ALGORITMO DE POSICIONAMENTO PARA FPGA BASEADO EM TRAVESSIA DE GRAFOS

Positioning Algorithm for FPGA based-Crossing Graphs

Luciana Rocha Cardoso

Mestre em Ciência da Computação, FACIG - Manhuaçu/MG – luroca@sempre.facig.edu.br

Resumo: O *hardware* reconfigurável é uma solução intermediária entre *software* e *hardware*, oferece a flexibilidade do *software* e o desempenho do *hardware*. Nos anos oitenta, os FPGAs surgiram como circuitos reconfiguráveis comerciais e escaláveis. Atualmente são baseados em tecnologia SRAM e formados por um conjunto de blocos lógicos reconfiguráveis e um sistema de interconexão reconfigurável, que podem ser reconfigurados totalmente ou parcialmente em tempo de execução. Possuem muita flexibilidade e resolveram com sucesso vários problemas nas últimas décadas com aceleração de 2 a 3 ordens de grandeza. E com a evolução da tecnologia, a complexidade do FPGA aumentou significativamente passando de centenas de blocos lógicos/interconexões para a ordem de milhões, exigindo mais desempenho, e aumentando a complexidade para mapear as aplicações. O problema de posicionar uma computação nos blocos lógicos e interligá-los é NP-Completo, e é um dos grandes desafios atuais. Várias heurísticas já foram propostas. Este trabalho propõe uma nova extensão de um algoritmo polinomial de posicionamento baseado em travessia em grafos, e comparando os resultados com estado da arte dos algoritmos de posicionamento, visando uma solução viável em tempo de execução sem deteriorar o caminho crítico e sem aumentar a complexidade do roteamento ao observar os pontos de *fanout* alto do circuito.

Palavras-chave: FPGA. Posicionamento. Tempo de execução.

Área do Conhecimento: Arquitetura de Computadores e Otimização.

INTRODUÇÃO

A Lei de Moore (1968) continua válida mostrando que em pouco tempo a densidade dos circuitos eletrônicos dobra, trazendo desafios na exploração desses recursos, como otimizar energia, o desempenho, os custos, etc. Diversas arquiteturas vêm sendo utilizadas [ASSIS, 2010], tais como processadores de propósito gerais (GPPs – *General Purporse Processors*), plataformas de arquiteturas reconfiguráveis (AR) e circuitos integrados para aplicações específicas (ASIC - *Application Specific Integrated Circuits*) e mais recentes processadores baseados em unidade (GPU).

Em um extremo, o GPP é uma arquitetura projetada para executar uma grande variedade de aplicações computacionais, ou seja, possui flexibilidade. No outro extremo, temos as arquiteturas baseadas em ASIC, que é um circuito integrado (CI) para executar tarefas específicas, cuja aplicação é diretamente implementada no *hardware*, gerando uma solução eficiente, mas com a desvantagem da inflexibilidade, pois não permitem alterações do circuito. Para implementar modificações deve-se reprojetar e refabricar o circuito [HAUCK; DEHON, 2008]. Enquanto, as AR são uma solução intermediária, que pode ser mais eficiente que os GPPs e mais flexíveis que os ASICs, pelo fato de permitir que o *hardware* seja reprogramado adaptando-se a uma determinada aplicação. A reconfiguração pode também ser realizada em tempo de execução para explorar o paralelismo da aplicação em função dos dados de entrada e outros parâmetros em tempo de execução.

Diversas AR vêm sendo propostas e utilizadas, e também vários sistemas embarcados vem sendo projetados com as ARs (CGRA, FPGA, etc) [ASSIS, 2010; HAUCK; DEHON, 2008; SILVA, 2006]. Os FPGA (*Field-Programmable Gate Arrays*) são dispositivos semicondutores que podem ser programados e reprogramados após sua fabricação revolucionando a forma como o *hardware* digital tem sido projetado e construído nas últimas três décadas [H.; K.; P.; D.; M., 2012], tendo uma alta densidade e alto desempenho, oferecendo um circuito rápido e programável de baixo custo. Os FPGAs atuais são baseados na tecnologia SRAM, e podem ser reconfigurados totalmente ou parcialmente em tempo de execução [H.; K.; P.; D.; M., 2012], proporcionando ao FPGA um alto nível de flexibilidade. Entretanto, a flexibilidade aumenta a complexidade do mapeamento motivando o desenvolvimento de novas heurísticas.

O mapeamento é uma das operações mais custosas devido à complexidade dos FPGA, e, envolve dois passos: posicionamento e roteamento, e ambos têm alto custo de execução**,** devido à sua complexidade e do problema de mapeamento, existem poucas soluções com reconfiguração dinâmica em tempo de execução [GONZALEZ, 2006; LIMA, 2008; MOREANO, 2005; ROSE; BROWN, 1991; SHARMA , 2005; FERREIRA; GARCIA; TEIXEIRA; CARDOSO, 2007; H.; K.; P.; D.; M., 2012; K.; A.; S, 2011; M; R, 2004].

O problema de posicionar uma computação nos blocos lógicos e interligá-los é NP-Completo. Várias heurísticas foram propostas. O posicionamento tem um impacto significativo sobre o desempenho do roteamento. Se o posicionamento é ruim, serão necessários muitos recursos para efetuar o roteamento e um longo tempo de execução. Em alguns casos não existe um roteamento possível para todas as conexões. Portanto a qualidade dos algoritmos de posicionamento afeta diretamente a utilização da arquitetura FPGA.

Este trabalho explora uma nova heurística de posicionamento baseada em travessia em profundidade de grafos que foi proposta em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], onde o posicionamento de blocos lógicos é realizado em tempo de execução. O objetivo foi explorar outras travessias e estudar o comportamento da heurística. O FPGA é modelado por um grafo, similar ao modelo proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. O grafo tem uma representação implícita. A representação tem uma correspondência direta com a estrutura física do FPGA. O posicionamento é implementado por uma função de mapeamento entre dois grafos (do FPGA e do circuito a ser mapeado). A localidade dos vértices é explorada durante este processo. Diferente da proposta apresentada em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], este trabalho propõe a exploração da travessia em largura (BF) e/ou profundidade (DF) em ambos os grafos, modificando também o sentido, em que a travessia é executada. Além da localidade das ligações, este trabalho difere da abordagem de [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], propondo avaliar durante o posicionamento os blocos lógicos que estão interligados a vários pontos e que tem um alto grau de saída (*fanout* alto) ou que tem grau 2.

