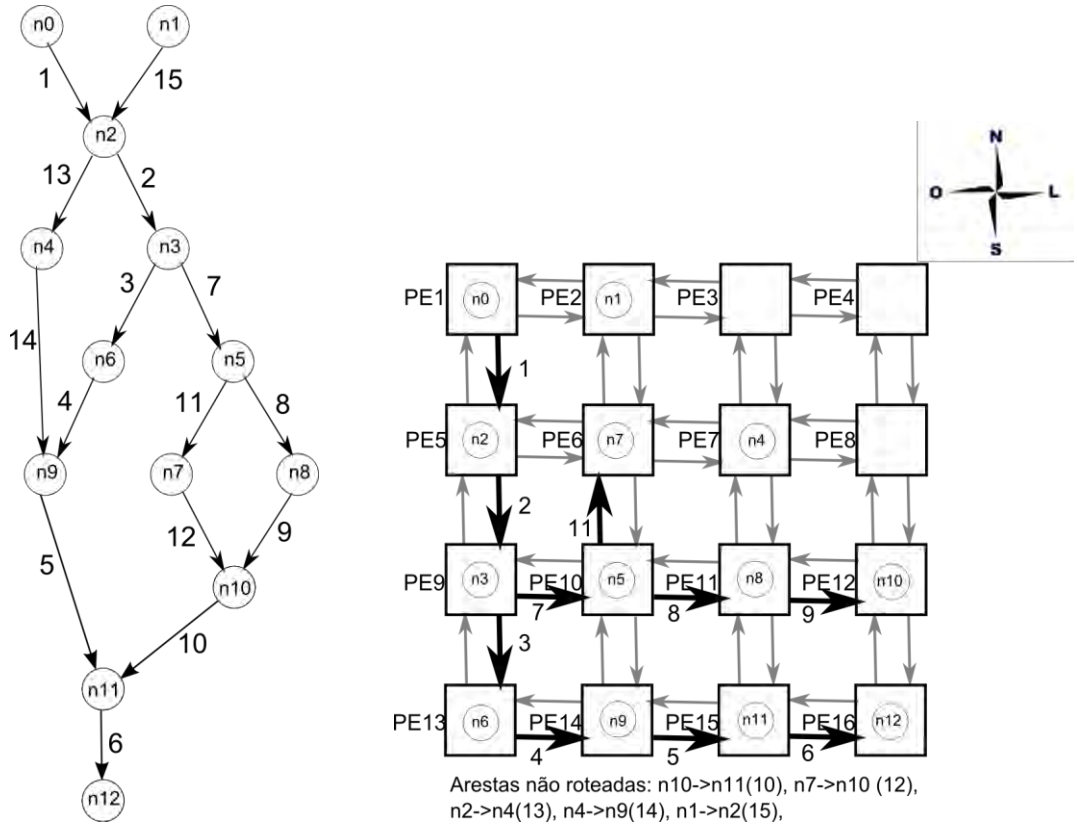


Figura 2.19. P&R na grade 2D de PE.

Porém falta rotear ainda as arestas, n10 → n11, n7 → n10, n2 → n4, n4 → n9 e n1 → n2. Estas arestas estão desconectadas na arquitetura, pois não estão posicionadas em PEs vizinhos, e é necessário executar o algoritmo de roteamento *Dijsktra* para conectá-las, passando por PEs intermediários.

O roteamento das arestas desconectadas é realizado utilizando a capacidade de roteamento dos PEs. A ligação da aresta n4 → n9 na arquitetura passará por dois recursos: PE11e PE15, como ilustra a Figura 2.20. Neste caso o PE conseguiu fazer o roteamento, mas ele é fixo. A versão inicial proposta em (FERREIRA et al., 2007), usou o algoritmo de menor caminho de *Dijsktra* para o roteamento e em (PIMENTEL, 2007) o roteamento foi substituído pelo algoritmo *Pathfinder*.

O posicionamento tem custo polinomial, e o algoritmo de roteamento baseado no algoritmo de *Dijsktra* também, usa a abordagem gulosa, faz o roteamento pela primeira rota mais curta. Uma desvantagem é o custo da arquitetura, ao inserir a capacidade de roteamento em cada PE. Em geral, são necessários de 8 a 12 conexões de entrada e saída por PE, que aumenta o custo de multiplexadores que farão o roteamento do circuito. Além disso, podem ser necessárias ligações toroidais nas extremidades para evitar a concentração de roteamento na região central da arquitetura.

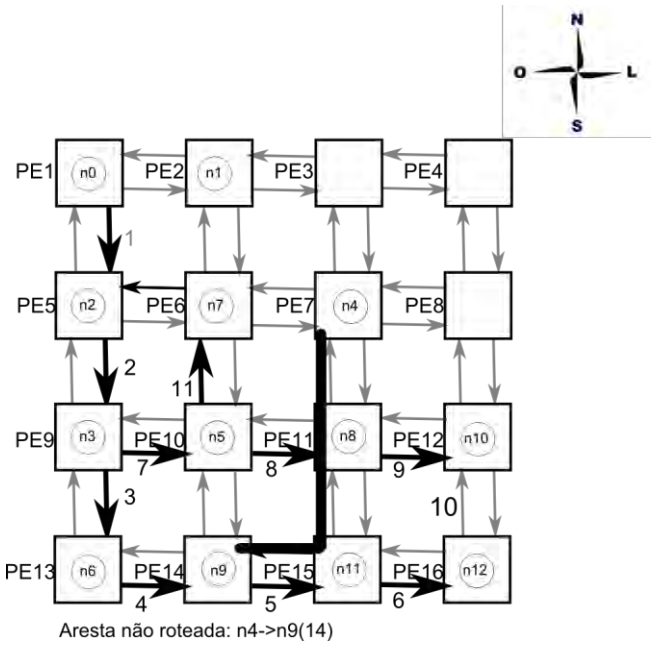


Figura 2.20. Roteamento através dos PE, aresta n4→n9

Em relação a outros algoritmos polinomiais, como por exemplo, o algoritmo baseado em árvores geradoras proposto por Lai em (LAI; LAI; YEH, 2005), o mapeamento baseado em profundidade é eficiente. Ambos usam uma abordagem baseada em grafos. A proposta de (LAI; LAI; YEH, 2005) usou a arquitetura rDPA com poucos recursos de roteamento local, o preço a ser pago é uma grande área e vários PEs ociosos, como ilustrado na Figura 2.12. Já a abordagem baseada em profundidade usa um PE com recursos de roteamento, e gera mapeamentos compactos em área. A arquitetura rDPA é limitada à ligações locais apenas com os vizinhos da frente e abaixo do DPU, em contrapartida a abordagem baseada em profundidade usa 4 ou mais vizinhos por PE, além do encaminhamento no PE.

O mapeamento da árvore geradora mínima na arquitetura rDPA consumiu 68% a mais de área com taxa de ocupação de 45% da área reservada, para o exemplo da Figura 2.12. Duas ligações usaram um nodo de encaminhamento, representado no rDPA pelo círculo escuro. Já o mapeamento por profundidade, apresentado na Figura 2.21, teve 76% de taxa de ocupação e quatro arestas não foram roteadas localmente. O preço são os recursos de roteamento, devido aos multiplexadores internos para cada saída do PE. Nesta dissertação mostraremos como reduzir este custo com a utilização de PE sem capacidade de roteamento e com a adição de uma rede global baseada em multiestágio. Estes pontos serão detalhados nos capítulos seguintes.

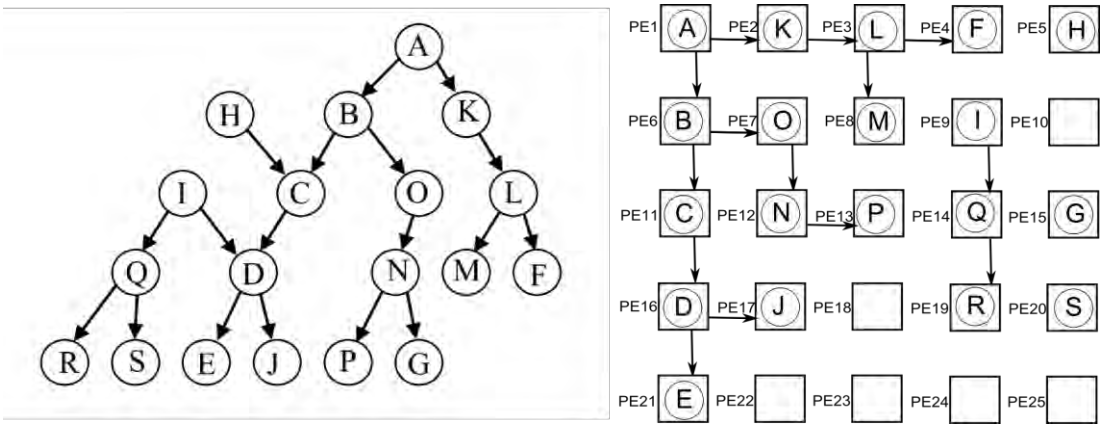


Figura 2.21. Mapeamento da árvore geradora mínima na grade 2D

2.5 Compilador *just-in-time* para FPGA

O ROCR, *Riverside On-Chip Router*, (LYSECKY; VAHID; TAN S. X.-D, 2004) propõe a geração de um mapeamento eficiente na arquitetura do FPGA para atender as demandas de compilação *Just-In-Time* (JIT).

O mapeamento baseia-se no roteador VPR (BETZ; ROSE; MARQUARDT, 1999). Diferencia do VPR ao rotear com um menor grafo de recursos e gera economia de memória. Os grafos são no nível de portas lógicas. Os *benchmarks* MCNC (YANG, 1991) foram avaliados e comparados ao VPR. O ROCR obteve um melhor desempenho, sendo 10x mais rápido que o roteamento do VPR (*timing-driven*) e consumiu 13x menos memória.

2.6 Considerações finais

Neste capítulo, inicialmente foram apresentados os algoritmos de posicionamento (*Simulated Annealing*) e roteamento (*Pathfinder*) em arquiteturas de grão fino e grão grosso. Foram apresentadas também novas arquiteturas para FPGA, onde a rede física utiliza a abordagem multiestágio. Estas soluções mostram que a rede multiestágio é uma alternativa promissora para interconexões em circuitos reconfiguráveis.

Dentre as abordagens para mapeamento em grão grosso, poucas priorizam o tempo de execução da síntese, dificultando sua utilização em ambientes de compilação JIT. Uma alternativa é o algoritmo de posicionamento polinomial (FERREIRA et al., 2007), entretanto o roteamento por encaminhamento nos PEs aumenta a complexidade da arquitetura.

A proposta desta dissertação difere dos trabalhos anteriores em vários aspectos. Primeiro, uma nova arquitetura será apresentada e implementada sobre um FPGA tradicional, como *soft-core*, no nível lógico. A arquitetura proposta simplifica os recursos de roteamento local, reduzindo o custo. O algoritmo de posicionamento polinomial proposto em (FERREIRA et al., 2007), foi adaptado. Para resolver o roteamento global, esta dissertação apresenta como contribuição o acoplamento de redes multiestágios, preservando a complexidade polinomial do algoritmo sem aumentar muito o custo em *hardware*. Além disso, analisou-se o posicionamento priorizando os nodos do caminho crítico do DFG, modificando o algoritmo original de posicionamento.

O próximo capítulo apresenta a arquitetura proposta. As modificações no algoritmo de posicionamento e roteamento são detalhadas no Capítulo 4.

3 ARQUITETURA HÍBRIDA

A arquitetura reconfigurável de grão grosso (CGRA) proposta neste trabalho é composta por uma grade de elementos de processamento (PE) em duas dimensões acoplada a uma rede de interconexão (MIN). Na topologia em grade 2D, os PE(s) podem se conectar localmente ou globalmente. A maioria das conexões será realizada através de conexões locais entre os PEs vizinhos locais. A segunda opção de interconexão são conexões globais realizadas pela rede Omega MIN, pela qual se conectam os PEs não vizinhos. Neste capítulo, iremos detalhar a arquitetura e os algoritmos de posicionamento e roteamento serão apresentados no próximo capítulo.

Inicialmente, iremos apresentar a arquitetura e como um grafo é mapeado. Posteriormente, uma etapa de pré-processamento do grafo para satisfazer as restrições da arquitetura será introduzida na seção 3.1. O modelo de conexão multiestágio é revisto na seção 3.2 e na seção seguinte o contexto de uso das redes multiestágios será apresentado, que é diferente das abordagens anteriores. Finalmente, apresentamos os resultados do custo de implementação em FPGA que foram um fator motivador para o desenvolvimento da arquitetura descrita aqui.

A Figura 3.1 mostra um exemplo da arquitetura com apenas 4 PEs. Neste exemplo, o PE3 pode se conectar aos seus vizinhos, PE1 e PE4, usando conexão local. Para conectar ao PE2, que não é vizinho, usa-se a conexão global. Este modelo de arquitetura será configurado sobre um FPGA tradicional, como um *soft-core*.

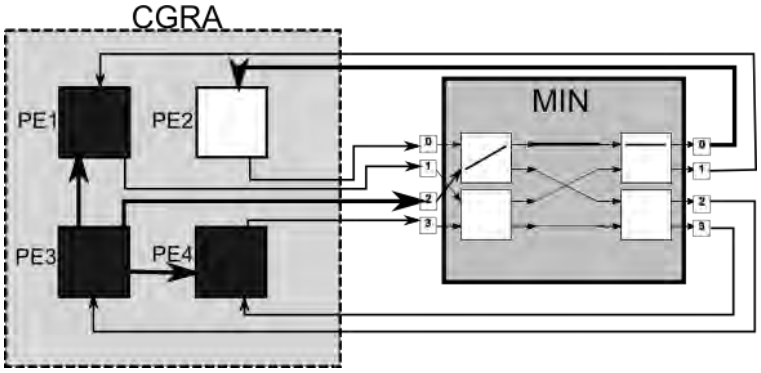


Figura 3.1. Vizinho local e global do PE3

O acoplamento da rede Omega na arquitetura em grade será exemplificado a partir do grafo para a computação de c = a + b. O grafo pode ser mapeado na arquitetura, como ilustrado na Figura 3.2. Os nodos **a**, **b**, **c** e **+** do DFG foram mapeados na arquitetura CGRA nos elementos de processamento PE1, PE2, PE4 e PE3, respectivamente. O roteamento local só foi possível para as seguintes conexões: PE1→PE3 e PE3→PE4. A conexão PE2→PE3 utilizou a rede Omega MIN (roteamento global), através do terminal 0 de entrada e terminal 2 de saída que fazem o roteamento entre PE2 e PE3.

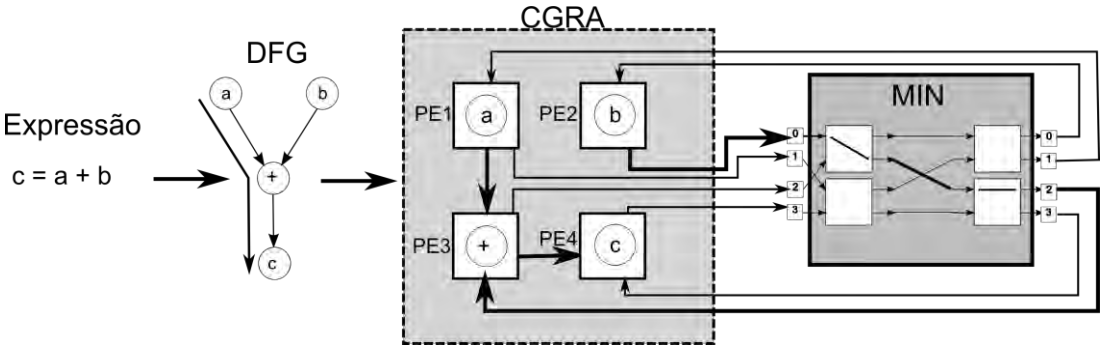


Figura 3.2. Conversão expressão em DFG e mapeamento CGRA com MIN

Para reduzir o custo das conexões na grade, o PE se conecta apenas aos PEs vizinhos diretos (norte, sul, leste, oeste) que não possuem capacidade de roteamento. Essa proposta simplifica a arquitetura apresentada em (FERREIRA et al., 2007), em que os PE possuem a capacidade de roteamento. Como já foi mencionado, o mapeamento é realizado em dois passos, no primeiro é feito o posicionamento dos nodos na arquitetura juntamente com o roteamento local, e depois o roteamento global é realizado.

Os PEs são responsáveis por executar operações aritméticas e lógicas e desvios condicionais. A grade é formada por PEs homogêneos, ou seja, todos tem a mesma capacidade de processamento. Assim qualquer nodo pode ser alocado em qualquer PE. O próximo exemplo de mapeamento (veja a Figura 3.3), ilustra um caso onde foi necessário utilizar duas MIN para conectar todos PEs da arquitetura. Foram desenhadas apenas as conexões ativas na arquitetura e na MIN.

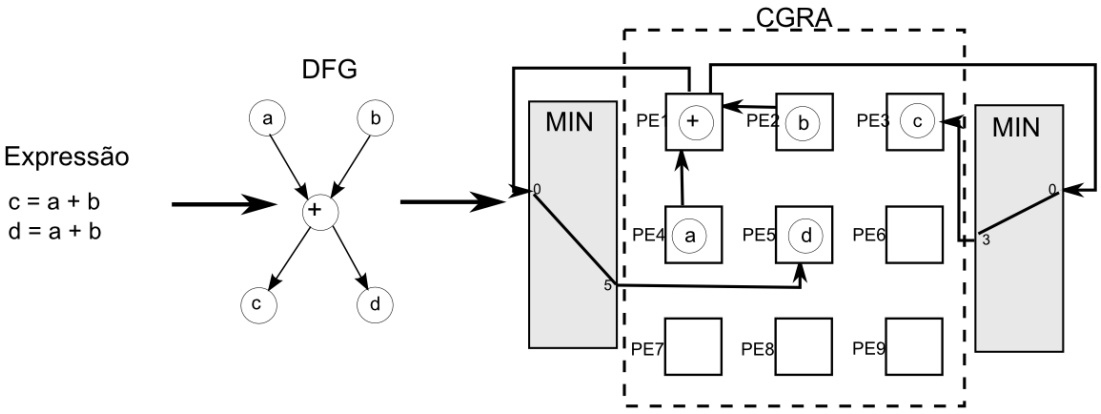


Figura 3.3. Conversão expressão para DFG e mapeamento na CGRA com duas MIN

O posicionamento dos nodos DFG na arquitetura CGRA ocorreu da seguinte forma: os nodos **a**, **b**, **c**, **d** e **+** foram alocados em PE4, PE2, PE3, PE5 e PE1, respectivamente. As arestas roteadas localmente foram **b**→ + e **a**→+. As arestas +→**c** e +→**d** não foram roteadas localmente na arquitetura pois não existe conexão direta entre os PE1→PE3 e PE1→PE5. Observou-se que ambas tem o mesmo elemento de processamento, PE1. Serão necessárias duas redes MINs. Se a conexão PE1→PE3 for roteada pela MIN da direita, o PE1→PE5 utilizará a MIN da esquerda. Como será mostrado no capítulo 5, para o conjunto de aplicações avaliadas, no máximo duas redes foram necessárias para realizar o roteamento global.