DESENVOLVIMENTO

Referencial Teórico

A reprogramação dos blocos lógicos e interconexões do FPGA é realizada em geral por memórias SRAM que fazem com que a funcionalidade de unidades lógicas do FPGA se adapte a cálculos específicos, estabelecendo interligações entre as unidades lógicas através de canais de roteamento pré-fabricados. Com isso, cada aplicação pode ser mapeada em um FPGA, gerando uma solução de alto desempenho sem perder em flexibilidade. Mas devido à alta complexidade e tempo envolvidos em gerar e reconfigurar um FPGA por completo, nas duas últimas décadas surgiram FPGA parcialmente reconfiguráveis. A reconfiguração parcial permite que apenas uma parte do circuito seja modificado, reduzindo o tempo para geração e carregamento dos bits de configuração (*bitstreams*), dando ao FPGA a capacidade de se adaptar a novos contextos em tempo de execução (ou *online*).

Apesar do posicionamento e roteamento serem problemas estudados a mais de três décadas, as pesquisas atuais relacionadas ao mapeamento e desenvolvimento de algoritmos eficientes [H.; K.; P.; D.; M., 20124; K. A. S., 2011; A.; V., 2011; M.; J., 2010; M.; G.; S., 2011; H.; A.; A.. J., 2010; S.; S., 2011; FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] vem mostrando a importância destes problemas para o desenvolvimento de ferramentas para FPGA. Como já mencionado, o posicionamento é importante, pois a sua ineficiência afeta diretamente o mapeamento e pode impossibilitar o roteamento do circuito.

Os FPGA permitem a prototipagem rápida, reduzindo o *time-to-market* no desenvolvimento de novos produtos, e são cada vez mais usados para programar qualquer função lógica como um circuito ASIC [SHARMA, 2011]. Seus componentes lógicos programáveis são fisicamente conectados em uma hierarquia reconfigurável para executar funções combinacionais complexas. Além disso, os FPGA tem suporte a conexões em cascata propiciando sua utilização para operações aritméticas de alta velocidade [ALTERA CORPORATION, 2011; ALTERA CORPORATION, 2012].

No cenário *online*, encontrar e gerenciar espaços vazios disponíveis permite um maior reuso do FPGA. Como se trata de situações dinâmicas para serem usadas em tempo de execução, as etapas de posicionamento e roteamento devem ser implementadas de forma eficiente. A abordagem [M.; M., 2003] apresenta uma solução para realizar a deslocação/realocação dinâmica de blocos lógicos e suas interligações sem interromper a execução, resolvendo o problema de fragmentação dos espaços livres no FPGA através de trocas rápidas de contexto. Um algoritmo de posicionamento baseado no algoritmo *staircase* foi proposto em

[M.; R., 2004]. Em [H.; K.; P.; D.; M., 2012] foi apresentado um novo *framework* de compilação JIT (*Just-In-Time*). O *bitstream* para um FPGA virtual é calculado de forma rápida. O posicionamento e o roteamento são gerados em tempo de execução, através de um posicionamento inicial permitindo a reutilização dos recursos de *hardware* no FPGA virtual que pode estar mapeado sobre diferentes FPGA reais. A vantagem da virtualização FPGA [M.; P.; R.; D.; K.; J., 2011] é que as etapas, de posicionamento e roteamento são independentes do *hardware* físico, beneficiando da capacidade da reconfiguração dinâmica em tempo de execução, permitindo também que FPGAs que não suportam esse recurso, ou seja, o FPGA virtual, possa ser usado por qualquer tipo de FPGA.

A maior parte dos algoritmos de posicionamento propostos na literatura são implementados em tempo de compilação ou síntese do circuito, isto é, em um cenário estático. O principal foco é o mapeamento do circuito como um todo. Com o aumento da complexidade dos projetos e da disponibilização de FPGA mais complexos, os desafios são o posicionamento de circuitos com mais de 100.000 blocos. Na maioria dos algoritmos, o tempo de execução não é a primeira prioridade e a execução pode demorar horas ou até mesmo dias. Recentemente, o uso da computação paralela e/ou o uso de GPU vem sendo proposto para reduzir o tempo de execução dos algoritmos de posicionamento.

O VPR é um *framework* acadêmico proposto pela Universidade de Toronto que permite a avaliação e teste de arquiteturas de FPGA e de algoritmos de posicionamento e roteamento. A maioria dos algoritmos usa com base para comparação à ferramenta VPR [BETZ; CAMPBELL; et al, 2008; BETZ , 2011]. Em [LUDWIN; BETZ, 2011] foi proposta uma abordagem paralela determinística para o algoritmo de posicionamento *simulated annealing* (SA) que é muito usada em várias soluções de posicionamento e avalia movimentos em paralelo. Entretanto, o tempo de execução pode ser elevado. Essa abordagem acelera de 2.8x a 3.7x a solução do VPR. Em [M.; J., 2011] é proposto um algoritmo DAST (*Dynamically Adaptive Stochastic Tunneling algorithm*) para evitar o problema de "congelamento" em uma abordagem SA usando um esquema de detecção de aprisionamento de mínimos locais com base em DFA e uma programação dinâmica adaptativa para o problema de posicionamento em FPGAs, mostrando uma redução de 18,3% no tempo de execução de alguns *benchmarks* em comparação com a ferramenta VPR.

Uma abordagem baseada no algoritmo Star+ foi proposta em [M.; G.; S., 2011] com o intuito de desenvolver uma solução escalável com um método analítico independente do tamanho da rede, que pode ser resolvido por um sistema de equações não-lineares. Os resultados mostraram uma redução do caminho crítico em 8-9% e uma aceleração no tempo de execução de 5x em comparação com o VPR no seu modo rápido. Como já mencionado, o tempo de posicionamento e roteamento para circuitos com mais de 100k LUT é elevado, podendo chegar a horas, ou até mesmo dias. E infelizmente, as abordagens de SA não são adequadas neste cenário. Abordagens de posicionamento de circuitos complexos foram propostas em [H.; A.; A.; J., 2010; S.; S., 2011].

Como o foco deste trabalho são algoritmos de posicionamento viáveis em tempo de execução onde os circuitos em geral são formados módulos que ocupam centenas de blocos ou poucos milhares, as abordagens anteriores não são adequadas. Recentemente, uma abordagem baseada em uma solução gulosa [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] gerou um ganho de aceleração de três ordens de grandeza em relação ao VPR, tornando-se sua utilização viável em tempo de execução.

METODOLOGIA

Este trabalho propõe explorar variações de uma heurística de posicionamento proposta em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. Ao longo dessa seção serão relatadas as semelhanças e diferenças dessa abordagem sobre esse trabalho exposto.