3.1 Decomposição do grafo antes do mapeamento

Os DFGs foram decompostos para satisfazer as restrições da arquitetura onde cada unidade de processamento (PE) tem internamente uma ALU com duas entradas e duas saídas. A decomposição irá configurar cada nodo de forma que não tenha mais que duas arestas de entradas e duas arestas de saída. Suponha que um nodo deva realizar a operação de soma de n arestas de entradas E1, E2, E3 e E_n, respectivamente. A decomposição insere novos nodos como ilustra a Figura 3.4.

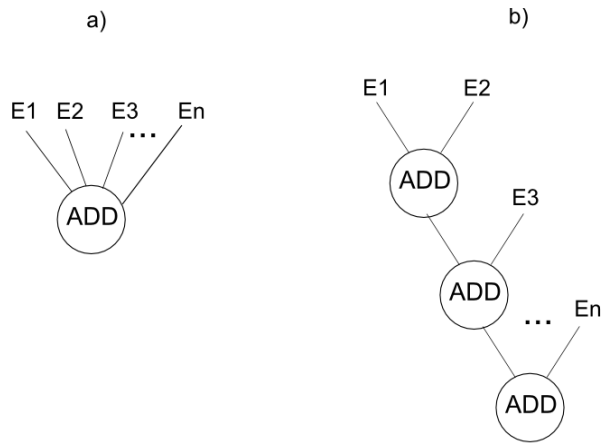


Figura 3.4. Decomposição entrada a) nodo com n entradas b) nodo decomposto

Para realizar a decomposição dos nodos com mais de duas arestas de saída são inseridos nodos de cópia de valores, COPY, para propagar o valor da saída do nodo, como ilustra a Figura 3.5. Os nodos auxiliares inseridos no DFG não irão alterar as dependências de execução das operações, desta forma será mantida a integridade dos grafos. Na primeira versão foi implementada uma decomposição desbalanceada, porém uma implementação balanceada é mais adequada para minimizar o caminho crítico.

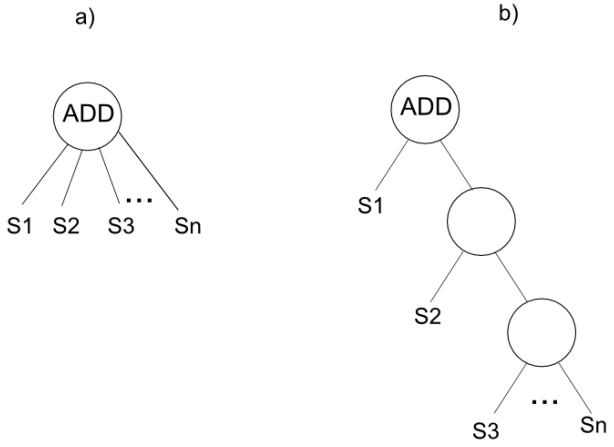


Figura 3.5 Decomposição saída a) nodo com n saídas b) nodo decomposto

3.2 Multiestágio

Nesta seção iremos rever as principais características das redes multiestágios. As redes de interconexão multiestágio (MINs) tem como objetivo conectar um dispositivo de entrada a um dispositivo de saída através de estágios de elementos de chaveamento. Cada elemento de chaveamento é uma rede completa, normalmente 2x2 (DUATO; YALAMANCHILI; NI, 2003). A capacidade de roteamento de uma MIN é determinada pelo número de estágios e pelos padrões de ligação entre os

estágios. As MINs foram inicialmente propostas para atender as redes de telefonia nos anos 50 (CLOS, 1953)(BENES, 1962b) e posteriormente foram muito utilizadas em arranjos de processadores e multiprocessadores nas décadas de 70 e 80.

Existem muitas formas de realizar as permutações entre os estágios adjacentes da rede. A Figura 3.6 mostra uma rede MIN genérica com N entradas e M saídas com *g* estágios de G_0 até G_{g-1}.

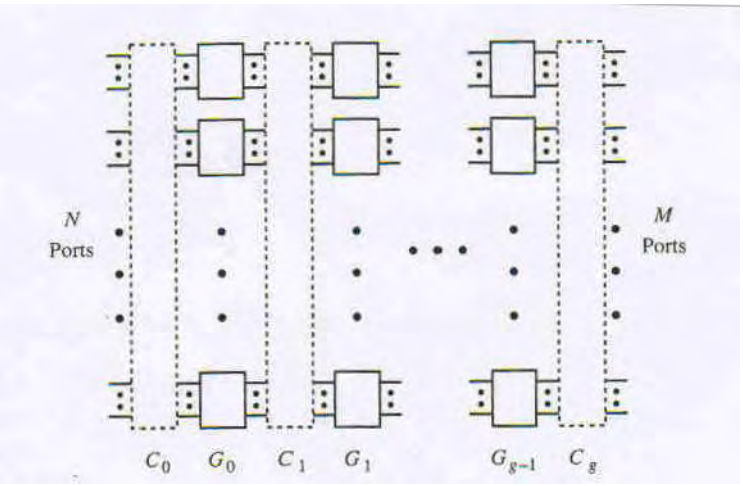


Figura 3.6. MIN genérica (DUATO; YALAMANCHILI; NI, 2003)

Alguns dos principais padrões de permutação serão apresentados a seguir (DUATO; YALAMANCHILI; NI, 2003). Suponha que a seguinte configuração com 8 terminais, chaves 2x2 e 3 estágios de chaves. Os endereços de cada linha serão especificados pela sequência de *bits* $x_2x_1x_0$.

O padrão de permutação *Perfect shuffle* é o padrão usado na rede Omega. Ele executa um deslocamento cíclico (rotação) dos *bits* para esquerda, este padrão esta ilustrado na Figura 3.7. Ou seja, a linha 3 (011) é ligada à linha 6 (110).

Padrão
$i_2i_1i_0$ -> $i_1i_0i_2$

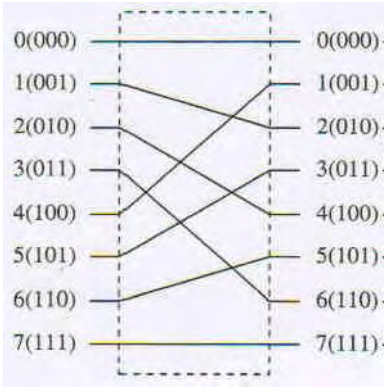


Figura 3.7. Padrão *Perfect Shuffle* (DUATO; YALAMANCHILI; NI, 2003)

O padrão de permutação *butterfly* está relacionado ao estágio C_i da rede. O endereço de destino é obtido pela permutação dos *bit* x_i e x_0. A Figura 3.8 ilustra a

conexão para diferentes estágios. Neste padrão, as conexões mudam em função do estágio da MIN.

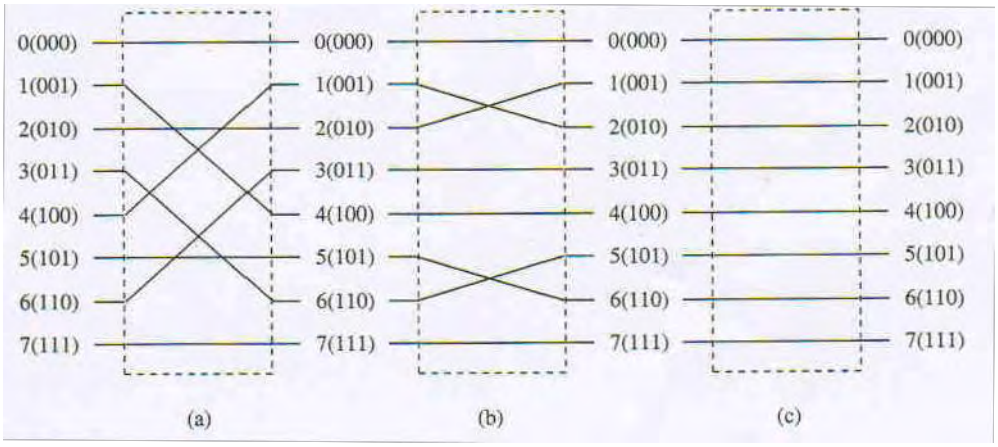


Figura 3.8. Permutação *butterfly* a) segunda ordem b) primeira ordem c)zero ordem (DUATO; YAMALANCHI; NI, 2003)

O padrão de permutação *cube* também depende do estágio de interconexão C_i. O endereço do terminal de destino é obtido pelo complemento do bit x_i. A Figura 3.9 ilustra a conexão dos estágios na rede.

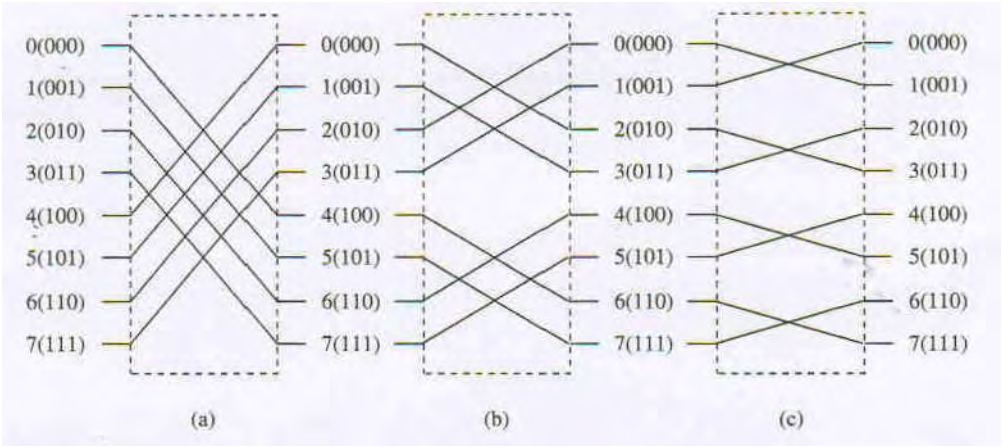


Figura 3.9. Permutação *cube* a) segunda ordem b) primeira ordem c) ordem zero (DUATO; YALAMANCHILI; NI, 2003)

As redes multiestágios podem ser classificadas conforme a capacidade para realizar as conexões (DUATO; YALAMANCHILI; NI, 2003). São divididas em três classes:

- Bloqueante: não executa todas as permutações completas de entrada/saída. Para redes com $\log_2 N$ estágios existe apenas um único caminho entre um par de entrada e saída. Ao se adicionar estágios extras, aumenta-se o número de caminhos, porém algumas permutações não serão realizadas, pois os múltiplos caminhos são compartilhados e geram conflitos.

- Não-bloqueante: a porta de entrada pode conectar-se a qualquer porta de saída livre sem afetar outras ligações, independente da ordem em que as ligações são feitas. O número de estágios é no mínimo 3 $\log_2 N$, ou seja, três vezes maior que uma rede bloqueante que tem $\log_2 N$.
- Rearranjáveis: qualquer porta de entrada pode ligar a qualquer porta de saída livre. Entretanto, durante o roteamento, ao se adicionar um novo par de entrada/saída, as ligações anteriores podem ser modificadas e rearranjadas. Usando algoritmos semelhantes à coloração de grafos, a rede é programada para qualquer permutação completa. Possuem um custo menor que as redes não bloqueantes, em geral 2 $\log_2 N$ - 1. A rede BENES é a rede rearranjável mais estudada (BENES, 1962b). Mas tem o dobro de custo das redes bloqueantes.

Neste trabalho optamos por uma rede bloqueante, devido ao custo reduzido com $\log_2 N$ estágios. Como iremos mostrar, apenas 2 ou 4 estágios extras serão necessários para resolver os conflitos de roteamento. Nosso contexto envolve permutações parciais, como será discutido na próxima seção, diferente dos outros trabalhos, onde a rede é usada para interligar todas as entradas e saídas ao mesmo tempo. Usamos uma rede com o padrão *perfect shuffle* (rede Omega) mas outras redes poderiam ser utilizadas como a *butterfly* ou *baseline* que são funcionalmente equivalentes. Alguns exemplos comerciais de computadores paralelos que utilizam redes multiestágios são (DUATO; YALAMANCHILI; NI, 2003):

- Thinking Machines CM-5: utiliza a topologia *fat tree* com vias de 4 bits bidirecionais e um canal de 40 Mhz, com uma largura de banda de 20Mbytes/s em cada direção.
- IBM SP2: utiliza uma rede Omega com vias bidirecionais de 16 bits a 150MHz que permite transmissão de 300Mbits em cada direção.
- SGI SPIDER: Suporta configurações de redes multiestágio não bloqueantes e topologias irregulares.

3.2.1 Uso total e parcial da multiestágio

A capacidade de roteamento de uma rede multiestágio (MIN) *SW-banyan* é um problema complexo para se criar um modelo analítico (GAZIT; MALEK, 1989). Porém, através de simulações, pode-se estimar o número de permutações que a rede realiza sem conflito no universo de N! permutações.

Além de avaliar a capacidade de roteamento da rede *banyan*, um estudo da capacidade da rede com adição de um estágio extra foi apresentado em (GAZIT; MALEK, 1989). A Tabela 3.1 apresenta os resultados deste estudo. Este resultado também se aplica a rede Omega por pertencer a mesma classe de MIN.

Tabela 3.1. Conexões realizadas na *SW-banyan* e adição de estágio extra

N	*SW-banyan*	Estágio Extra	N!
2	2	2	2
4	16	24	24
8	4096	18688	40320
16	~4,3 x 10^{9}	~2,7 x 10^{11}	~2,1 x 10^{13}
32	~1,2 x 10^{24}	~2 x 10^{28}	~2,6 x 10^{35}

Considerando N! o número total de permutações, para N=32, o número de conexões é da ordem de 10^{35}. O valor é muito superior ao número de permutações conectadas sem conflito que é da ordem 10^{24}. Ao se adicionar um estágio extra, o número de permutações com sucesso aumenta para 10^{28}.

Se a rede MIN for rearranjável todas as conexões podem ser realizadas. Porém, a rede terá o dobro de estágios, 2 . $\log_2$ N, ou seja, o custo dobra, bem como a latência. Além disso, é necessário o conhecimento a priori da permutação completa para realizar o roteamento.

A abordagem utilizada nesta dissertação está contextualizada em permutações parciais, ou seja, um subconjunto de entrada se conecta a um subconjunto de saída. As figuras 3.10, 3.11, 3.12 ilustram a conexão parcial com os seguintes percentuais de terminais: 100%, 50% e 25%.

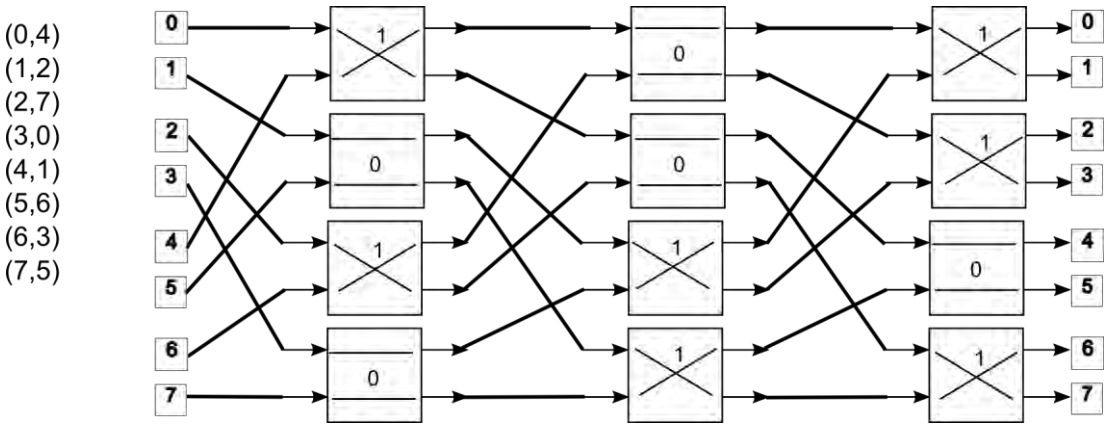


Figura 3.10. Omega MIN com N=8 e 100% (8) dos terminais usados

(2,7)
(3,0)
(4,1)
(6,3)

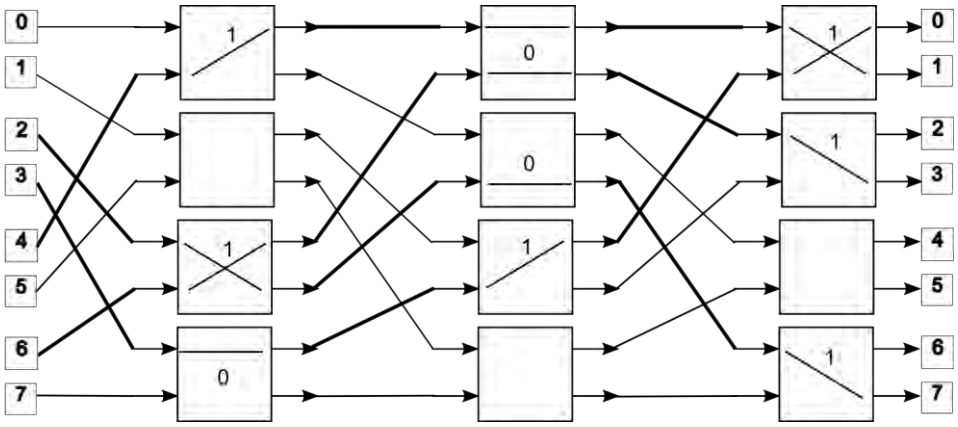


Figura 3.11. Omega MIN com N=8 e 50% (4) dos terminais usados

(2,7)
(4,1)

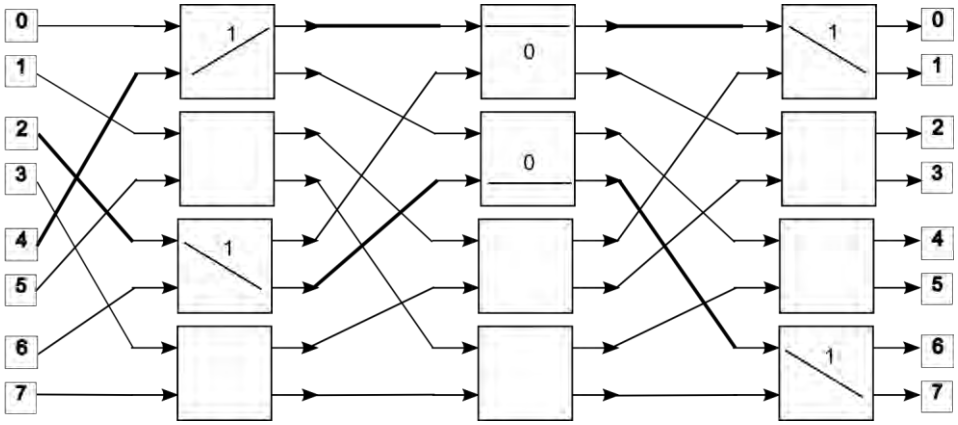


Figura 3.12. Omega MIN com N=8 e 25% (2) dos terminais usados

Neste trabalho foi realizado um estudo por amostragem para avaliar a capacidade de roteamento da MIN, usando permutações parciais, que será apresentado no capítulo de resultados. A rede multiestágio bloqueante com poucos estágios extras atende as necessidades para a demanda de permutações parciais com taxa de uso inferiores a 40%.