Mapeamento

A proposta é percorrer o grafo no outro sentido, diferente da abordagem proposta em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], partindo das entradas em direção as saídas. A Figura 1 ilustra um exemplo de um grafo representado por um grafo de LUT de k-entradas. As entradas externas (entradas e saídas) são representadas pelos vértices em cinza e as LUTs pelos vértices em branco. Neste exemplo k = 4 mas apenas o vértice 13 tem 4-entradas. Esta figura será usada para mostrar os percursos durante o mapeamento baseados na travessia em profundidade ou *Depth-Fisrt* (DF) e na travessia em largura ou *Breadth-First* (BF).

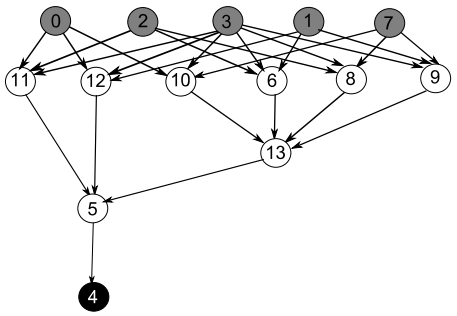


Figura 1: Exemplo grafo de LUT k-entrada, k máximo é 4.

As propostas apresentadas em [ASSIS, 2010; FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] utilizam o P&R (posicionamento e roteamento) baseado na travessia dos grafos. Este trabalho se limita apenas ao problema de posicionamento em FPGAs sem implementar o roteamento. O roteamento é feito com o VPR [BETZ; CAMPBELL et al 2008].

Mapeamento em DF

Durante a travessia em DF, os vértices descendentes são ordenados pelo maior caminho que possa alcançar as saídas, priorizando o caminho crítico durante a travessia, nessa travessia o algoritmo percorre aresta por aresta em profundidade. Suponha que na travessia o vértice 0 (Figura 1) foi o primeiro a ser selecionado. Este vértice possui três descendentes: 10, 11 e 12. Assim, o vértice 10 será o primeiro a ser visitado para priorizar o caminho crítico durante o mapeamento que seguirá a ordem de 0→10→13→5→4. No modelo apresentado na Figura 2, à abordagem de posicionamento mostra o problema de mapeamento gráfico onde um gráfico de entrada é mapeado sobre um gráfico de saída (o gráfico de saída é o FPGA), onde podemos observar que é dada prioridade a posições que possuem menor custo de posicionamento (próximas).

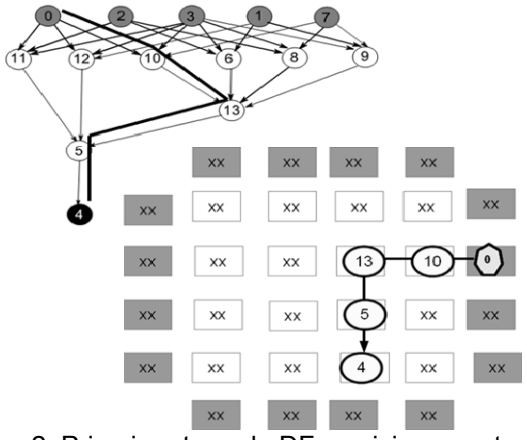


Figura 2: Primeira etapa de DF: posicionamento do vértice entrada 0.

Seguindo a ordenação em DF, como ilustrado na Figura 3, a próxima aresta é a aresta 0 → 11, onde o vértice destino 11 é posicionado próximo do vértice 0. Como a aresta de saída do vértice 11 leva ao vértice 5, que já foi visitado e posicionado, a próxima aresta é 0 → 12 que determina a posição do vértice 12. Como todas os descendentes da entrada 0 já foram visitadas, a busca em DF deve recomeçar por outra entrada não visitada.

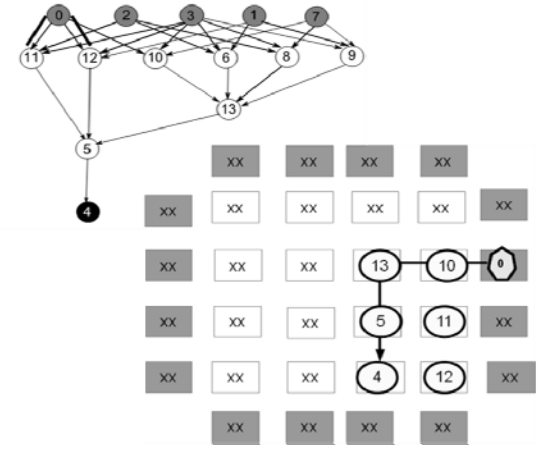


Figura 3: Segunda etapa de DF: posicionamento dos vértices 11 e 12.

E finalmente, após o posicionamento de todos os vértices e seus descendentes, é possível estimar o custo total do roteamento. A Figura 4 mostra o custo estimado em número de conexões para cada aresta. A soma dos custos das conexões usadas para mapear todas as arestas é 79, para este posicionamento. Esta medida é referenciada por CTER (Custo Total Estimado para o Roteamento). O custo médio de conexões por aresta é 3,03. Assim, o vértice 3 que possui o mais alto grau concentra 23,07% do total das arestas e 31,64% dos recursos de roteamento.

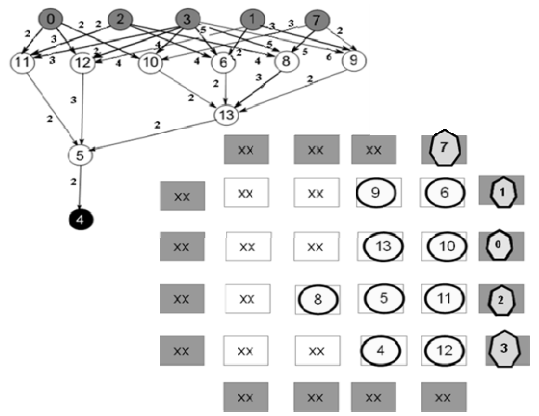


Figura 4: Etapa final de DF: Custo Total Estimado para o roteamento.

Mapeamento em BF

Na travessia do grafo em BF, o algoritmo irá priorizar os nós com maior grau de saída (*fanouts*) para estabelecer uma ordem parcial na travessia. A travessia começa pelo vértice de maior grau. Na ordenação das arestas adjacentes ao vértice, o critério será o grau de saída dos vértices destino.

Considere novamente o grafo da Figura 1. A Figura 5 mostra o posicionamento parcial partindo

do vértice 3, que possui o maior grau de saída (*fanout*) com 6 arestas. A busca em largura irá posicionar todos os seus descendentes (6, 8, 9, 10, 11, 12) em nós LUTs próximas a entrada 3.