3.2.2 Implementação das MIN em FPGA

Uma das motivações para o modelo de arquitetura usado nesta dissertação foi um estudo do espaço ocupado pelas MIN e pelos PEs quando mapeados em uma arquitetura FPGA. Este estudo foi desenvolvido em paralelo e de forma complementar a esta dissertação. Fazem parte deste estudo um trabalho de IC (Iniciação Científica) (VENDRAMINI, 2009) e um relatório de pesquisa (FERREIRA, 2010) desenvolvido em parceria com o pesquisador João Cardoso da Faculdade do Porto (FEUP) – Portugal. Foi analisado o gasto de área da rede MIN em função do número de LUTs *(LookUp Tables*) e a frequência de *clock*.

A rede Omega foi sintetizada com a ferramenta IDE Xilinx (XILINX, 2010). A síntese da rede acompanha, em termos de área, a complexidade $O(N.\log_2 N)$. Dois padrões de interconexão foram comparados, a Omega e a *Butterfly*. Além disso, foi feito um estudo com os elementos de chaveamento com 4x4, ou seja, 4 entradas e 4 saídas, denominado por Radix4. Esta abordagem aumenta a complexidade do elemento de chaveamento porém reduz o número de estágios. Os resultados são apresentados na Tabela 3.2, onde NA (Não se Aplica).

Tabela 3.2. Custo das redes LUTs, redes com largura de 32 bits (VENDRAMINI, 2009)

Número de entradas (N)	Rede Omega	Rede Radix-4	Rede Butterfly
8	576	NA	576
16	1824	1024	1472
32	4096	NA	4096
64	9216	6144	9216
128	22528	NA	22528
256	49152	32768	49152

As redes Omega e *Butterfly* tiveram resultados semelhantes. A radix-4 teve menor custo por causa da complexidade de $(N.\log_4 N)$, porém apresenta mais conflitos de roteamento.

Foi avaliado também a utilização da rede Omega em arquiteturas de grão-grosso. Cada arquitetura sintetizada tinha 16 unidades de processamento e a rede Omega com largura de 16 bits. A área utilizada é apresentada na Tabela 3.3.

Tabela 3.3. Área de ocupação da arquitetura (VENDRAMINI, 2009)

Arquitetura	Ocupação (LUTs)
Arquitetura com capacidade de roteamento	13597
Arquitetura sem capacidade de roteamento	4898
Arq. sem capacidade de roteamento com uma rede Omega	6240
Arq. sem capacidade de roteamento com duas redes Omega	7952

A área ocupada pela arquitetura com recurso de roteamento agregado ao PE apresentado por (FERREIRA et al., 2007), é quase 3x maior que a mesma arquitetura sem capacidade de roteamento que é a proposta desta dissertação. Acoplando duas redes Omega, a área ainda é 30% menor que a área da arquitetura com recurso de roteamento.

Para avaliar os custos de processamento e atrasos, a arquitetura foi sintetizada em uma Xilinx Virtex-5 FPGA (xc5vlx330ff1760-2) e considerou-se PEs com 32 bits e unidade funcional (FU) capaz de realizar operações aritméticas (adição, multiplicação, subtração, negação e comparação), deslocamento e operações de bits (AND, OR, XOR e NOT).

A Figura 3.13 apresenta os recursos do FPGA (#LUTs) utilizados para diferentes tamanhos de redes Omegas com entrada/saída de 32 *bits*. O custo da área em LUTs é uma função de ($N.\log_2 N$). Na Figura 3.14, o atraso aumentou em aproximadamente 0,5 ns para cada terminal de entrada/saída adicional na rede. E uma rede Omega com 128 entradas/saídas usou apenas 11% dos LUTs disponíveis no FPGA.

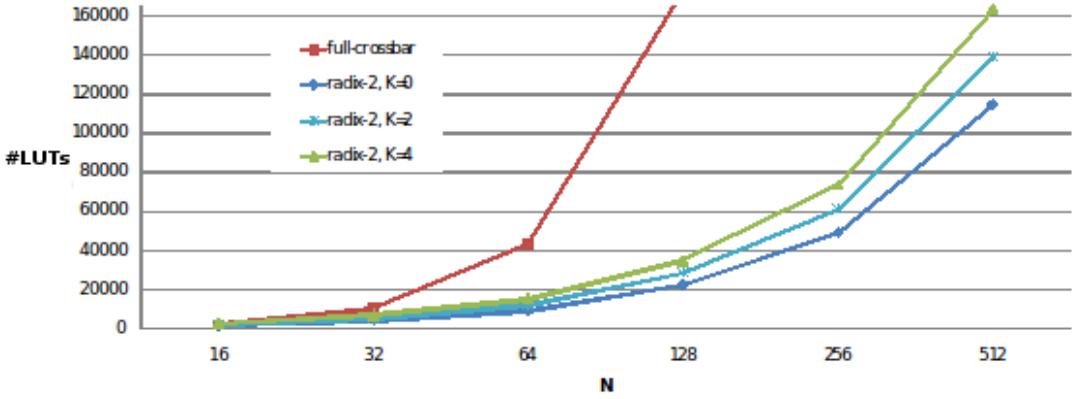


Figura 3.13. Ocupação LUTs pela rede Omega com radix-2 de 32 bits (FERREIRA, 2010)

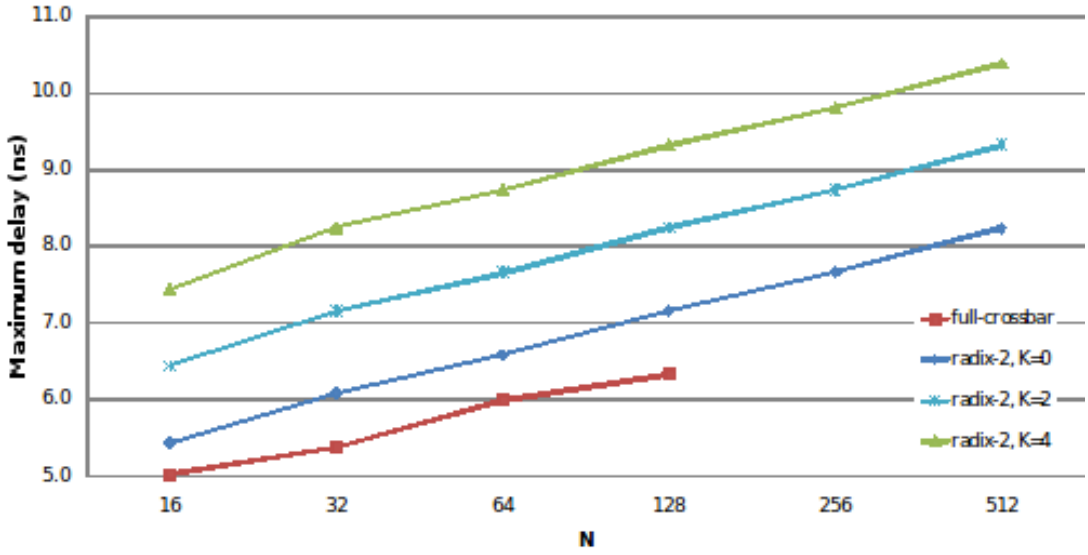


Figura 3.14. Atraso máximo na rede Omega com radix-2 de 32 bits (FERREIRA, 2010)

4 POSICIONAMENTO E ROTEAMENTO NAS CGRA

4.1 Introdução

A complexidade do problema de posicionamento e roteamento (P&R) pode comprometer uma solução que exige compilação dinâmica, *just-in-time*, devido ao fato de ser um problema NP-Completo (TESSIER, 1999). Várias abordagens utilizadas para o P&R são baseadas em heurísticas: algoritmos gulosos, *simulating annealing* e algoritmos genéticos (HARTENSTEIN, 2001a).

A solução para o P&R proposta neste trabalho, é uma heurística polinomial implementada em dois passos. Primeiro é realizado o posicionamento e roteamento local de um DFG em uma arquitetura reconfigurável de grão grosso (CGRA). A Figura 4.1 ilustra um exemplo de CGRA. O P&R irá posicionar os nodos de um DFG, como ilustrado na Figura 4.2 sobre o CGRA. O segundo passo é o roteamento global com uma rede de interconexão multiestágio (MIN). O DFG é representado por G(V,E), onde V é o conjunto de nodos que representam as operações e E as arestas que representam as relações de dependências das operações. O grafo representa uma relação de dependência, pois uma operação não é processada antes que as operações imediatamente precedentes sejam finalizadas. Por exemplo, o nodo n2 só é ativado, depois que os nodos n0 e n1 forem executados, (vide Figura 4.2). O CGRA também será representado por um grafo direcionado onde cada nodo representa um elemento de processamento (PE) e as arestas são as conexões entre os PE. Cada operação do DFG deve ser alocada em um PE da CGRA, ou seja, é o mapeamento de um grafo em outro grafo. Para facilitar a explicação do algoritmo, iremos usar os termos nodos e arestas para referenciar o grafo do DFG, e os termos PE e conexão para referenciar o grafo da arquitetura, o CGRA. O termo grafo será utilizado para se referir ao DFG. Usaremos o termo arquitetura ou arquitetura reconfigurável para o CGRA.

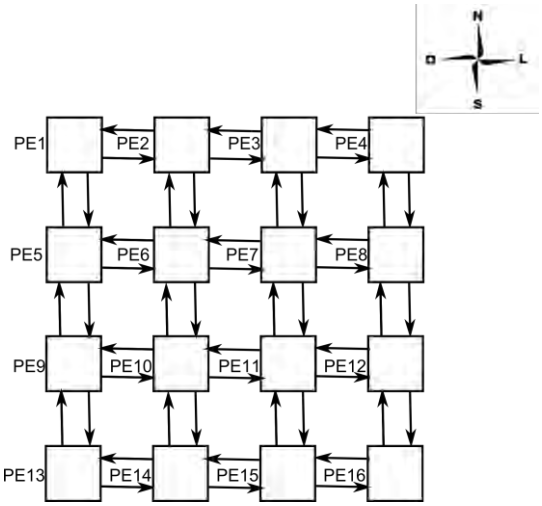


Figura 4.1. CGRA

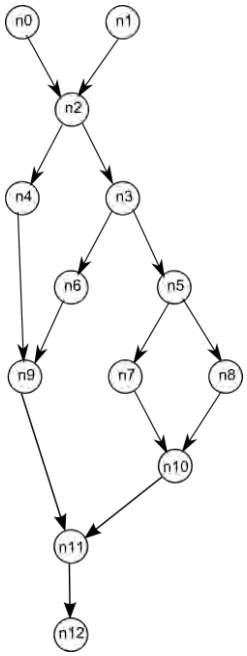


Figura 4.2. Grafo de entrada DFG

Foram desenvolvidos três algoritmos para realizar o posicionamento e roteamento do DFG no CGRA. O primeiro algoritmo é semelhante ao algoritmo apresentado em (FERREIRA; et al., 2007), porém a abordagem proposta realiza apenas o roteamento local e uma rede multiestágio foi acoplada para resolver as conexões globais. Cada algoritmo possui um critério de posicionamento buscando um balanceamento entre o tempo de execução do algoritmo e a qualidade do posicionamento em termos de caminho crítico. A diferença está na implementação da função de busca em profundidade que percorre o DFG durante o posicionamento. Iremos referenciar os algoritmos pela função de busca como: (a) busca em profundidade; (b) busca em profundidade priorizando nodos no caminho crítico (c) busca em profundidade somente dos nodos no caminho crítico. Na seção 4.2 apresentamos os algoritmos de posicionamento e roteamento local e na seção seguinte o roteamento global.

4.2 Posicionamento e roteamento local

4.2.1 Busca em Profundidade

A busca em profundidade consiste em percorrer o DFG para realizar o posicionamento e roteamento local na arquitetura reconfigurável proposta. O posicionamento é feito ao alocar os nodos do DFG nos PE do CGRA. O roteamento

conecta os PE vizinhos que receberam os nodos de origem e destino de uma aresta do DFG.

O número de vizinhos de cada PE depende de sua posição na grade, os PE de extremidade, o de borda e o central, possuem dois, três e quatro vizinhos, respectivamente (Figura 4.3).

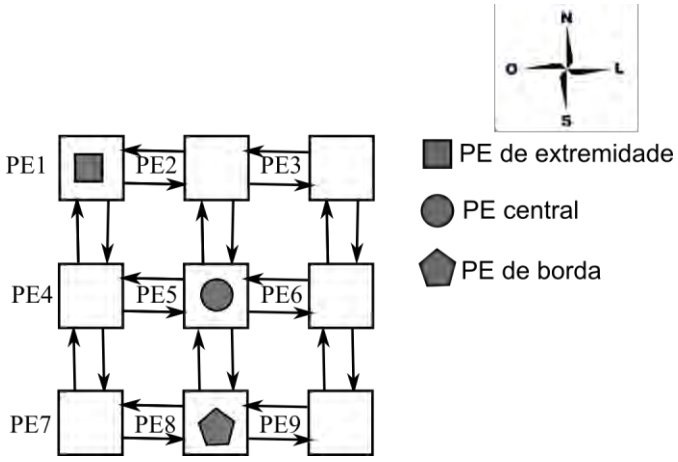


Figura 4.3. Elementos de processamento (PE) vizinhos

Nesta seção serão descritas as etapas de P&R executados na arquitetura em grade (Figura 4.1). O DFG, representado na Figura 4.2, será usado em todos os exemplos, denominado como grafo de entrada para o algoritmo. Para garantir que todos os nodos possam ser mapeados em um elemento de processamento, o processo de decomposição é aplicado no DFG antes de realizar o P&R. O objetivo da decomposição é garantir que todos os nodos do grafo tenham no máximo duas arestas de entrada e duas arestas de saída, viabilizando o seu mapeamento no PE da arquitetura reconfigurável utilizada neste trabalho. A decomposição atende a necessidade da arquitetura em grade, pois um PE pode ter no máximo quatro vizinhos.

O algoritmo de busca em profundidade irá percorrer o DFG uma única vez e à medida que for visitando os nodos os mesmos serão posicionados e roteados, na grade. A complexidade do algoritmo é O(A+V), onde V é o conjunto de vértice e A o conjunto de arestas. A principal vantagem desta solução é percorrer apenas uma vez em relação as outras duas versões dos algoritmos de P&R propostos nesta dissertação. No posicionamento dos nodos pode ocorrer que algumas arestas do grafo não possam ser roteadas localmente na arquitetura. Por exemplo, se os nodos n0 e n2 da Figura 4.2 forem posicionados nos PE1 e PE6 da arquitetura da Figura 4.1, não

será possível conectar n0 a n2. As arestas não roteadas serão resolvidas pela rede global.

O procedimento inicia com a identificação dos nodos raízes do DFG. No nosso exemplo são os nodos n0 e n1. Ao iniciar com o nodo n0, primeiro um PE livre foi obtido na grade. Supõe-se que seja o PE1. Para cada nodo raiz, a busca pelo primeiro PE livre iniciou-se na primeira linha da grade no sentido de oeste para leste. Caso atinja-se o último PE da linha e ainda não tiver encontrado um livre, a busca seria iniciada na próxima linha até o fim na grade. O algoritmo percorre o grafo em profundidade. Para alocar n2 é realizada uma busca em profundidade no PE1 do grafo da arquitetura, ou seja, na vizinhança do PE1 onde n0 foi alocado para obter um PE livre para n2. A busca retorna o primeiro PE disponível na grade iniciando pelo PE ao sul, leste, norte e oeste, neste caso o PE5. Esta abordagem regular foi feita para simplificar a busca em profundidade no CGRA, mas a ordem pode ser outra ou até mesmo aleatória.

Continuando a busca em profundidade, o nodo n2 tem dois sucessores, o algoritmo irá selecionar um deles para ser o primeiro sucessor e aplicar a busca em profundidade recursivamente. Não se pode determinar qual será o primeiro sucessor a ser percorrido, esta seleção depende da estrutura de dados do algoritmo. Nesta primeira versão do algoritmo o nodo n4 foi o primeiro sucessor escolhido. Não existe distinção ou preferência por nenhum sucessor. O algoritmo continua de n4 até n12, como ilustrada a Figura 4.4. O nodo n12 não possui sucessores e, portanto, a busca em profundidade retorna até o nodo n2 onde começa um novo percurso em profundidade em n9.

Importante observar que além de posicionar os nodos de n0, n2, n4, n9, n11 e n12 na arquitetura, as arestas deste percurso são roteadas ao mesmo tempo em que o algoritmo faz o posicionamento, ou seja, o algoritmo faz simultaneamente o P&R. Isso é possível pois o posicionamento foi feito em PE adjacentes, que tem uma conexão local. A Figura 4.4 mostra em destaque as arestas do DFG que já foram roteadas na arquitetura CGRA.

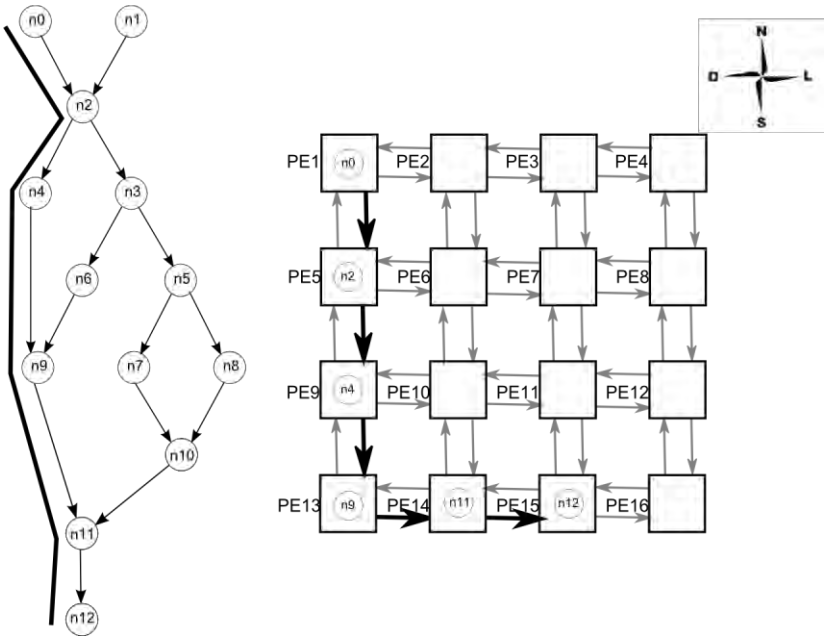


Figura 4.4. Primeira etapa P&R busca em profundidade

Para alocar o nodo n3 é realizada uma busca em profundidade na arquitetura, a partir do PE5, para obter um PE livre que neste caso foi o PE6, como ilustrado na figura 4.5. Como o nodo n3 tem dois sucessores, qualquer um dos dois pode ser escolhido. Suponha que n6 seja escolhido, a busca em profundidade continua até n9, porém, o nodo n9 já havia sido visitado e assim, o algoritmo irá retornar. Como o nodo n9 já foi posicionado e roteado para conectar diretamente com o nodo n4, não é possível garantir que o nodo n6 seja vizinho de n9 na arquitetura, pois n9 foi posicionado em função de n4. Porém antes de retornar para n6 é verificado se n6 e n9 foram ocasionalmente posicionados em PE vizinhos, caso verdadeiro, serão roteados localmente, senão, como ilustrado na Figura 4.5, a aresta é armazenada em uma lista de conexões não roteadas, que será resolvida pelo roteamento global.

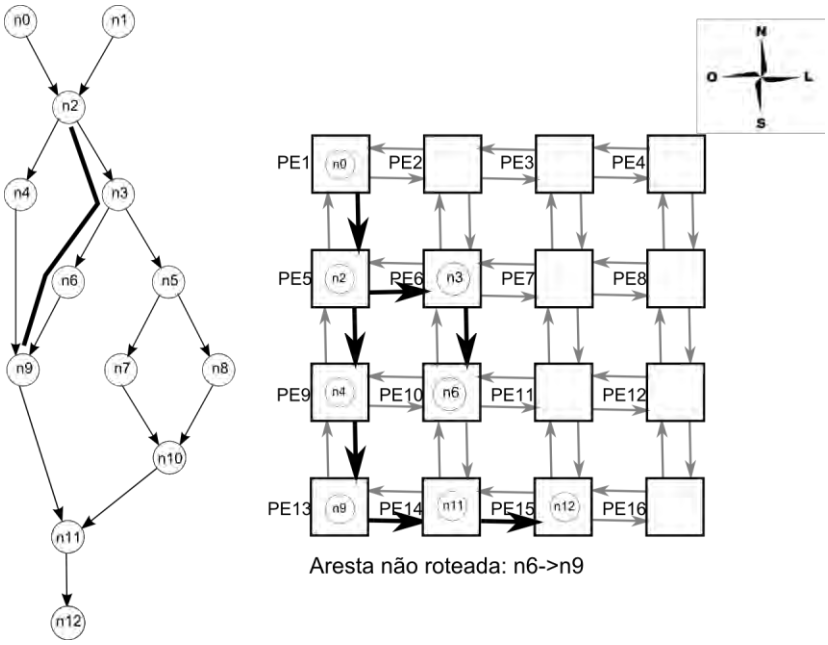


Figura 4.5. Segunda etapa P&R busca em profundidade

O algoritmo continua a busca em profundidade no segundo sucessor de n3, que é o nodo n5 até o nodo n11, que também já foi visitado e não pôde ser roteado nesta fase. A aresta n10→n11 é adicionada a uma lista de arestas não roteadas. Finalmente, o algoritmo retorna à n5, e caminha em direção a n8. O sucessor de n8, que é o nodo n10, já foi posicionado, porém diferente dos casos anteriores, n8 é vizinho de n10, e pode ser conectado localmente.

Resta ainda, percorrer o nodo raiz n1 e seus sucessores. Primeiro é feita a busca pelo próximo PE livre, na arquitetura, para n1 que é o PE2. Como todos os sucessores de n1 já foram alocados, o algoritmo apenas verifica a possibilidade de resolver localmente n1 à n2. Como não é possível, adiciona a aresta à lista de roteamento global. A Figura 4.6 apresenta o resultado da primeira etapa do P&R. A ordem em que as arestas foram percorridas é representada por um índice ao lado da seta na grade.

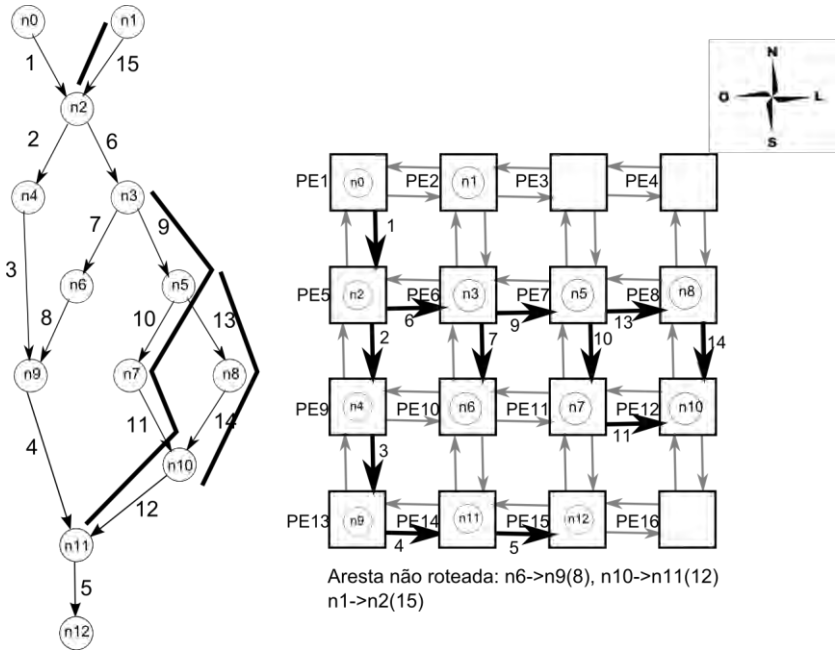


Figura 4.6. Etapa final P&R busca em profundidade

4.2.2 Busca em profundidade priorizando nodos no caminho crítico

Nesta seção, a busca em profundidade prioriza o posicionamento dos nodos pertencentes ao caminho crítico (CP) do DFG. Para os nodos com dois sucessores a seleção do primeiro nodo a ser posicionado na grade será do nodo que pertence ao CP. Nesta solução é necessário percorrer o grafo DFG três vezes. Duas vezes para marcar o CP no DFG e a terceira vez para executar o posicionamento e roteamento local na arquitetura CGRA, priorizando os nodos nos caminhos críticos. Esta solução pode ter um custo três vezes maior em tempo de execução que o algoritmo de busca em profundidade descrito na seção 4.2.1 .

A definição do CP foi feita utilizando os algoritmos de escalonamento ASAP (*as soon as possible*) e ALAP (*as late as possible*). Portanto foi necessário visitar todos os nodos para marcar o ASAP e mais uma vez para marcar o ALAP. No ASAP cada operação, nodo DFG, é executada o mais breve possível obedecendo suas dependências enquanto o ALAP executa as operações o mais tarde possível. A partir da diferença do ASAP e ALAP de cada nodo obteve-se o CP. O caminho crítico é o caminho mais longo de execução do grafo, consideramos que cada aresta tem tempo constante de execução. Veja o CP marcado no DFG na Figura 4.7.

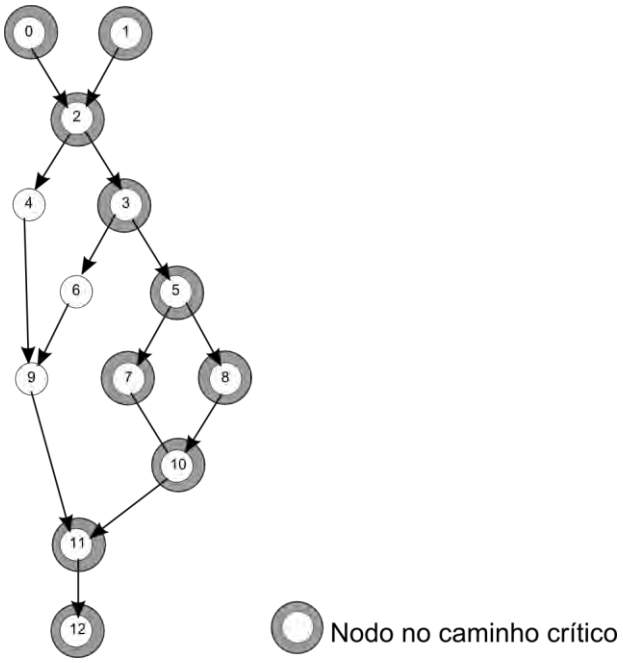


Figura 4.7. DFG nodos do caminho crítico marcados

O CP pode ser formado por mais de um caminho dentro do grafo DFG. No exemplo, mostrado na Figura 4.7, um CP pode ser iniciado pelos nodos n0 ou n1 seguindo por n2, n3 ao chegar em n5 tem mais duas opções de caminho por n7 ou n8 que seguem por n10, n11 e por fim atinge o nodo folha n12. No final tem-se quatro caminhos que representam o CP do grafo.

O procedimento de busca em profundidade priorizando o CP inicia com a identificação dos nodos raízes que foram n0 e n1. Como ambos os nodos raízes estão no CP logo obteve-se a mesma prioridade de posicionamento na arquitetura em grade. Supondo o primeiro nodo raiz selecionado seja n0, posicionado no PE1. O nodo tem um único sucessor, o nodo n2, que também faz parte do CP, é posicionado em PE5. O nodo n2 tem dois sucessores n3 e n4, como apenas n3 faz parte do CP será selecionado. A busca em profundidade continua até o nodo folha n12 e retorna ao nodo n5. A primeira etapa do P&R pôde ser acompanhada pela Figura 4.8.

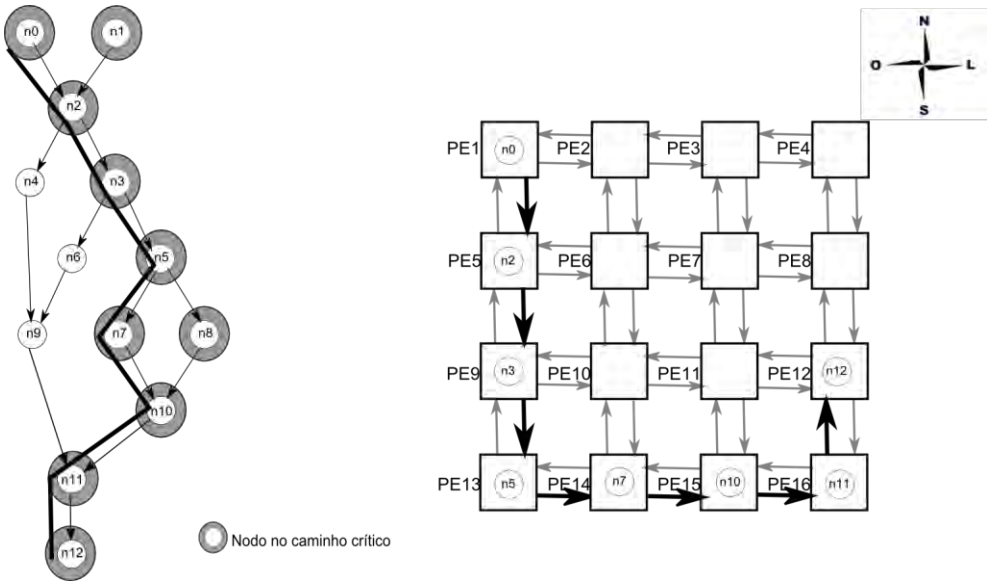


Figura 4.8. Primeira etapa P&R busca em profundidade priorizando o caminho crítico

Para alocar n8, não existe um PE livre adjacente ao PE13, onde o predecessor de n8, o nodo n5 está posicionado. O algoritmo escolhe um PE livre no sentido de leste para oeste na mesma linha de PE13, caso não encontre a busca inicia na próxima linha da arquitetura ciclicamente até encontrar um PE livre na linha. A última linha a ser percorrida será a anterior ao PE13. A aresta n5→n8 é adicionada a lista de arestas não roteadas.

O nodo n8 tem apenas um sucessor n10 no grafo, já visitado, a aresta também não é roteada localmente. A busca em profundidade retorna para n3 e percorre n3→n6. O resultado desta etapa do P&R pode ser acompanhado na Figura 4.9. Duas arestas não foram roteadas localmente n5→n8 e n8→n10.

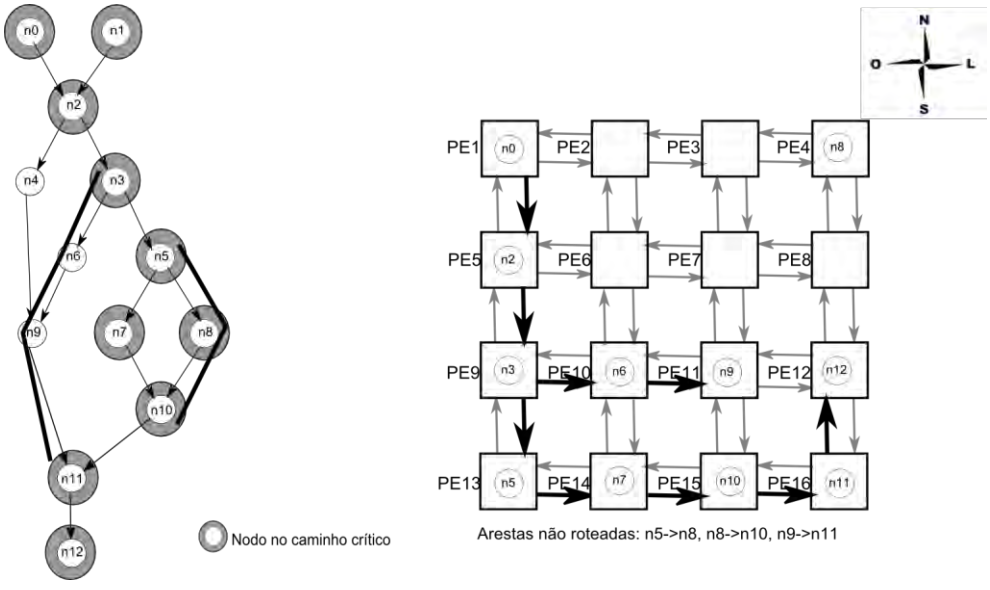


Figura 4.9. Segunda etapa P&R busca em profundidade priorizando caminho crítico

O n6 é alocado no PE10. A busca em profundidade continua até n11. O algoritmo retorna para n2 e percorre n2→n4 até n9. Nesta última parte do algoritmo foram visitados os nodos que não fazem parte do caminho crítico. E nessa etapa tem-se duas arestas que não foram roteadas localmente: n9→n11 e n4→n9.

O nodo n1 é o último elemento do CP que ainda não foi percorrido. Para posicioná-lo é obtido o próximo PE livre para nodo raiz que neste caso é o PE1. O único sucessor de n1 é o nodo n2 que já foi posicionado, verifica-se a possibilidade de n1 e n2 estarem posicionados em PE vizinhos. O resultado do algoritmo dessa seção pode ser observado na Figura 4.10.

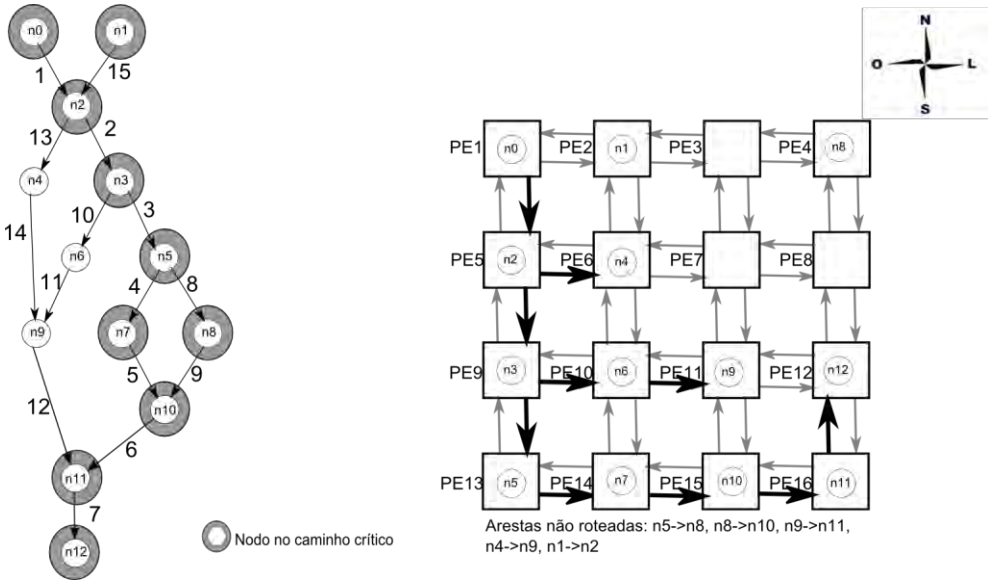


Figura 4.10. Etapa final P&R busca em profundidade priorizando o caminho crítico

Observou-se assim que foi necessário percorrer o grafo DFG três vezes para concluir o P&R desta seção, uma para o ASAP, outra para ALAP e o último para o posicionamento. O algoritmo da seção anterior só percorre uma vez. Além disso, 5 arestas não foram roteadas localmente.