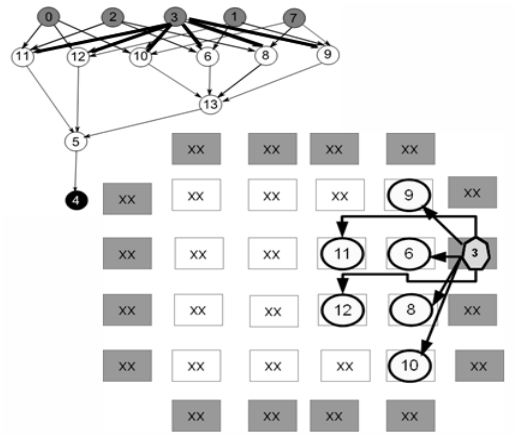


Figura 5: Etapa Parcial de BF: posicionamento do vértice entrada 3 e seus descendentes.

A travessia continua em largura. Como todos os descendentes do vértice 3 tem o mesmo grau, existem várias possibilidades. O algoritmo é guloso e só explora uma delas. Semelhante ao mapeamento DF, cada aresta a → b é visitada e a posição do vértice destino b é determinada pelo posicionamento do vértice a. A primeira vez que o vértice é visitado que irá determinar sua posição. Por exemplo, se o descendente 9 do vértice 3 é o primeiro na busca em largura, a visita a aresta 9 → 13 irá determinar a posição do vértice 13. As outras arestas: 6→13, 8→13, 10→13 não influenciam no posicionamento de 13, como mostra a Figura 6.

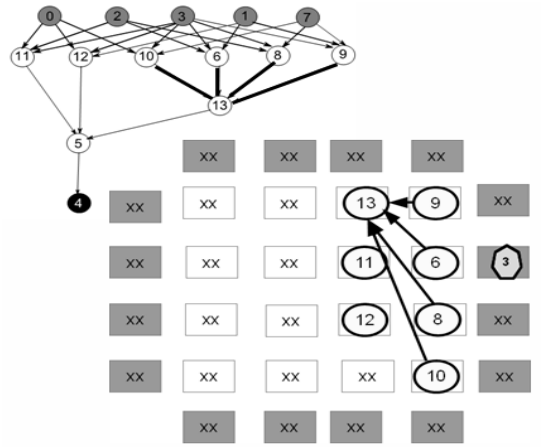


Figura 6: Etapa Parcial de BF: posicionamento do nó 13.

E finalmente, após o posicionamento de todos os vértices e seus descendentes, é possível estimar o custo total do roteamento. A Figura 7 mostra o grafo de entrada com os custos das conexões nas arestas. O custo total para a travessia BF é 75, que é menor que o valor 79 obtido pela travessia DF. O custo médio por aresta é 2,88. O vértice 3 que tem o maior grau concentra 23,07% do total das arestas e 21,33% dos recursos de roteamento. Para a travessia em DF, o vértice 3 concentrava 31% dos recursos de roteamento. Podemos observar que o BF reduziu o custo das conexões do vértice de alto grau. O caminho crítico para BF é 13 que é maior que o caminho crítico 12 gerado pela DF. Este exemplo é ilustrativo e não podemos tirar conclusões sobre o desempenho das duas travessias.

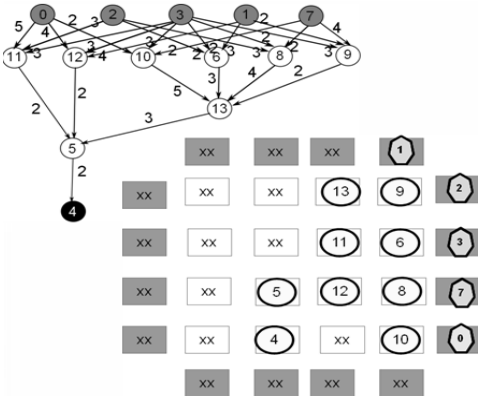
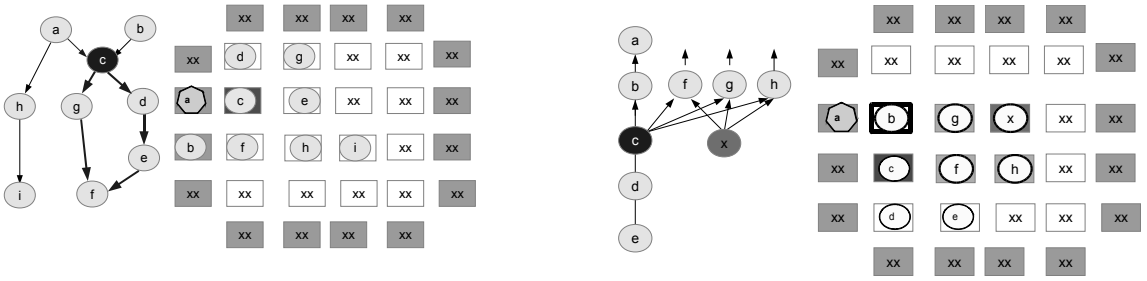


Figura 7: Custo Total Estimado para o Roteamento – após etapa de posicionamento baseado em BF.

Estratégias locais de otimização

A estratégia 1 (E1) tem por finalidade priorizar a reconvergência da classe de *fanout* 2 sem nenhuma reconvergência aninhada. O algoritmo percorre em DF (Figura 8(a)), mas ao encontrar um vértice com grau igual a 2, ele percorre no sentido oposto e os vizinhos são armazenados em uma lista antes de continuar a busca em profundidade. Enquanto o vértice possuir *fanout* 1, o algoritmo continua a busca até localizar um vértice já visitado, destacando uma reconvergência. Caso encontre um vértice que possua *fanout* maior que 1, a busca é interrompida.

A estratégia 2 (E2) foi desenvolvida utilizando a abordagem de busca em largura (BF), como ilustrado na Figura 8(b). Nela as entradas são ordenadas pelo maior grau de saída (*fanout* altos), posteriormente, no nível seguinte (interno), também será ordenado pelo *fanout* de maior valor e assim sucessivamente. Todo esse processo é executado enquanto não for encontrada uma saída. O processo é repetido com todas as demais entradas.



(a) E1 - Ordenação da busca DF pelos *fanout*. (b) E2 - Ordenação da busca BF pelos *fanout.*

Figura 8: Estratégias locais de otimização

Algoritmo

Em tempo de síntese, as ferramentas de CAD são responsáveis por gerar o grafo do circuito que pode ser armazenado em um arquivo como uma lista de pares de arestas, na ordem de travessia (BF e/ou DF). O próximo passo é posicionamento dos vértices na matriz FPGA, onde usamos as heurísticas propostas e comparamos os resultados com o algoritmo de P&R [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. Para o roteamento usaremos a ferramenta VPR.

Como já mencionado, este trabalho difere do algoritmo proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], ao explorar a travessia dos grafos no sentido inverso (entrada-saída), detectando durante a travessia os vértices com grau alto de *fanout* e propondo estratégias para priorizar o posicionamento deles e de seus descendentes. O caminho crítico e o comprimento total dos fios são usados para avaliar a qualidade das soluções propostas. O posicionamento é ideal quando uma aresta A→B do circuito é mapeada em dois nós adjacentes x→y no grafo do FPGA. Porém, nem sempre é possível, seja devido uma limitação física no grafo FPGA ou da ordem na qual as arestas são visitadas, uma vez que é uma solução gulosa para um problema NP-completo.

Uma contribuição deste trabalho, além do percurso no sentido inverso, foi à função de custo dos nós adjacentes no sentido inverso e sua simplificação. No modelo proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], existem 4 custos associados a cada entrada da LUT, como mostrado na Figura 9(b). Como nossa abordagem faz somente o posicionamento podemos simplificar o custo usando apenas o custo médio, reduzindo a busca por um fator 4x (Figura 9(c)). Considere o nodo $N_{i,j}$ na Figura 9(a), que mostra as posições dos 8 nodos na vizinhança do nodo $N_{i,j}$. A Figura 9(b) mostra, dentro de cada nodo, o custo de posicionamento para os nodos adjacentes como proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. Os custos são calculados em função do roteamento. Cada nodo tem 4 entradas, uma em cada direção: sul, oeste, norte e leste. A saída do nodo $N_{i,j}$ é sempre pela direção sul. Por exemplo, o nodo do topo $N_{i-1,j}$ tem custo 3, 2 ,3 e 2 se conectar nas entradas oeste, norte, leste e sul, respectivamente. O custo médio será (3+2+3+2)/4 = 2. Para simplificar ainda mais o modelo, só usamos custos com números inteiros.

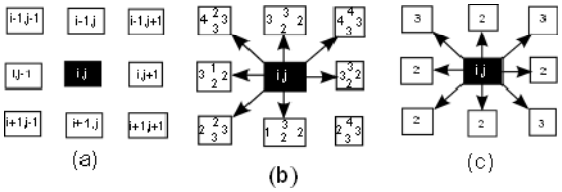


Figura 9: Vizinhança (a) LUTs FPGA (b) Custo LUT *output*

Alguns vértices não poderão ser posicionados nós adjacentes, pois não existem posições livres na vizinhança do vértice de origem da aresta, mesmo quando o conceito de nó adjacente é ampliado para custos maiores, estes casos podem ocorrer. Nesta situação, o algoritmo de busca em largura, procura a primeira posição vazia na vizinhança.

DISCUSSÃO DE RESULTADOS

Os resultados foram medidos usando 15 benchmarks MCNC [BETZ, 2011], é um padrão adotado para avaliar várias ferramentas de posicionamento e roteamento de FPGA. A ferramenta VPR também foi utilizada para roteamento e medida do caminho crítico e do número total de recursos de roteamento (segmentos e comutadores). Os experimentos foram realizados em um Intel Core i3-2100, de 3,09 GHz, 3,41MB de cache LGA1155, 2 núcleos.

A proposta apresentada nesse trabalho é fazer uso de FPGA virtual semelhante ao proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], onde o FPGA virtual é composto por uma matriz 100x100, e cada canal contem 50 trilhas de roteamento. O objetivo principal é reduzir o tempo de execução do posicionamento, além de

minimizar o comprimento total de fios e não deteriorar o caminho crítico. Além disso, avaliar também, qual será o tempo necessário para rotear o circuito com o algoritmo VPR.

Neste trabalho foram desenvolvidas e implementadas 4 variações do algoritmo proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], todas são no sentido entradas para saídas que difere da proposta de [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. Duas são em largura, a solução BF e a solução em largura E2. As outras duas variações são em profundidade, a variação DF e E1. Para comparar a qualidade das soluções obtidas pelas variações propostas, usamos o algoritmo original proposto por [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], referenciado como P&R e o posicionamento do VPR.

A Tabela 1 apresenta o tempo de posicionamento para as abordagens propostas. As variações DF e BF são é 1002x e 1343x mais rápidas que o VPR. As estratégias E1 e E2 são 432x e 1117x mais rápidas que o VPR. Comparando com os resultados do tempo de posicionamento para abordagem P&R [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], que foram disponibilizados apenas para 10 benchmarks (representamos com NA os benchmarks que não foram avaliados pelo P&R[FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]), o P&R é 460x mais rápido que o VPR e em relação ao DF, BF e E1, obteve uma perda de 82,4%, 76,7% e 85,2%, respectivamente. Em relação a E2, o P&R obteve um ganho de 34,3%.

Tabela 1: Tempo de execução do Posicionamento

				VPR	DF		BF		P&R		E2		E1	
Bench	CLB	Input	Output	sec	ms	ganho	ms	ganho	ms	ganho	ms	ganho	ms	ganho
fir16	47	1	1	0,0245	0,28	87,50	0,297	82,49	0,1	245	0,265	92,45	0,45	54,44
9symml	179	9	1	0,0947	0,39	242,82	0,375	252,53	1,29	73,41	0,313	302,56	0,562	168,51
alu2	241	10	6	0,135	0,421	320,67	0,39	346,15	1,28	105,47	0,343	393,59	0,593	227,66
Alu4	415	14	8	0,2721	0,515	528,35	0,546	498,35	0,74	367,70	0,484	562,19	0,765	355,69
term1	261	34	10	0,1685	0,453	371,96	0,5	337,00	0,492	342,48	0,421	400,24	0,64	263,28
C6288	2417	32	32	2,8288	1,781	1588,32	1,82	1554,29	NA	NA	1,828	1547,48	4,6	614,96
too_large	992	38	3	0,8764	0,859	1020,26	0,87	1007,36	1,58	554,68	0,843	1039,62	3,42	256,26
C7552	2536	207	108	3,0689	1,76	1743,69	1,83	1676,99	NA	NA	1,828	1678,83	2,546	1205,38
Misex3	771	14	14	0,6885	0,672	1024,55	0,703	979,37	1,05	655,71	0,671	1026,08	3,21	214,49
Ex5p	646	8	63	0,5892	0,672	876,79	0,703	838,12	0,88	669,55	0,655	899,54	3,23	182,41
Ex1010	964	10	10	0,9604	0,782	1228,13	0,79	1215,70	1,54	623,64	0,765	1255,42	3,28	292,80
PDC	2857	16	40	4,7344	1,76	2690,00	1,84	2573,04	4,87	972,16	1,859	2546,75	4,71	1005,18
cordic	2058	23	2	2,357	6,72	350,74	1,46	1614,38	NA	NA	1,391	1694,46	4,125	571,39
dalu	1618	75	16	1,6355	1,641	996,65	1,15	1422,17	NA	NA	1,141	1433,39	3,79	431,53
t481	2073	16	1	2,7445	1,4	1960,36	1,43	1919,23	NA	NA	1,453	1888,85	4,29	639,74

A Tabela 2 apresenta o caminho crítico, ou seja, o caminho mais longo no grafo da FPGA para as abordagens propostas neste trabalho, afim de comparar com os algoritmos de posicionamento P&R [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] e VPR [23]. Em relação à ferramenta VPR como referência, o DF apresentou um aumento médio do caminho crítico em 47%, o BF gerou um aumento de 81%, enquanto que o P&R é equivalente ao VPR. E em relação às estratégias E1 e E2, estes tiveram um aumento de 13% e 49% respectivamente. Portanto as heurísticas baseadas em profundidade tem um comportamento melhor quando o critério de custo é o caminho crítico, pois uma das informações usadas no percurso é a priorização do caminho crítico inicial.