4.2.3 Busca em profundidade priorizando apenas nodos no caminho crítico

A busca em profundidade apenas dos nodos do caminho crítico diferencia do algoritmo de busca que prioriza o conjunto de nodos que pertencem ao CP. O algoritmo proposto nesta seção irá percorrer o DFG quatro vezes. Duas vezes são necessárias para marcar os nodos do CP execução do ASAP e ALAP. Uma vez para realizar o P&R somente dos nodos do CP. E por último faz o P&R dos nodos que não fazem parte do CP.

Temos dois nodos raízes no caminho crítico n0 e n1. Suponha que n0 foi o primeiro selecionado e posicionado em PE1. A busca em profundidade passa pelos nodos do CP até atingir o nodo folha n12, como ilustrado na figura 4.11, nesta primeira execução não houve incidência de arestas não roteadas na grade.

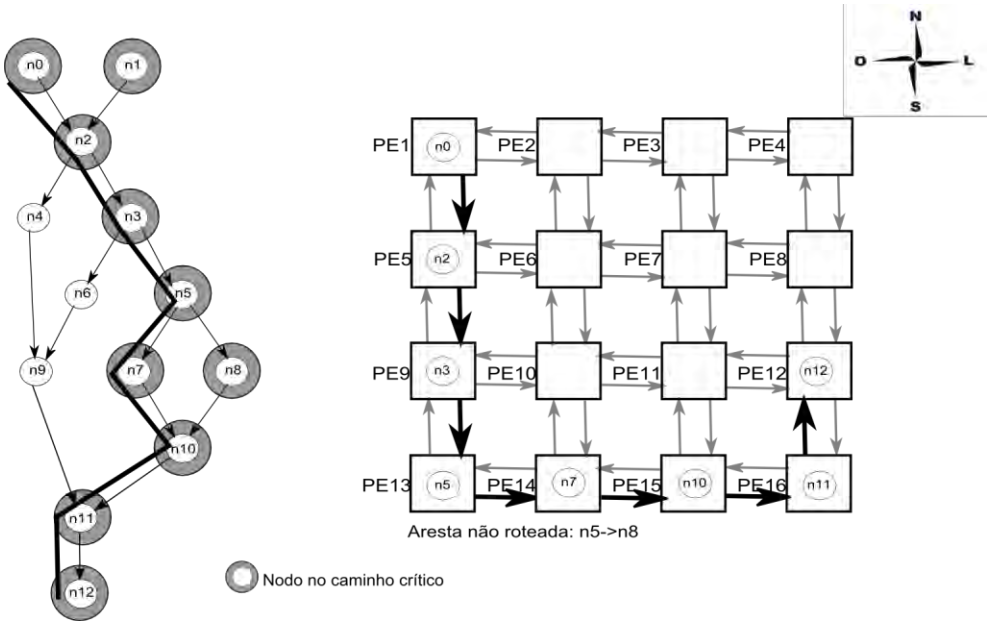


Figura 4.11. Primeiro passo P&R somente nodos no caminho crítico

O segundo passo começa após atingir o nodo folha, a busca em profundidade retorna para o nodo n5 e percorre os sucessores no CP não visitado. Segue pela aresta n5→n8, mas n5 não possui vizinho livre. O nodo n8 é alocado em PE11 e como n10 já foi visitado a busca em profundidade retorna para n3. O nodo n3 não possui sucessores no CP não visitados, a busca retorna para o nodo n2 que também não possui sucessores no CP não visitado. A busca em profundidade recomeça no próximo nodo raiz do CP, n1. Segue a busca pela aresta n1→n2, como n2 foi visitado é verificado se n2 possui vizinho livre para alocar n1 e retorna PE6. O nodo n1 é alocado no PE6. Assim, os nodos que pertencem ao CP foram visitados como ilustra a Figura 4.12. Com apenas uma aresta não roteada na grade.

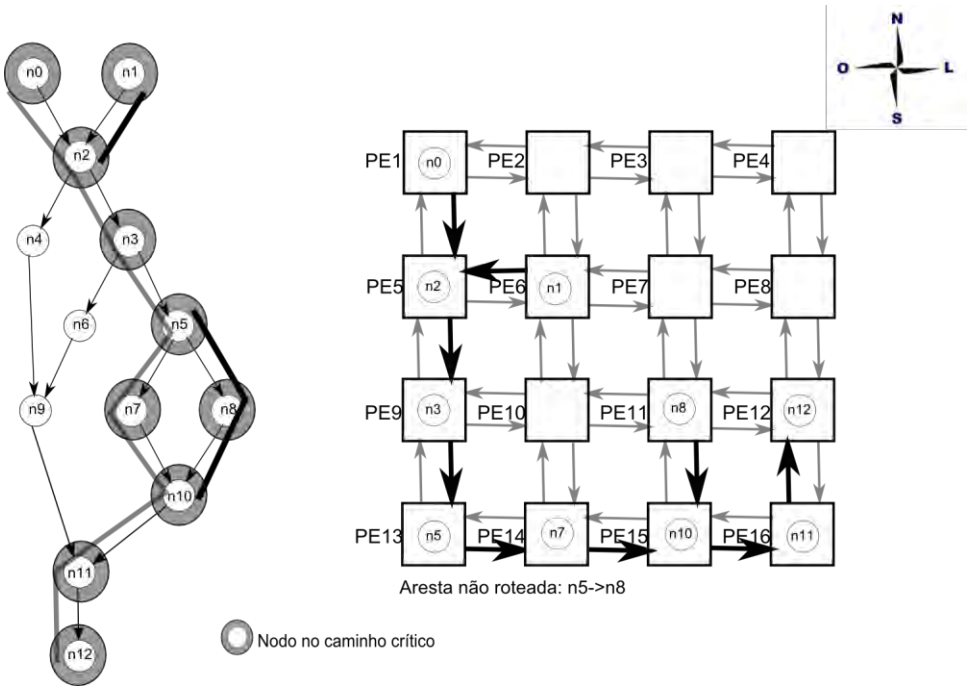


Figura 4.12. Segundo passo P&R somente nodos no caminho crítico

A segunda etapa é fazer o P&R dos nodos que não pertencem ao CP. É feita uma nova busca em profundidade a partir das raízes mapeando os nodos não visitados, pois todos os nodos do CP foram visitados. Considere que a busca em profundidade começa pelo nodo raiz n1 que busca recursivamente por sucessores não visitados. Nesta busca o primeiro nodo não visitados é n4, sucessor do nodo n2. Como n2 não possui vizinho livre então a aresta n2→n4 é adicionada a lista de arestas não roteadas. O nodo n4 é posicionado na primeira posição livre na direção de leste para oeste. A busca em profundidade continua pelo nodo n4 e n9. Outro nodo não visitado é n6, sucessor do nodo n3. A procura por vizinho livre de n3 para alocar n6 retorna PE10. O nodo n6 é alocado em PE10 e a busca continua em n6 onde seu sucessor n9 já foi visitado e não faz parte da sua vizinhança, então a aresta n6→n9 é adicionada a lista de arestas não roteadas. A conclusão desta etapa só ocorre após percorrer todo o DFG. A figura 4.13 ilustra o mapeamento na grade desta etapa.

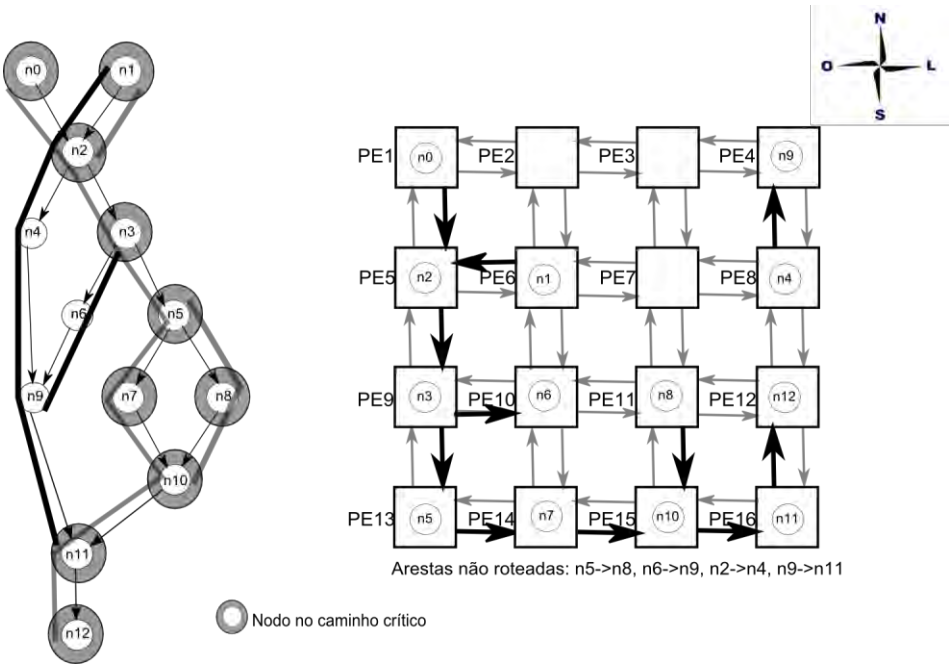


Figura 4.13. Etapa final P&R busca em profundidade somente caminho crítico

O algoritmo de busca em profundidade priorizando somente o caminho crítico, apresentado nesta seção, pode ser executado percorrendo apenas três vezes o DFG.

Para a marcação do CP o DFG é percorrido duas vezes, sendo uma para o ASAP e outra para o ALAP. Na terceira execução, o DFG é percorrido recursivamente a partir de sua(s) raíz(es) mapeando na CGRA somente os nodos do CP e os nodos que não compõem o CP são colocados em uma estrutura de pilha. Os nodos mapeados são marcados como visitado. Após visitar todos os nodos do CP, cada elemento da pilha é retirado e o nodo e seus sucessores que ainda não foram visitados, são percorridos em profundidade até que todos sejam visitados. Considere ABP-F2 a nova implementação do algoritmo ABP-F.

4.2.4 Roteamento Global

A solução empregada para resolver a lista de arestas não roteadas na arquitetura foi o acoplamento de redes multiestágios (MIN) à arquitetura. A MIN realiza o roteamento global, ou seja, qualquer par de PE na arquitetura pode ser conectado, desde que não ocorra conflito na MIN. Cada PE da arquitetura está ligado a um terminal de entrada e de saída da MIN. A arquitetura tem disponível até duas redes MINs. Para aumentar a capacidade de roteamento das MIN são adicionados estágios extras (k). Para simplificar o nosso exemplo usaremos uma rede Omega com

16 terminais sem estágios extras. A rede Omega com os terminais associados aos PEs da CGRA é apresentado na Figura 4.14.

Vamos apresentar o roteamento na MIN e para isso será utilizado a lista de arestas não roteadas gerada pela busca em profundidade descrita na seção 4.2.1 . A lista é composta pelas seguintes arestas n6→n9, n10→n11 e n1→n2. Os nodos que compõem as arestas n1, n2, n6, n9, n10 e n11 foram alocados em PE2, PE5, PE10, PE13, PE12 e PE14 respectivamente. Teremos que conectar então PE10→PE13, PE12→PE14 e PE2→PE5

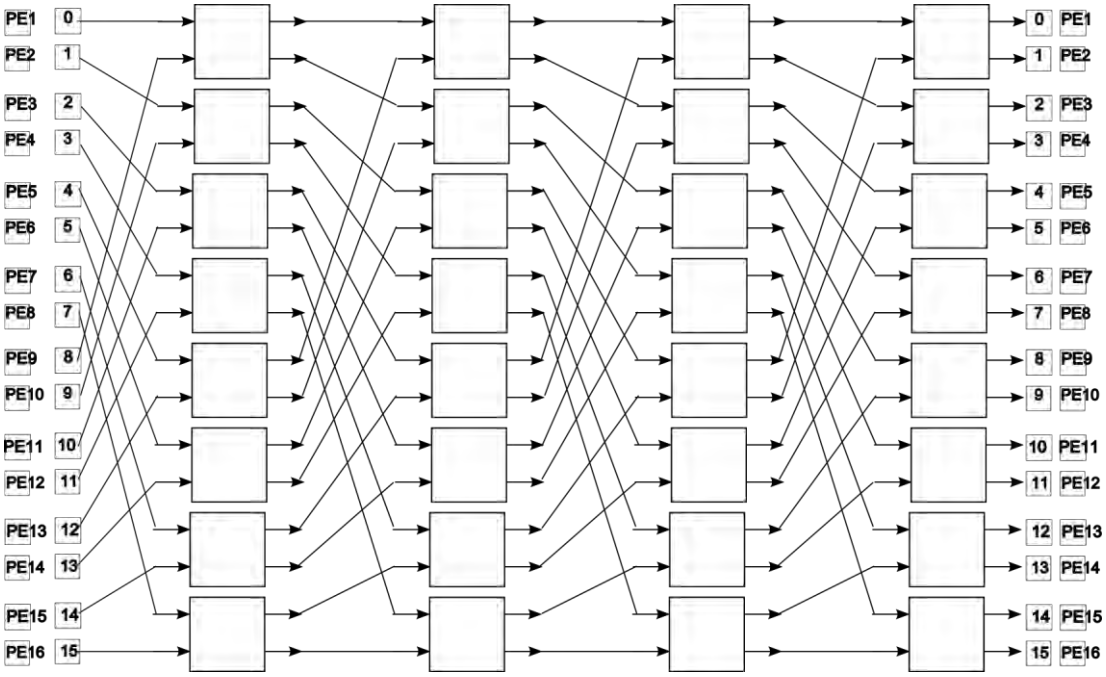


Figura 4.14. Rede Omega MIN 16x16

Como o PE X está conectado ao terminal X-1, teremos os seguintes terminais da MIN para rotear: 9→12, 11→13 e 1→4. O primeiro passo é criar uma palavra binária (W) através da concatenação da representação binária do terminal de entrada e saída da MIN. Suponha a conexão 9→12, a entrada 9 (1001) e saída 12 (1100), a palavra W será 10011100. O caminho de roteamento é único e formado a partir da palavra W. A saída de cada estágio (Z), a linha do estágio, é obtida com o deslocamento de uma janela de $\log_2 N$ bits sobre a palavra W. Para o primeiro estágio, a janela é posicionada na palavra, da esquerda para direita com o deslocamento de um bit, que corresponde ao endereço em negrito na palavra W=1**0011**100. Para o próximo estágio, a janela se desloca um bit para a direita gerando 10**0111**00, depois no estágio seguinte será 100**1110**0 e finalmente o endereço de destino no último estágio 1001**1100**. Cada valor de Z representa um

endereço em cada estágio intermediário. A palavra de controle (CW) irá definir como será a configuração interna da chave comutadora em cada estágio, a ligação interna da chave pode ser direta (0) ou cruzada (1). CW é obtido pela operação XOR entre os endereços de entrada e saída da MIN. Neste exemplo, a palavra de controle é 0101. O resultado dessa conexão pode ser visto na Figura 4.15.

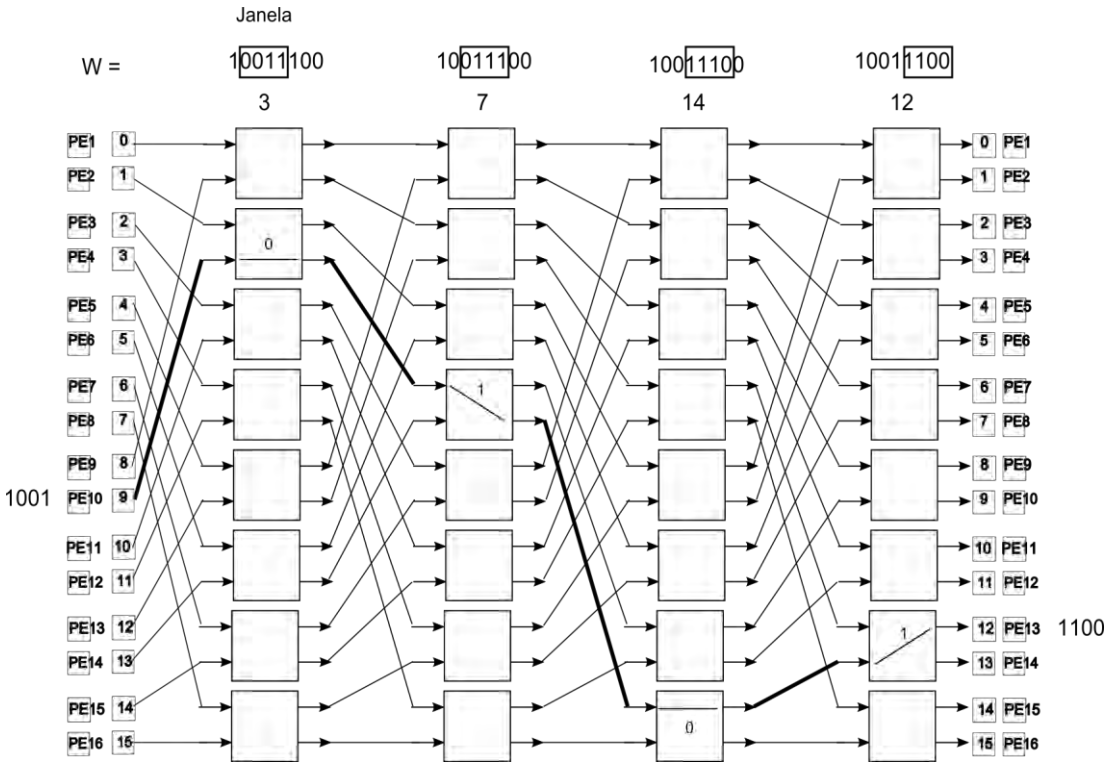


Figura 4.15. Conexão 9→12 na rede Omega 16x16

Vamos prosseguir com o roteamento das arestas 1→4 e 11→13. Vamos conectar na MIN a aresta 1→4. Temos então a palavra W=00010100, os endereços intermediários são 0010, 0101, 1010 e 0100 e CW=0101.

E para finalizar a lista de arestas falta ainda 11→13. Temos W=10111101, os endereços dos estágios intermediário são 0111, 1111, 1110 e 1101 e CW= 0110. Observe que o endereço do terceiro estágio, 1110, da MIN é igual na aresta 11→13 e 9→12, isso significa que ocorreu uma colisão na conexão da MIN, no terceiro estágio. Na Figura 4.16 podemos visualizar o conflito entre os pares de conexões na MIN.

Uma forma de resolver este conflito é aumentar a capacidade de roteamento. A adição de um estágio extra na rede Omega MIN dobra as opções para rotear as entradas e saídas. Com o estágio extra, o caminho é determinado pela representação binária da entrada seguida do estágio extra e por último do endereço da saída. Vamos

retomar a lista de PE que não foram roteados na arquitetura 9→12, 11→13 e 1→4. Suponha uma rede Omega 16x16 com um estágio extra. A concatenação W terá um bit a mais, representado por X, que simboliza o estágio extra, teremos W = 1001X1100. Para a conexão 9→12, podemos assumir X=0 ou X=1. Vamos considerar X=0 então W= 1001**0**1100 e temos o caminho, onde negrito simboliza a janela e o sublinhado o bit X, formado pela sequência 1**001**0**1**100, 10**01**0**1**100, 100**10**1**1**00, 1001**0**1**10**0 e 1001**0**1**100**. A palavra de controle para programar os switches será o XOR entre as palavras de entrada e saída, onde o bit extra entra no final da palavra de entrada e no início da palavra de saída, para o nosso exemplo de 9→12, temos CW=11110, ou seja apenas a última chave é programada diretamente e as outras são cruzadas. O resultado está na Figura 4.17. Posteriormente, temos a conexão 1→4, considerando X=0, W=000100100. E por último, 11→13 se considerarmos X=0, temos W=101101101, ocorrerá um conflito no terceiro estágio, na linha 11. Mudando para X=1, a conexão 11→13 agora terá W=101111101, não ocorre conflito e a conexão é roteada. A rede Omega MIN 16x16 com estágio extra pode ser vista na Figura 4.17.

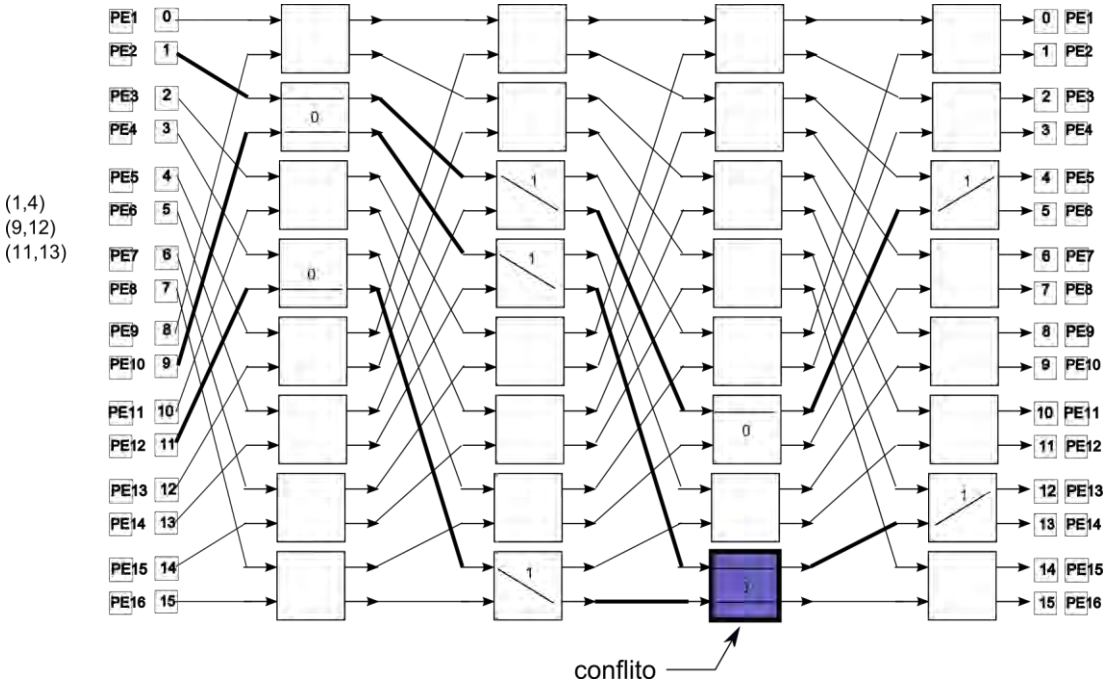


Figura 4.16. Conflito na MIN conexões 9→12 e 11→13

Suponha M estágios extras, teremos 2^M opções para cada ligação. O algoritmo percorre a lista de arestas não conectadas pelo roteamento local. Para cada aresta, é avaliado o roteamento na MIN para todas opções de 2^M, começando com X=0 até $X=2^M-1$. A primeira que satisfaz é roteada e a rede MIN é programada para esta

conexão. Se só existe uma rede MIN, o roteamento falha quando nenhum valor de X resolve os conflitos. No caso de duas redes MIN, quando a aresta não é roteada na primeira rede, a segunda rede é testada. A lista de aresta é percorrida apenas uma vez, e nenhuma conexão já programada na MIN é desfeita, ou seja, não usamos um algoritmo rearranjável, usando um algoritmo simples, priorizando a execução direta.

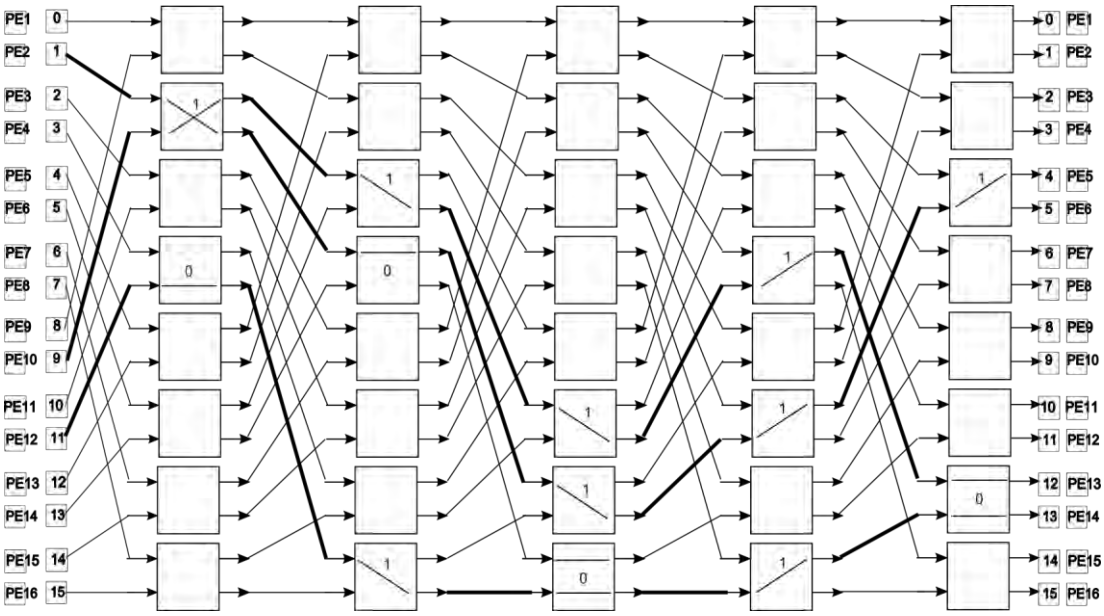


Figura 4.17. MIN N=16 com estágio extra, sem conflito

.

5 RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos neste trabalho. Inicialmente, apresentamos a capacidade de roteamento de uma forma isolada de uma rede Omega .

Na seção seguinte, a MIN, juntamente com a arquitetura, serão avaliadas em termos de capacidade de realizar o roteamento completo. O tempo de execução também será avaliado. Finalmente, na seção 5.3, os algoritmos com prioridades para caminho crítico serão analisados.

5.1 Roteamento na rede Omega

A capacidade de roteamento das redes MIN com estágio extra é um problema complexo (GAZIT; MALEK, 1989) e adotaremos uma solução por amostragem. O número de permutações de uma rede MIN com N entradas/saídas é dado pelo fatorial do número de entradas/saídas, N!. O número de permutações de entrada de uma rede com N=256 é 10^{506}, o que inviabiliza a avaliação de todo o espaço de solução. A técnica adotada foi a geração de uma amostragem aleatória, para avaliar a capacidade de roteamento da rede Omega MIN. Três amostragens com 1 milhão, com 100 milhões e com 1 bilhão de combinações de conexões foram avaliadas nas redes com N entradas e k estágios extras. Cada conjunto de conexões foi formado por N pares (entrada/saída). O resultado das execuções de 1 milhão, 100 milhões e 1 bilhão não apresentaram grandes diferenças. A Figura 5.1, mostra uma permutação aleatória. As conexões geradas foram aplicadas a várias configurações da rede com N variando de 16 à 256 e com K estágios extras variando de 1 à 4. Foi avaliada a capacidade com uma e com duas redes Omega. Nas configurações com duas redes Omega, a conexão foi verificada na primeira rede e caso não seja roteada será testada na segunda rede.

Entrada	Saída	Par (Entrada,Saída)
5	4	(5, 4)
12	2	(12, 2)
13	0	(13, 0)
7	8	(7, 8)
6	7	(6, 7)
11	15	(11, 15)
0	14	(0, 14)
15	9	(15, 9)
3	12	(3, 12)
1	5	(1, 5)
14	13	(14, 13)
9	10	(9, 10)
2	3	(2, 3)
8	11	(8, 11)
10	6	(10, 6)
4	1	(4, 1)

Figura 5.1. Pares de conexões aleatórios para rede com N=16.

A partir do conjunto de conexões, foram definidos percentuais de utilização, através da limitação do número de terminais que serão ligados na rede MIN. O nível de utilização foi definido em percentuais de 100%, 75%, 50% e 25% das ligações aplicadas. Para uma rede Omega MIN com N = 16, o conjunto de conexões foi formado por 16 pares de ligações para 100% de utilização, para 75% de utilização temos 12 pares, para 50% de utilização temos 8 e para 25% de utilização temos 4 pares.

Em cada execução gerada, obteve-se um percentual de conexões bem semelhantes, a maior diferença das execuções foi de 0,01%. As Tabelas 5.1, 5.2 e 5.3 apresentam o percentual de conexões que foram realizadas com sucesso, ou seja, não houve conflito nas ligações dos N terminais na rede Omega, a área sombreada destaca os percentuais superiores a 90%. Na primeira coluna da esquerda tem-se as configurações das redes que podem ter uma ou duas redes com estágios extras K = 0, 1, 2 e 4. Na primeira linha tem-se o número de terminais da rede N = 16, 32, 64, 128 e 256 e na segunda linha com a legenda "Redes", o nível de utilização (100%, 75%, 50% e 25%) da rede, na primeira coluna abaixo da legenda “Redes” tem-se a configuração da rede.

Tabela 5.1. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=16 e 32

N	16,				32,			
Redes	100%	75%	50%	25%	100%	75%	50%	25%
Omega	0,02%	0,20%	7,27%	60,43%	0,00%	0,00%	0,05%	19,89%
Omega + k=1	0,37%	3,00%	38,98%	94,94%	0,00%	0,00%	2,61%	71,63%
Omega + k=2	2,68%	15,77%	77,00%	99,81%	0,00%	0,05%	22,93%	96,10%
Omega + k=4	22,23%	64,96%	99,07%	100,00%	0,04%	6,84%	87,11%	99,97%
2 Omegas	50,24%	77,09%	94,25%	99,65%	2,18%	21,12%	66,69%	96,52%
2 Omegas + k=1	98,47%	99,88%	100,00%	100,00%	51,59%	89,37%	99,20%	99,99%
2 Omegas + k=2	99,97%	100,00%	100,00%	100,00%	96,93%	99,87%	100,00%	100,00%
2 Omegas + k=4	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Tabela 5.2. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=64 e 128

N	64,				128,			
Redes	100%	75%	50%	25%	100%	75%	50%	25%
Omega	0,00%	0,00%	0,00%	1,18%	0,00%	0,00%	0,00%	0,00%
Omega + k=1	0,00%	0,00%	0,00%	28,62%	0,00%	0,00%	0,00%	2,22%
Omega + k=2	0,00%	0,00%	0,42%	79,72%	0,00%	0,00%	0,00%	41,83%
Omega + k=4	0,00%	0,00%	41,36%	99,55%	0,00%	0,00%	2,43%	97,33%
2 Omegas	0,00%	0,27%	19,08%	84,32%	0,00%	0,00%	0,45%	54,46%
2 Omegas + k=1	1,28%	38,41%	91,48%	99,86%	0,00%	1,02%	59,53%	99,02%
2 Omegas + k=2	53,58%	96,08%	99,95%	100,00%	0,90%	65,20%	99,33%	100,00%
2 Omegas + k=4	99,96%	100,00%	100,00%	100,00%	97,70%	99,97%	100,00%	100,00%

Tabela 5.3. Permutação de conexões que foram roteados nas configurações da rede Omega MIN, N=256

N	256,			
Redes	100%	75%	50%	25%
Omega	0,00%	0,00%	0,00%	0,00%
Omega + k=1	0,00%	0,00%	0,00%	0,00%
Omega + k=2	0,00%	0,00%	0,00%	6,09%
Omega + k=4	0,00%	0,00%	0,00%	89,56%
2 Omegas	0,00%	0,00%	0,00%	15,64%
2 Omegas + k=1	0,00%	0,00%	11,00%	95,42%
2 Omegas + k=2	0,00%	6,92%	94,67%	99,99%
2 Omegas + k=4	67,74%	99,35%	100,00%	100,00%

O aumento de terminais (N) na rede Omega MIN aumenta muito a possibilidade de conflito de roteamento. Na Figura 5.2, pode-se observar que a capacidade de roteamento da rede Omega MIN aumentou com o número de estágios extras (K) e com a diminuição do percentual de conexão. A utilização de duas redes Omega MIN com estágios extras (1, 2 e 4) aumentou muito a capacidade de roteamento.

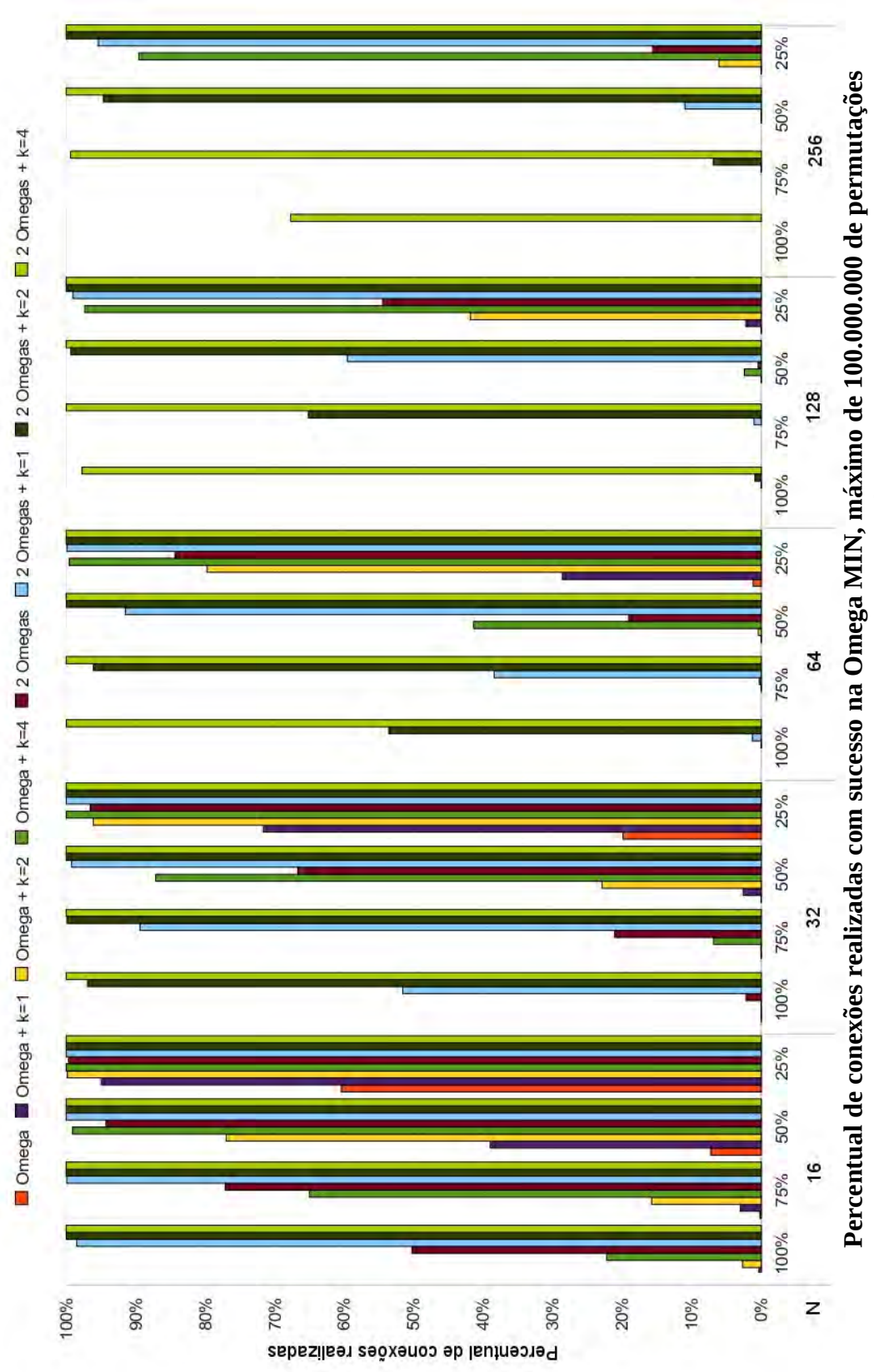


Figura 5.2. Percentual de conexões realizadas com sucesso na Omega MIN, máximo de 100.000.000 de permutações considerando N entradas e N saídas

Como veremos na próxima seção, as conexões que não foram roteadas localmente na arquitetura proposta estão abaixo de 50%. Considerando que o percentual de utilização da rede MIN seja no máximo 50% (metade dos terminais) notou-se que duas redes Omega MIN com dois estágios extras são suficientes mesmo para redes com 256 terminais.