Tabela 2: Caminho Crítico

Bench	DF	BF	P&R	VPR	E2	E1
fir16	13,23	14,1	12,1	12,7	14,01	14,12
9symml	9,71	10,76	9,7	8,76	13,7	10,14
alu2	23,14	29,63	23,1	25,31	37,12	23,13
Alu4	26,11	34,56	24,2	25,8	37,47	26,51
term1	8,69	9,53	8,7	8,85	11,81	9,87
C6288	128,48	166	86,74	82	25,41	111,34
too_large	42,9	50,94	32,1	38,09	81,71	37,87
C7552	39,61	67,65	24,63	23,6	73,5	28,5
Misex3	21,22	24,38	12,7	17,89	36,51	23,44
Ex5p	24,55	40,48	15,5	14.43	38,16	27,2
Ex1010	26,67	35,88	13,8	17,89	35,36	25
PDC	44,96	24,65	26,6	25,3	60,25	35,76
cordic	31,71	36,41	13,92	15,4	30,33	23,33
dalu	45,56	56,55	26,81	28,61	72,66	38,57
t481	22,92	25,51	14,34	14,76	33,48	22,61

Tabela 3 apresenta os tempos de execução para rotear cada benchmark após a etapa de posicionamento das abordagens DF, BF, E1 e E2 propostas neste trabalho, do P&R proposto em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] e da ferramenta VPR [23]. Para facilitar a comparação, uma coluna mostra o ganho em relação ao posicionamento gerado pelo

VPR. O roteamento foi executado com os parâmetros: -route_only –route_chan_witch 50. Na coluna representada pela sigla G/P indica G-ganho (valores acima de 1) e P-perda (valor abaixo de 1).

Tabela 3: Tempo de Roteamento

	DF	G/P	BF	G/P	P&R	G/P	VPR	E2	G/P	E1	G/P
Bench	sec		sec		sec		sec	sec		sec	
fir16	0,03	0,00	0,02	1,10	0,03	0,00	0,03	0,03	0,00	0,05	0,50
9symml	0,17	0,92	0,08	1,84	0,17	0,90	0,15	0,17	0,90	0,24	0,65
alu2	0,19	1,00	0,11	1,68	0,23	0,81	0,19	0,18	1,02	0,18	1,02
Alu4	0,34	0,87	0,18	1,63	0,44	0,67	0,30	0,36	0,83	0,33	0,90
term1	0,25	0,75	0,12	1,57	0,28	0,67	0,19	0,24	0,80	0,28	0,68
C6288	2,60	0,86	2,69	0,84	1,54	1,46	2,25	5,63	0,40	2,66	0,85
too_large	2,10	0,94	2,17	0,91	1,19	1,66	1,97	2,32	0,85	2,15	0,92
C7552	3,12	0,81	3,32	0,76	1,71	1,47	2,52	3,69	0,68	3,42	0,74
Misex3	1,77	0,82	1,26	1,15	0,92	1,58	1,45	1,71	0,85	1,85	0,79
Ex5p	1,62	0,90	1,67	0,87	0,75	1,94	1,46	1,67	0,87	1,61	0,91
Ex1010	2,07	0,89	1,25	1,47	1,52	1,20	1,83	2,42	0,76	2,14	0,86
PDC	3,65	1,18	3,41	1,26	21,10	0,20	4,31	15,40	0,28	6,08	0,71
cordic	3,53	0,80	3,41	0,83	1,85	1,53	2,84	3,88	0,73	3,10	0,92
dalu	2,51	0,89	2,68	0,83	1,10	2,02	2,23	2,78	0,80	2,56	0,87
t481	1,44	4,18	5,51	1,09	4,78	1,26	6,02	5,19	1,16	5,88	1,02

Em geral, o P&R gerou um desempenho mais próximo do VPR. As soluções DF, BF, E1 e E2 foram em média piores e apresentaram resultados semelhantes, exceto para o PDC e o T481. Para o PDC que é o maior circuito, o DF e o BF obtiveram um ganho de 19% e 21% em relação ao VPR, respectivamente, enquanto a E1 obteve uma perda de 41%, e as abordagens E2 e P&R tiveram perda bem considerável, acima de 200%, em relação ao VPR.

A Tabela 4 apresenta o total de segmentos. Na coluna representada pela sigla G/P indica G-ganho e P-perda, para facilitar a análise.