5.2 Capacidade de roteamento da arquitetura e tempo de execução P&R

Para avaliar a estrutura da arquitetura proposta foram utilizados dois conjuntos de *benchmark*, num total de 26 grafos de fluxos de dados (DFGs). O primeiro grupo foi usado em (FERREIRA; et al., 2007) e compreende: dct, Cplx8, filterRGB, DCT line, fir64, Fir16, filterRGBpaeth e SNN. O segundo conjunto foi composto por 18 DFGs extraído do *ExPRESS Group* (*Electrical&Computer Engineering Department at the UCSB*, 2008) que é um repositório selecionado entre mais de 1400 DFGs obtido de aplicações do *MediaBench benchmark suite*. Estes algoritmos exploram o paralelismo, de operações entre matrizes, processamento de sinais digitais e compressões de imagens. A complexidade dos DFGs varia de 11 à 359 nodos e de 8 à 380 arestas.

A Tabela 5.4 mostra o número de arestas não roteadas para o algoritmo de busca em profundidade (ABP) proposto. Devido a baixa conectividade da arquitetura em grade e a simplicidade do algoritmo de mapeamento não foi possível rotear todas as arestas do grafo DFG. O percentual de aresta não roteadas na arquitetura em grade foi de 33% (ver a quarta/quinta coluna da Tabela 5.4). Quando uma rede Omega MIN foi adicionada a CGRA em grade 2D o percentual caiu para 11% (coluna 6). Considerando que a MIN use estágios extras, o número de arestas não roteadas diminui para 4% para dois estágios extras (k=2) e para 2% para quatro estágios extras (k=4), vide (colunas 7 e 8). Para o caso em que a rede Omega MIN usa quatro estágios extras (k=4) quase 40% dos *benchmarks* avaliados foram mapeados com sucesso.

O congestionamento na MIN pode ser causado por duas situações. A primeira é devido a conflito interno da MIN. Este conflito pode ser resolvido adicionando estágios extras na MIN para aumentar a capacidade de roteamento. A segunda situação ocorre quando duas arestas não roteadas na grade precisam conectar-se a um

mesmo nodo receptor ou emissor (extremidades da aresta). A adição de outra rede MIN resolve esse problema.

A configuração mínima para rotear todos os *benchmarks* possui duas redes MIN com dois estágios extras (K=2). Quando não se usou estágios extras (K=0) menos de 2% do total de arestas não foi roteado, conforme representado nas colunas 9, 10 e 11 da Tabela 5.4.

Tabela 5.4. P&R dos algoritmos na arquitetura em grade com MIN

Benchmark	Nodos	Arestas	Número de arestas não roteadas							
			Grade	% Grade	Grade+Uma MIN			Grade + Duas MINs		
					K=0	K=2	K=4	K=0	K=2	K=4
dct	139	186	68	36,56	32	10	8	6	0	0
Cplx8	46	60	19	31,67	9	4	3	1	0	0
filterRGB	57	70	19	27,14	3	2	0	0	0	0
DCT	92	124	41	33,06	15	4	2	1	0	0
fir64	193	255	92	36,08	32	10	6	11	0	0
Fir16	49	63	30	47,62	6	0	0	2	0	0
filterRGBpaeth	84	106	31	29,25	8	5	0	1	0	0
snn	253	299	77	25,75	31	8	1	6	0	0
fir1	44	43	20	46,51	7	1	0	1	0	0
arf	28	30	10	33,33	2	0	0	0	0	0
jpeg_idct_ifast	170	210	74	35,24	30	15	6	7	0	0
smooth_color_z	197	196	64	32,65	24	7	4	2	0	0
collapse_pyr	72	89	28	31,46	11	5	0	0	0	0
horner_bezier_surf	18	16	3	18,75	0	0	0	0	0	0
jpeg_fdct_islow	175	210	71	33,81	32	8	0	6	0	0
matmul	117	124	33	26,61	15	5	0	4	0	0
fir2	40	39	15	38,46	4	2	0	1	0	0
motion_vectors	32	29	11	37,93	3	0	0	0	0	0
cosine1	66	76	29	38,16	7	2	2	0	0	0
idctcol	186	236	84	35,59	31	15	6	4	0	0
interpolate_aux	108	104	41	39,42	12	3	0	1	0	0
feedback_points	54	51	17	33,33	4	1	0	0	0	0
ewf	42	55	18	32,73	3	1	0	0	0	0
write_bmp_header	111	93	23	24,73	5	1	1	0	0	0
invert_matrix_gen	359	380	106	27,89	36	8	1	5	0	0
hal	11	8	1	12,50	0	0	0	0	0	0
Média	106	121	39,4		13,9	4,5	1,5	2,3	0,0	0,0
Total	2743	3152	1025		362	117	40	59	0	0
% médio de arestas não roteadas				32,5	11,5	3,7	1,3	1,9	0,0	0,0

Os tempos de execução do P&R proposto nesta dissertação são, apresentados na Tabela 5.5. O tempo de execução obtido para o algoritmo de P&R mostra que a nova estrutura proposta neste trabalho pode ser utilizada em ambientes de compilação JIT.

Tabela 5.5. Tempo de execução do P&R local e na MIN

Benchmark	Nodos	Arestas	CPU time (ms)		
			Grade + Duas MINs, K=2		
			Posicionamento + roteamento local	Roteamento MINs	total
dct	139	186	6,40	1,30	7,70
Cplx8	46	60	1,30	0,10	1,40
filterRGB	57	70	1,65	0,10	1,75
DCT	92	124	5,40	0,50	5,90
fir64	193	255	4,15	0,75	4,90
Fir16	49	63	1,20	0,05	1,25
filterRGBpaeth	84	106	1,75	0,55	2,30
snn	253	299	4,90	3,90	8,80
fir1	44	43	1,65	0,25	1,90
arf	28	30	0,70	0,15	0,85
jpeg_idct_ifast	170	210	3,00	0,55	3,55
smooth_color_z	197	196	9,05	0,60	9,65
collapse_pyr	72	89	1,55	0,35	1,90
horner_bezier_surf	18	16	0,30	0,10	0,40
jpeg_fdct_islow	175	210	4,30	1,05	5,35
matmul	117	124	1,85	0,30	2,15
fir2	40	39	0,65	0,10	0,75
motion_vectors	32	29	2,25	0,10	2,35
cosine1	66	76	1,25	0,40	1,65
idctcol	186	236	2,75	1,00	3,75
interpolate_aux	108	104	2,45	0,40	2,85
feedback_points	54	51	0,50	0,20	0,70
ewf	42	55	0,60	0,10	0,70
write_bmp_header	111	93	1,30	1,50	2,80
invert_matrix_gen	359	380	9,00	1,45	10,45
hal	11	8	0,05	0,05	0,10
Média	106	121	2,80	0,63	3,43
Total	2743	3152			

Um experimento foi realizado para avaliar o tempo de execução do P&R. Como a ferramenta VPR é uma referência para P&R, foram comparados os tempos

de execução ABP e o VPR (BETZ; ROSE; MARQUARDT, 1999; VPR and T-VPack 5.0.2, 2009), aplicados aos *benchmarks* apresentados. O ABP executa na CGRA com duas rede Omegas MINs e o VPR executa em CGRA estilo-ilha, veja Figura 2.2. Apesar de serem duas arquiteturas diferentes o custo em recursos de roteamento é semelhante. Os tempos do VPR foram medidos para a opção mais rápida (*-fast -timing analysis off*). A comparação é apresentada na Figura 5.3, que mostra a aceleração do ABP em relação o VPR.

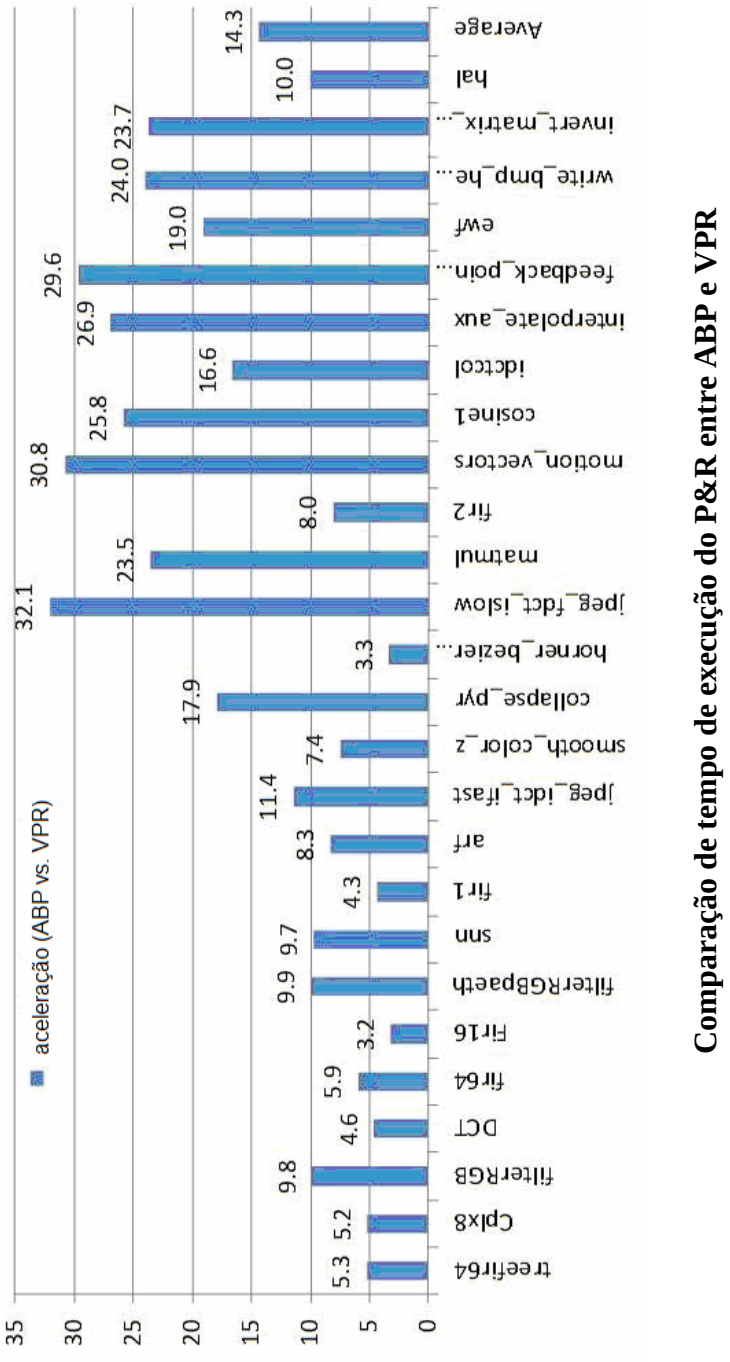


Figura 5.3. Comparação de tempo de execução do P&R entre ABP e VPR

Em média o VPR foi 14,3x mais lento. O maior ganho de aceleração do tempo de execução foi com jpeg_fdct_islow (32,1x) e o menor foi com fir1(3,2x). O P&R são operações demoradas e o algoritmo ABP consegue aumentar em uma ordem de grandeza. O VPR é uma ferramenta de referência na área de P&R e o ganho, de acelerar em mais de 10x em relação ao VPR, assim viabiliza a implementação em compiladores JIT.

Na Tabela 5.6, apresenta o tempo total, mapeamento CGRA e roteamento na MIN, das váriações dos algoritmos ABP, ABP-P, ABP-F e por último ABP-F2, versão modificada do ABP-F. A melhor média de 100 execuções foi do algoritmo de busca em profundidade (ABP) com 3,558ms e a maior média de execução foi o ABP-F2 que levou 16ms a mais na média de execução do ABP. Apesar de reduzir o número de vezes que percorre o DFG, o ganho no tempo do P&R é pequeno, variando de 0ms à 8,66ms

Tabela 5.6. Tempo de mapeamento no CGRA ABP, ABP-P, ABP-F e ABP-F2.

Benchmarks	**tempo CPU (ms)**			
	Grade + Duas MINs, K=2			
	ABP	**ABP-P**	**ABP-F**	**ABP-F2**
Cplx8	2,180	1,970	2,160	2,210
filterRGB	13,620	13,870	15,290	14,820
DCT	2,960	3,630	3,260	4,270
fir64	18,570	18,920	14,890	23,550
Fir16	2,480	2,230	2,580	2,830
filterRGBpaeth	5,900	5,310	9,770	5,770
snn	10,170	11,600	9,000	11,250
fir1	0,940	1,130	1,120	1,900
arf	0,750	0,530	0,660	0,550
jpeg_idct_ifast	2,290	4,040	2,830	3,100
smooth_color_z_triangle	2,470	3,000	3,190	3,530
collapse_pyr	2,140	2,850	1,990	2,090
horner_bezier_surf	0,360	0,220	0,340	0,320
jpeg_fdct_islow	2,180	2,800	2,960	3,180
matmul	1,260	1,450	1,440	2,140
fir2	1,040	1,330	1,060	1,110
motion_vectors	0,400	0,440	0,500	0,500
cosine1	2,100	2,080	1,980	2,740

Benchmarks	tempo CPU (ms)			
	Grade + Duas MINs, K=2			
	ABP	ABP-P	ABP-F	ABP-F2
idctcol	3,000	3,180	3,450	3,480
interpolate_aux	2,130	2,600	2,880	2,460
feedback_points	1,430	1,380	1,230	1,330
ewf	0,580	2,430	0,660	0,690
write_bmp_header	1,990	2,100	1,780	2,210
invert_matrix_general	7,840	8,360	8,510	9,290
hal	0,160	0,400	0,460	0,210
Média	3,558	3,914	3,760	4,221
Total	88,940	97,850	93,990	105,53

5.3 Caminho Crítico

Para estudar a influência das conexões locais e globais na latência de execução do DFG mapeado na arquitetura, foi definida uma relação entre o peso do atraso de uma conexão global e local. O objetivo é analisar os algoritmos de posicionamento tem como critério a latência final da implementação. Usaremos o formato PE:MIN, onde os valores representados são o atraso local do PE e da ligação global na MIN.

Foi analisado o aumento no caminho crítico (CP) dos *benchmarks* após o posicionamento e roteamento. Foram considerados diferentes atrasos PE:MIN (1:1 e 1:2). O aumento da latência do caminho crítico será comparado com a taxa de atraso de 1:0, que é equivalente para obtenção dos caminhos críticos para o DFG quando assumimos atraso 1 para cada operação e desconsideramos os atrasos de interconexão, ou seja uma rede ideal.

Supondo que o grafo representado possuía atraso de 0 (zero) nas interconexões locais e cada nodo, elemento de processamento, possui atraso local de 1 (uma) unidade. Usando a relação 1:0, a latência do caminho crítico será 4 e passa pelos nodos A, D, E e C, conforme Figura 5.4.

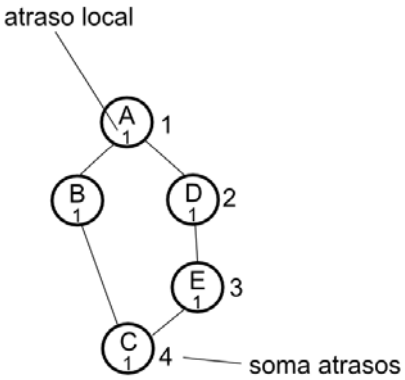


Figura 5.4. DFG com atraso do caminho crítico de 4 unidades

Para a configuração 1:1, o atraso da conexão global é de 1 unidade. Supondo que a aresta E → C do caminho crítico seja mapeada global com atraso de 1 unidade, irá aumentar a latência para 5 unidades. Se o atraso global for 2, a latência será, como ilustrado na Figura 5.5.

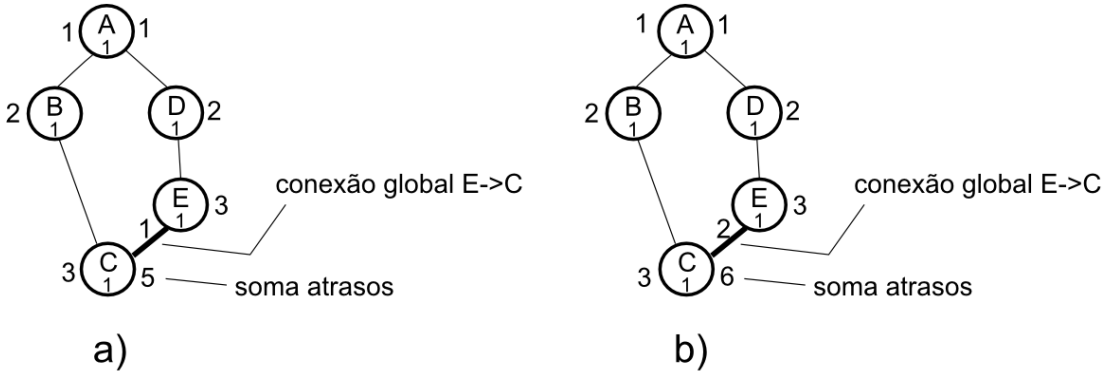


Figura 5.5. DFG com conexão global a) atraso de 1 unidade b) atraso de 2 unidades

Uma aresta global também pode estar fora do caminho crítico, por exemplo, B → C. A conexão global foi utilizada em local fora do caminho crítico e não houve problema. Se o atraso é 1:1, o caminho A, B e C que tinha latência 3 passa para latência 4, mas não houve aumento na latência total. Este caminho que não era crítico tornou-se crítico após o mapeamento.

O algoritmo que busca primeiro o mapeamento do caminho crítico pode ajudar aplicações onde o importante é minimizar a latência total. Entretanto se o atraso for 1:2, como ilustrado na Figura 5.6 o atraso passa de 3 para 5 degradando a solução. Caso o atraso 1:2 ocorra em uma aresta do caminho crítico, passa a latência para 6. A busca em profundidade normal pode ter latência 5, e o algoritmo que prioriza o caminho crítico obteve latência 4, 20% a menos na relação 1:2.

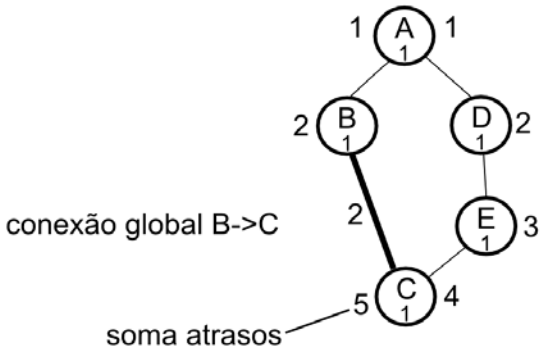


Figura 5.6. DFG com atraso total de 4 unidades

A latência do caminho crítico foi medida comparando os três algoritmos de P&R apresentados nesta dissertação. O primeiro Algoritmo de busca em profundidade (ABP) que não leva em conta o caminho crítico (CP) do DFG. O segundo algoritmo faz a busca em profundidade, priorizando (ABP-P) o posicionamento do nodos pertencentes ao CP. E por último, o algoritmo que faz a busca em profundidade posicionando primeiro os nodos do CP (ABP-F) e depois o restante.