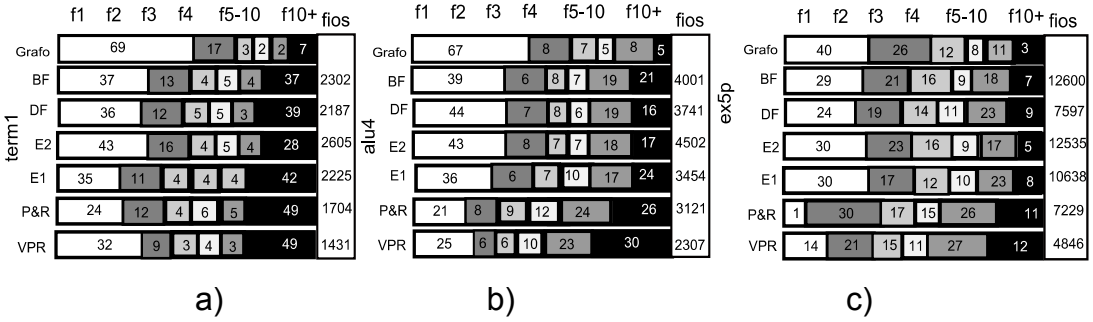
Tabela 4: Segmentos de fiação

Bench	DF	G/P	BF	G/P	P&R	G/P	VPR	E2	G/P	E1	G/P
term1	109	28,24	118	38,82	123	44,71	85	128	50,59	103	21,18
9symml	1275	42,78	1359	52,18	1141	27,77	893	1559	74,58	1213	35,83
fir16	1662	41,57	1773	51,02	1593	35,69	1174	2121	80,66	1614	37,48
Misex3	3741	61,81	4001	73,05	3121	34,99	2312	4502	94,72	3454	49,39
Ex1010	2187	59,75	2302	68,15	1704	24,47	1369	2605	90,28	2225	62,53
cordic	20161	123,27	24248	168,53	8726	-3,37	9030	47491	425,92	20856	130,96
t481	10300	91,17	17381	222,59	6522	21,05	5388	21905	306,55	15954	196,10
Ex5p	33838	229,71	40020	289,94	9452	-7,90	10263	48053	368,22	35226	243,23
alu2	9654	58,11	14552	138,32	8845	44,86	6106	18285	199,46	12994	112,81
Alu4	7597	38,48	12600	129,68	7229	31,77	5486	12535	128,49	10638	93,91
C7552	18620	86,65	25214	152,75	12450	24,80	9976	26091	161,54	18026	80,69
PDC	68096	114,92	30001	-5,31	53045	67,42	31684	90713	186,31	69025	117,85
dalu	31519	164,87	38435	222,98	11919	0,16	11900	38466	223,24	22108	85,78
too_large	19825	180,53	21607	205,75	6268	-11,31	7067	33511	374,19	21483	203,99
C6288	41411	228,24	48071	281,03	13218	4,77	12616	57928	359,16	33779	167,75

Pode-se observar que o VPR é superior a todas as heurísticas, pois avalia diferentes opções de posicionamento com a técnica SA, enquanto que as outras soluções são gulosas e avaliam apenas 1 posicionamento, reduzindo significativamente o tempo de execução. O preço a ser pago é o uso de mais conexões. Podemos observar que para maioria dos circuitos, as soluções baseadas em profundidade P&R, DF e E1 tem o melhor desempenho. As soluções em largura usam, em geral, uma quantidade maior de fios, exceto para o circuito PDC que apesar de ser um caso específico é o maior circuito. Para o PDC em relação ao VPR, o BF obteve ganho de

5,3%, e DF, E1, E2 e P&R obtiveram perda de 114%, 117%, 186% e 67%, nesta ordem.

A distribuição de *fanouts* em circuitos não é uniforme, apesar da média ser próxima de 3. Para facilitar a análise, agrupamos os *fanouts* maiores em classes com vários graus diferentes. O índice f5-10 representa as classes com os *fanouts* f5, f6, f7, f8, f9 e f10. A classe f10+ por sua vez representa todos os vértices com *fanout* i > 10. A maioria dos vértices possuem grau em f1 e f2, que justifica a média ser em torno de 3 como mostram a figura a seguir.



a) b) c)

Figura 10: Disposição de vértices para os benchs: term1, alu4 e ex5p.

Os nós de *fanouts* elevados contribuem para o custo total de fios, mesmo que estes sejam poucos frequentes no grafo original. Por exemplo, considera-se o benchmark term1 (Figura 10 (a)) que possui o *fanout* alto na classe f10+ com apenas 6.5% dos nós de entrada, após o mapeamento, estes nós consumirão 36.9% (BF), 38.2% (DF), 27.8% (E2), 41.5% (E1) e 48.2% (P&R) do total de fios.

Comparando as estratégias baseadas em largura (BF e E2) e profundidade (DF e E1), as últimas são melhores nos três benchmarks considerando o consumo total de fios. Novamente, como já mencionado, as estratégias BF e E2 tem foco nas ligações com múltiplos *fanout*. Essas estratégias priorizam a classe f10+. Para circuito term1, BF e E2 usam 851 e 729 fios, nesta ordem, em comparação com P&R que usa 834 fios. Porém a melhora nos múltiplos *fanout* não compensa o número maior de fios que são usados pelas outras classes, que leva a um resultado pior se olharmos o total de fios usados pelas heurísticas DF e E1.

Podemos observar que a estratégia E2 é pior em 12 dos 15 casos avaliados se olharmos o custo total. Porém como ela prioriza o posicionamento dos vértices de alto grau, a E2 é a melhor de todas as soluções para reduzir o consumo de comutadores da classe F10+ em 8 dos 15 casos, sendo que em 7 casos é uma das melhoras. Ou seja, ela cumpre o seu objetivo de otimizar a classe f10+, porém a priorização destes vértices deteriora o custo total, pois gera um posicionamento ruim para as outras classes.

CONCLUSÃO

Este trabalho explorou uma nova técnica de posicionamento de FPGA e com novas variações, que foi proposta inicialmente em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. O FPGA é modelado por um grafo com uma correspondência direta com sua estrutura física. Neste trabalho o modelo é simplificado e gera resultados semelhantes em muitos casos quando comparado ao trabalho anterior [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013]. O posicionamento consiste em mapear o grafo do circuito em um grafo FPGA. A localidade dos vértices foi explorada durante esse processo. Porém, diferente do trabalho anterior [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013], buscou-se explorar e avaliar outras estratégias de travessia. As abordagens em largura e profundidade foram avaliadas. O sentido do percurso também foi modificado. Além disso, duas estratégias locais para explorar o grau 2 (*fanout*=2) e o grau múltiplo de saída dos vértices (alto *fanout*) foram avaliadas.

Com base nos resultados experimentais, o resultado obtido pelo algoritmo de posicionamento proposto neste trabalho foi em média de 1002x e 1343x, em profundidade e em largura, respectivamente, quando comparados à ferramenta VPR [BETZ; CAMPBELL; FANG; JAMIESON et al, 2008; BETZ, 2011].

O objetivo é ter heurísticas rápidas que sejam viáveis para serem usadas em ambientes de reconfiguração parcial durante a execução das aplicações. O espaço de soluções é exponencial e cresce muito rápido. Entretanto, mostramos neste trabalho que os 4 algoritmos propostos sempre geram soluções que podem ser roteadas. Usamos como referência o posicionamento baseado em SA do VPR [BETZ; CAMPBELL; FANG; JAMIESON et al, 2008]. Esta técnica gera milhares ou até milhões de tentativas a mais que as abordagens propostas aqui. Além disso, apesar de fazer apenas uma tentativa, as abordagens propostas aqui conseguem gerar um resultado que no pior caso é apenas 2x pior no consumo de recursos de roteamento. Como o número de fios é fixo, um consumo maior não afeta a funcionalidade do circuito. Em termos de caminho crítico, os resultados mostram que as heurísticas gulosas aumentam em 30-60% o caminho crítico, dobrando em alguns casos. Como já ressaltado, o tempo de posicionamento é reduzido em 3 à 4 ordens de grandeza, para uma perda de 2x apenas.

A alternativa explorada aqui é a nova técnica proposta em [FERREIRA; ROCHA; SANTOS; NACIF; WONG; CARR, 2013] baseada apenas em profundidade. Podemos observar que se pode usar tanto travessia em largura quando em profundidade, como contribuição deste trabalho. O tempo de execução permanece baixo e os posicionamentos geram uma solução que podem ser roteada. Em geral, a profundidade é melhor. Apesar da largura reduzir os custos relativos dos vértices com alto grau, o consumo total e o caminho crítico em média, para os circuitos avaliados, é pior.

REFERÊNCIAS

ASSIS, Alex Damiany. Um Algoritmo de Posicionamento e Roteamento Polinomial para Arquiteturas Reconfiguráveis de Grão Grosso com Redes Multiestágio. Viçosa: MG, 2010. 14 p. Tese (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Viçosa, Viçosa, 2010.

HAUCK, Scott; DEHON, Andre. Reconfigurable Computing: The Theory and Practice Of FPGA-Based Computation. Elsevier. 2008.

SILVA, Marcos Vinícius da. Exploração do Espaço de Projeto de Arquiteturas Reconfiguráveis em Arranjos. Viçosa: MG, 2010. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Viçosa, Viçosa, 2006.

H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris, and M. Hubner, On sup-porting efficient partial reconfiguration with just-in-time compilation, in PhD Forum (IPDPSW), IEEE, 2012.

Y.L. Wu; D. Chang. On the NP-completeness of regular 2-D FPGA routing architectures and a novel solution. IEEE/ACM International Conference on Computer-aided design, pages 362-366. CA, USA, 1994.

GONZALEZ, Jose Artur Quilici. Uma Metodologia de Projetos para Circuitos com Reconfiguração Dinâmica de Hardware aplicada a Support Vector Machines. São Paulo: SP, 2006. Tese (Doutorado) – Programa de Pós-Graduação em Engenharia (área do conhecimento: Microeletrônica), USP: Escola Politécnica, São Paulo, 2006.

LIMA, Ednaldo M. V. de. Posicionamento Automático em Circuitos Implementados em FPGAs: Desenvolvimento de Rotinas de Aceleração Computacional. João Pessoa: Paraíba, 2008. Tese (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal da Paraíba, João Pessoa, 2008.

MOREANO, Nahri Balesdent. Algoritmos para Alocação de Recursos em Arquiteturas Reconfiguráveis. Campinas: SP, 2005. Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, Instituto de Computação Universidade Estadual de Campinas – UNICAMP, São Paulo, 2005.

ROSE, Jonathan; BROWN, Stephen. Flexibility of Interconnection Structures for Field-Programmable Gate Arrays. IEEE Journal of Solid-State Circuits, vol. 26, n 03, march 1991.

SHARMA, Akshay. Place and Route Techniques for FPGA Architecture Advancement. Washington: EUA, 2005. Tese (Doutorado) – Programa de Pós-Graduação em Filosofia (Área de conhecimento: Engenharia Elétrica), University of Washington Graduate Scholl, Washington, 2005.

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.

K. Papadimitriou, A. Dollas, S. Hauck. Performance of partial reconfiguration in FPGA systems: A survey and a cost model. ACM Trans. Reconf. Technol. Syst., vol. 4, Dec. 2011.

M. Handa and R. Vemuri. An efficient algorithm for finding empty space for online FPGA placement. in Proceedings of DAC, 2004.

FERREIRA, Ricardo; ROCHA, Luciana; SANTOS, Andre Gustavo dos; NACIF, Jose Augusto; WONG, Stephan; CARR, Luigi. A Run-time Graph-Based Polynomial Placement and Routing Algorithm for Virtual FPGAs. 23rd International Conference on Field Programmable Logic and Applications. Porto, Portugal, 2013.

A. Ludwin and V. Betz. Efficient and Deterministic Parallel Placement for FPGAs. ACM Trans. Des. Autom. Electron. Syst., vol. 16, no. 3, 2011.

M. Lin and J. Wawrzynek Improving FPGA placement with dynamically adaptive stochastic tunneling. CAD of Integrated Circuits and Systems, IEEE Trans-actions on, vol. 29, no. 12, 2010.

M. Xu, G. Grewal, and S. Areibi. Starplace: A new analytic method for FPGA placement. Integration, the VLSI Journal, vol. 44, no. 3, pp. 192 – 204, 2011.

H. Bian, A. C. Ling, A. Choong, and J. Zhu. Towards scalable placement for FPGAS. in Proceedings of FPGA, 2010.

S. Y. Chin and S. J. Wilton. Towards scalable FPGA cad through architecture. In Proceedings of FPGA, 2011.

ALTERA CORPORATION - White Paper. FPGA Run-Time Reconfiguration: Two Approaches. San Jose, CA, 2008. Disponível em: http http://www.altera.co.jp/literature/wp/wp-01055-fpga-run-time-reconfiguration.pdf. Acesso: 10 de novembro de 2011.

ALTERA CORPORATION - FPGAs. Disponível em: www.altera.com/products/fpga.html. Acesso: 12 de maio de 2012.

BETZ, Vaughn. 175. Vpr: SPEC CPU2000 Benchmark Description File. Disponível no site:

http://www.spec.org/cpu2000/CINT2000/175.vpr/docs/175.vpr.html. Acesso: 15 de setembro de 2011.

BETZ, Vaughn; CAMPBELL, Ted; FANG, Wei Mark; JAMIESON, Peter; KUON, Ian; LUU, Jason; MARQUARDT, Alexander; ROSE, Jonathon; YE, Andy. VPR and T-VPack1 User's Manual. Summer 2008 VPR 5.0 Full Release, 2008.

BETZ, Vaughn. FPGA Architecture for the Challenge. Disponível no site: http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html. Acesso: 15 de setembro de 2011.

CARVALHO, Gustavo Rezende. Heurísticas para a fase de Roteamento de Circuitos Integrados Baseados em FPGAs. João Pessoa: PB, 2010. Tese (Mestrado) – Programa de Pós-Graduação em Informática (área de conhecimento: Sistemas de Informação), UFPB: Universidade Federal da Paraíba, João Pessoa, 2010.

M. Gericota, G. Alves, M. Silva, J. Ferreira. Run-time management of logic resources on reconfigurable systems. in Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 974–979.

M. Hubner, P. Figuli, R. Girardey, D. Soudris, K. Siozios, and J. Becker. A heterogeneous multicore system on chip with run-time reconfigurable virtual FPGA architecture. In Workshops and Phd Forum (IPDPSW), 2011.