Tabela 5.6. Percentual de latência nos algoritmos de posicionamento

(PE:MIN)	(1:1)			(1:2)		
	Min.	Max.	Média	Min.	Max.	Média
ABP	10%	71%	**31%**	29%	114%	**66%**
ABP-P	2%	57%	**27%**	3%	143%	**59%**
ABP-F	1%	44%	**17%**	3%	114%	**47%**

Os resultados apresentados para os três algoritmos de P&R. Para a taxa de 1:1 (mesma latência para o PE e MIN) teve-se aumento médio na latência do CP de 31% (máximo 71% e mínimo 10%), 27% (máximo 57% e mínimo de 2%) e 17% (máximo de 44% e mínimo de 1%), para ABP, ABP-P e ABP-F, respectivamente. Quando a taxa é de 1:2 (MIN com dobro de latência do PE), o aumento médio da latência do CP é 66% (máximo de 114% e mínimo de 29%), 59% (máximo 143% e mínimo de 3%) e 47% (máximo de 114% e mínimo de 3%) para ABP, ABP-P e ABP-F, respectivamente. Notou-se que a redução na latência do CP que foi atingida pelos ABP-P e ABP-F sobre ABP onde a média foi de 2,9 e 10,9% para 1:1, respectivamente, e 4,2 e 11,4% para 1:2.

Foi realizado uma estimativa do comportamento dos algoritmos de P&R que priorizam o caminho crítico (ABP-P e ABP-F), observou-se menor latência no CGRA que o algoritmo sem critério de seleção dos nodos (ABP). Em relação ao

ABP-F e ABP-P, o primeiro tem menor percentual de latência, pois tem maior priorização do caminho crítico.

Estimou-se os pesos para arestas mapeadas no CGRA e na MIN, como 1 e 2 respectivamente. É necessário acoplar esta ferramenta a um compilador para extrair das aplicações os elementos que serão executados na arquitetura proposta. Assim será possível executar e avaliar o tempo de execução das aplicações. Durante a execução, um trecho que era o caminho crítico pode deixar de ser. Por exemplo um operador de fluxo condicional "*if-else*" que no CP estimado passa pelo "if" que possui um número maior de elementos de fluxo em relação ao "else", mas durante a execução passará pelo "*else*".

6 CONCLUSÕES E TRABALHOS FUTUROS

O objetivo deste trabalho foi avaliar uma nova proposta de posicionamento e roteamento para arquiteturas de grão grosso, visando atender demandas de compilação dinâmica e *just-in-time*. Para tal, apresentou-se uma arquitetura em grade 2D, sem recurso de roteamento nos PE, que ocupa uma área 30% menor que a arquitetura proposta por (FERREIRA et al., 2007) com recurso de roteamento. A idéia foi simplificar os recursos locais de roteamento para reduzir o custo do PE, entretanto, algumas conexões necessitam de um mecanismo global. A solução foi dividida em duas etapas. Na primeira, realizou-se o posicionamento e roteamento local usando três algoritmos de busca em profundidade. O primeiro algoritmo foi proposto em (FERREIRA et al., 2007) e outros dois foram desenvolvidos para avaliar o impacto do caminho crítico no mapeamento. A segunda etapa utiliza uma rede multiestágio para realizar o roteamento global, diferente de propostas anteriores que usam barramentos, roteamento pelos PEs, um padrão mais rico de conexões locais ou algoritmos mais complexos e lentos para P&R.

Com relação à primeira etapa de posicionamento e roteamento local, o primeiro algoritmo, ABP, percorre o DFG uma única vez e faz o posicionamento e roteamento local. Os outros algoritmos priorizam o caminho crítico, percorrendo mais de uma vez o DFG para executar o mapeamento. Ao estimar que o peso das arestas ligadas na MIN fosse o dobro do peso da aresta na CGRA, o algoritmo que percorre o caminho crítico teve menor latência média, ou seja, para os casos em que o atraso de execução da MIN for maior que na CGRA a latência de execução será menor, como apresentado na Tabela 5.6. A latência pode ser influência pela natureza da aplicação de entrada, pois o grafo da aplicação pode conter operadores condicionais, *"if-else"*, que gera um falso caminho crítico, onde nem sempre o maior caminho estará ativo durante a execução. É necessário o acoplamento de ferramentas para processar as aplicações na arquitetura e analisar seu comportamento. Uma ferramenta de compilação é necessária para fazer os testes de latência dos circuitos mapeados, executando os algoritmos propostos.

Para resolver as conexões do DFG que não foram ligadas localmente na arquitetura, foram acopladas até duas redes globais multiestágio. Para aumentar a conectividade da rede global foram adicionados k estágios extras que aumentam em 2^k, a conexão entre os terminais de entrada/saída. Outro ponto importante é que no lugar de desenvolver uma nova arquitetura de grão grosso, que teria que ser fabricada, a proposta desta dissertação é baseada na utilização de um FPGA com base para implementação física, adicionando um nível acima para reconfiguração.

O uso de estágios extras na rede Omega, aumenta a capacidade de roteamento, porém, não resolve todas as permutações completas mesmo para redes modestas em tamanho. Contudo, o contexto de utilização deste trabalho é diferente de outros trabalhos de multiestágios. Aqui foi mostrado que para uso parcial da MIN, onde o percentual de ligações é menor que 50% (Tabela 5.2), a rede MIN responde com sucesso à demanda de roteamento.

Com a arquitetura proposta, obteve-se a simplificação e diminuição do tempo de execuções do posicionamento e roteamento dos elementos de processamento. O tempo gasto para o mapeamento na arquitetura de grão grosso foi da ordem de milissegundos enquanto o mapeamento por *Modulo Schedule* ou rDPA é da ordem de segundos (MEI et al., 2003). A estimativa da proposta de P&R com o ABP é promissora para ambientes JITs onde o tempo de execução foi da ordem de milissegundos. Obteve aceleração média de 14,3x comparado com a ferramenta VPR que é uma ferramenta referência, apresentada na Figura 5.3. A utilização em grade regular da arquitetura, CGRA, permite boa ocupação de área reservada, a simplicidade de conexão dos PE, conecta apenas ao vizinhos, reduz a densidade de fios na arquitetura.

Os resultados apresentados com a adição da rede multiestágio global foram promissores, pois o ganho de desempenho superou a perda em área, ocasionado por essa rede. Nossa proposta pode ser utilizada em sistemas de computação reconfigurável que necessitam de posicionamento e roteamento em tempo de execução, como por exemplo a execução de compilação dinâmica.

As principais contribuições deste trabalho foram de usar um algoritmo de mapeamento polinomial para grafos DFG em arquitetura de grão grosso e de mesclar algoritmo de posicionamento polinomial com o roteamento em multiestágio.

Uma proposta de trabalho futuro é fazer o acoplamento da arquitetura em um compilador e executar os algoritmos de *benchmarks* para fazer medições da latência com carga de trabalho.

Estimativas do atraso foram feitas considerando os pesos 1 e 2 para a arestas na CGRA e MIN, respectivamente. Para uma avaliação real é necessário configurar um FPGA escolher um algoritmo de filtro de imagem por exemplo, configurá-lo na arquitetura, ao executar várias imagens para cada um dos mapeamentos propostos medindo o tempo final dos grafos onde alguns terão operadores condicionais e outros não.

Na Figura 6.1, temos um DFG com um elemento condicional (*if-else*), nodo A, o maior caminho é pelo nodo B. Se o nodo A for verdadeiro vai passar por B, C e D, caso contrário irá passar pelo nodo E. O CP estimado passa por B, mas durante a execução os dados que for usar o algoritmo irão cair muito mais no "*else*" passando pelo nodo E, assim o posicionamento do caminho crítico não é vantajoso.

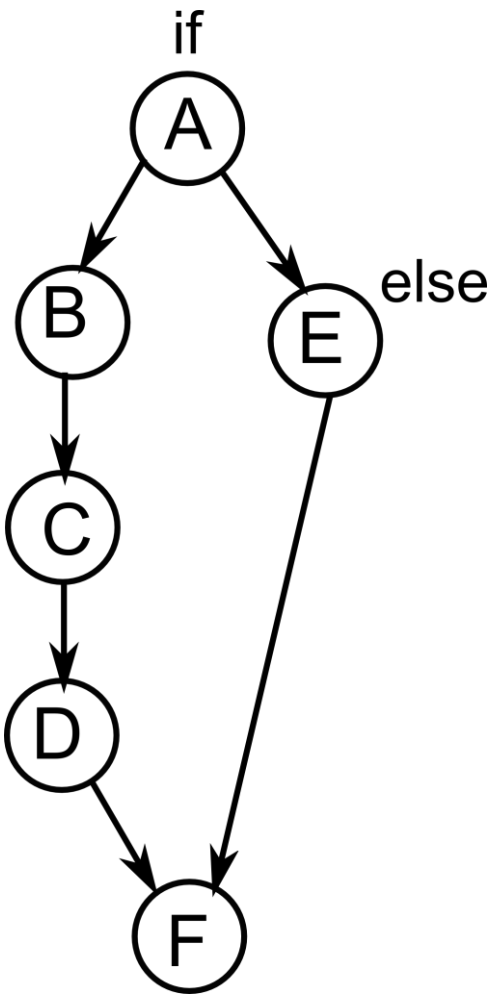


Figura 6.1. DFG “if-else”

REFERÊNCIAS BIBLIOGRÁFICAS

BENES, V. E. **General Stochastic Process in the Theory of Queues.** Addison-Wesley, 1962a.

BENES, V. E. **On rearrangeable three-stage connecting networks.** Bell System Technnical, J. 41, p. 1481-1492, 1962b.

BETZ, V.; ROSE, J. **VPR: A New Packing, Placement and Routing Tool for FPGA Research.** International Workshop on Field Programmable Logic and Applications, London, UK, p. 213-222, 1997.

BETZ, V.; ROSE, J.; MARQUARDT, A. **Architecture and CAD for Deep-Submicron FPGAs.** Kluwer Academic Publishers, 1999.

CHANG, M. L. **Device Architecture.** In: HAUCK, S.; DEHON, A. **Reconfigurable computing: the theory and practice of FPGA-based computation.** United States: Elsevier, 2008

CLOS, C. **A study of non-blocking switching networks**. Bell System Technical Journal 32 (5): 406–424, 1953.

DEHON, A. **Compact, multilayer layout for butterfly fat-tree.** Proceedings of the 12th annual ACM symposium on Parallel algorithms and architectures, Bar Harbor, Maine, EUA, p. 205-215, 2000.

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

EBELING, C.; MCMURCHIE, L.; HAUCK, S. A.; BURNS, S. **Placement and Routing Tools for the Triptych FPGA.** IEEE Transactions on VLSI Systems, vol. 3, n 4, p 473-482, December, 1995.

Electrical&Computer Engineering Department at the UCSB, U. **ExPRESS Benchmarks**, http://express.ece.ucsb.edu/benchmark/. Acesso em: 3 de nov. 2008.

FERREIRA, R. S. **Relatório Técnico em Redação.** Departamento de Informática – DPI, UFV, 2010.

FERREIRA, R. S.; GARCIA, A.; TEIXEIRA, T.; CARDOSO, J. M. P. **A Polynomial Placement Algorithm for Data Driven Coarse-Grained Reconfigurable Architectures**. in IEEE Computer Society Annual Symposium on VLSI (ISVLSI'07), 2007.

GAZIT I.; MALEK M. **On the Number of Permutations Performable by Estra-Stage Multistage Interconnection Networks**. IEEE Transactions on Computers VOL.38 NO. 2, 1989.

HARTENSTEIN, R.; JACOBI, R. P.; RINCON, M. A.; LLAMOS, C. H. **Using Rewriting-Logic Notation for Funcional Verification in Data-Stream Based Reconfigurable Computing.** In: FDL 2003, Frankfurt. Forum on Specification and Design Languages - FDL 03, 2003. Frankfurt, Germany, p. 1-10, 2003.

HARTENSTEIN, R. **Coarse Grain Reconfigurable Architecture (embedded tutorial).** Asia and South Pacific Design Automation Conference (ASP-DAC'01), Yokohama Japan, p. 564-570, 2001a.

HARTENSTEIN, R. A **Decade of Reconfigurable Computing: A Visionary Retrospective.** Conference on Design Automation and Test in Europe (DATE), 2001b.

HARTENSTEIN, R. W.; KRESS, R. **A Datapath Synthesis System for the Reconfigurable Datapath Architecture.** Asisa and South Pacific Design Automation Conference, ASP-DAC'95, August, 1995.

HUANG, R.; WAWRZYNEK, J.; DEHON, A. **Stochastic, Spatial Routing for Hypergraphs, Trees and Meshes.** Proceedings of the International Symposium on Field-Programmable Gate Arrays FPGA, Monterey, California - USA, p. 78-87, February, 2003.

KIRKPATRICK, S.; GELATT, C. D. Jr.; VECCHI, M. P. **Optimization by Simulated Annealing.** Science, p. 671-680, 1983.

LAI, Y.-T.; LAI, H.-Y; YEH, C.-N. **Placement for the Reconfigurable Datapath Architecture.** Circuits and Systems – ISCAS2005, IEEE International Symposium on, vol. 2, p. 1875-1878, May, 2005.

LEISERSON, C. E. **Fat-trees: Universal networks for hardware efficient supercomputing**. IEEE Transactions on Computers, p. 892–901, Oct. 1985.

LYSECKY, R.; VAHID, F.; TAN, S. X.-D. **Dynamic FPGA Routing for Just-in-Time FPGA Compilation.** Proceedings of the 41st Annual Design Automation Conference, p. 954-959, 2004.

MCMURCHIE, L.; EBELING, C. **PathFinder: A Negotiation-Based Performance-Drive-Router for FPGAs.** Proc of International Symposium on Field Programmable Gate Arrays, Monterey, California, USA, 1995.

MEHTA, G.; STANDER, J.; BAZ, M.; HUNSAKER, B.; JONES, A. K. **Interconnect Customization for a Hardware Fabric.** ACM Transactions on Design Automation of Electronic Systems, vol. 14, n. 1, Article 11, January, 2009.

MEHTA, G.; SANTER, J.; BAZ, M.; HUNSAKER, B.; JONES, A. K. **Interconnect Customization for a Coarse-grained Reconfigurable Fabric.** IEEE Internatinal

Parallel and Distributed Processing Symposium, March, 2007.

MEI, B.; VERNALDE, S.; VERKEST, D.; MAN, H. D.; LAUWEREINS, R. **Exploiting Loop-Level Parallelism On Coarse-Grained Reconfigurable Architectures Using Modulo Scheduling.** Design, Automation and Test in Europe Conference and Exhibition, Munich, Germany, 2003.

MOREANO, N. B. **Algoritmos para Alocação de Recursos em Arquiteturas Reconfiguráveis.** 113f. Tese (Doutorado em Ciência da Computação) - Instituto de Computação, Universidade Estadual de Campinas, Campinas-SP, 2005.

MUTHUKRISHNAN, S.; PATERSON, M.; SAHINALP, S. C.; SUEL, T. **Compact Grid Layouts of Multi-Level Networks.** Proceedings of the thirty-first Annual ACM Symposium on Theory of Computing, Atlanta, Georgia, United States, p. 455-463, 1999.

NAIR, R. **A Simple Yet Effective Technique for Global Wiring,** IEEE Transactions on Computer-Aided Design, vol. CAD-6, no. 6, p. 165-172, March 1987.

PIMENTEL, B. L. **Array de Processadores com Grão Grosso e Fluxo de Dados.** (Relatório final CNPq), Departamento de Informática – DPI,UFV,ago./2006-jul./2007, 2007.

POON, K. K. W.; WILTON S. J. E.; YAN, A. **A Detailed Power Model for Field-Programmable Gate Arrays .** ACM Transactions on Design Automation of Electronic Systems, vol. 10, n. 2, p. 279-302, 2005.

OTERO, J. C. S.; WAGNER, F. R.; CARRO, L. **JAVARRAY – Um Array Reconfigurável Para Redução De Consumo De Energia Em Arquiteturas Java.** Instituto de Informática – Universidade Federal do Rio Grande do Sul – UFRGS, 2005.

SEDGEWICK, R. **Algorithms.** Addison Wesley, 1998.

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 Transactions on Computers, vol.49, n 5, 2000.

TANIGAWA, K.; ZUYAMA, T.; UCHIDA, T.; HIRONAKA, T. **Exploring compact design on high throughput coarse-grained reconfigurable architectures.** Field Programmable Logic and Applications – FPL 2008, p. 543-546, September, 2008.

TESSIER, R. G. **Fast Place and Route Approaches for FPGAs.** PhD Thesis, MIT Massachusetts Institute of Technology, 1999.

VENDRAMINI, J. C. G. **Ambiente para Co-Projeto de Hardware/Software em Plataformas de FPGAs com Aplicação em Robótica Móvel.** (Iniciação Científica), Departamento de Informática – DPI, UFV, ago., 2009.

XILINX. **Logic Design Tool.** Disponível em: http://www.xilinx.com/. Acessado em: 23 fev. 2010.

VORWERK, K.; KENNING, A.; GREENE, J.; CHEN, D.T. **Improving Annealing via Directed Moves.** Field Programmable Logic and Applications - FPL 2007, Amsterdam, Netherlands, p. 363-370, aug., 2007.

VPR and T-VPack 5.0.2: **Full CAD flow for Heterogeneous FPGAs.** March 24, 2009. Disponível em: http://www.eecg.utoronto.ca/vpr/. Acessado em: 25/Ago./2009.

YANG, S. **Logic Synthesis and Optimization Benchmarks**, Version 3.0. Technical Report, Microelectronics Center of North Carolina, 1991.

YEH, C.-H.; VARVARIGOS, E. A.; PARHAMI, B. **Efficient VLSI Layouts of Hypercubic Networks.** Proceeding of the The 7th Symposium on the Frontiers of Massibely Parallel Computation, p. 98, 1999.

ZIED, M.; HAYDER, M.; EMNA, A.; HABIB, M. **Efficient tree topology for FPGA interconnect network.** Proceedings of the 18th ACM Great Lakes symposium on VLSI. p. 321-326, Orlando, Florida, USA, 2008.