

UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

Otimização do tempo de execução em algoritmos de roteamento global: Explorando métricas e paralelismo

Jacques de Jesus Figueiredo Schmitz Junior

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande - FURG, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Computação

Orientador: Prof. Dr. Paulo Francisco Butzen
Co-orientadora: Prof^a. Dr^a. Cristina Meinhardt

Rio Grande, 2019

Ficha catalográfica

S355o Schmitz Junior, Jacques de Jesus Figueiredo.
Otimização do tempo de execução em algoritmos de roteamento
global : explorando métricas e paralelismo / Jacques de Jesus
Figueiredo Schmitz Junior. – 2019.
90 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande –
FURG, Programa de Pós-Graduação em Computação, Rio
Grande/RS, 2019.

Orientador: Dr. Paulo Francisco Butzen.

Coorientadora: Dra. Cristina Meinhardt.

1. EDA 2. VLSI 3. Roteamento 4. Roteamento global 5.
Microeletrônica I. Butzen, Paulo Francisco II. Meinhardt, Cristina III.
Título.

CDU 004.2



MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

ATA DE SESSÃO DE DEFESA DE DISSERTAÇÃO DE MESTRADO

Ata nº 5/2019

Na data de 19 de junho de 2019, às 16 horas, ocorreu a Sessão de Defesa de Dissertação de Mestrado de Jacques de Jesus Figueiredo Schmitz Junior, que apresentou a dissertação intitulada "Otimização do tempo de execução em algoritmos de roteamento global: Explorando métricas e paralelismo", realizada sob a orientação do Prof. Dr. Paulo Francisco Butzen e coorientação da Profª. Drª. Cristina Meinhardt. A banca examinadora foi constituída pelo Prof. Dr. Ricardo Augusto da Luz Reis (UFRGS) e Prof. Dr. Odorico Machado Mendizabal (PPGCOMP/FURG), sob a presidência do orientador. Após a apresentação do trabalho, a banca arguiu o candidato e, a seguir, deliberou pela

- ☒ aprovação da Dissertação
- ☐ aprovação da Dissertação, sugerindo modificações no texto
- ☐ reprovação da Dissertação

Rio Grande, 19 de junho de 2019.

Prof. Dr. Ricardo Augusto da Luz Reis

Prof. Dr. Odorico Machado Mendizabal

Prof. Dr. Paulo Francisco Butzen
Orientador

Profª. Drª. Cristina Meinhardt
Coorientadora

AGRADECIMENTOS

Meu agradecimento inicial é a Deus, por colocar pessoas maravilhosas em meu caminho e ter me permitido chegar até aqui. Agradeço minha mãe Valdirene, meu irmão Eduardo e minha noiva Marília por me darem força e apoiarem nos momentos difíceis, vocês foram essenciais para a realização deste trabalho.

Agradeço especialmente ao meu Orientador e minha Co-orientadora, vocês foram maravilhosos desde o momento em que me receberam no Programa de Pós-Graduação até a reta final da Dissertação, sem vocês este trabalho não seria possível, aprendi muito com vocês e espero um dia poder retribuir repassando todo conhecimento que adquiri nesta caminhada a outras pessoas.

"Tudo o que temos de decidir é o que fazer com o tempo que nos é dado."—
J.R.R. TOLKIEN

RESUMO

SCHMITZ JUNIOR, Jacques de Jesus Figueiredo. **Otimização do tempo de execução em algoritmos de roteamento global: Explorando métricas e paralelismo.** 2019. 96 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

Roteamento é a etapa do fluxo de síntese física de circuito integrados responsável pela geração das conexões do circuito após as células estarem posicionadas. A evolução na tecnologia dos transistores e, consequente evolução dos circuitos, trouxe consigo novos desafios para todas as etapas da síntese física. Rotear um circuito VLSI (*Very Large Scale Integration*) é uma tarefa de difícil resolução, tanto por possuir um grande volume de dados a ser processado, quanto por se tratar de um problema NP-completo, significando que não existe uma solução ótima que possa ser processada em tempo polinomial. Para resolver o problema, é necessária a utilização de algoritmos gulosos, heurísticas, aprendizado de máquina ou técnicas mais específicas para trabalhar com as estruturas de dados de forma mais intuitiva e obter melhores resultados.

Apesar dos roteadores estado-da-arte gerarem resultados satisfatórios, ainda é possível realizar melhorias nestes. Este trabalho explora técnicas de otimização de programação aplicadas a um roteador estado-da-arte de código-fonte aberto. A partir das análises realizadas no roteador NTHU Route, apresentamos melhorias envolvendo parâmetros utilizados nos algoritmos, e aplicação de paralelismo em alguns dos estágios do roteador, buscando uma melhoria no desempenho e na qualidade dos resultados. Com a aplicação das estratégias envolvendo alteração nos parâmetros, mais especificamente com a aplicação de um limiar para aceitação de congestionamento na fase principal do roteador, foi possível diminuir o tempo de execução de todos *benchmarks* utilizados em média em 9%, sendo no melhor caso observada uma redução de aproximadamente 20% sem grandes impactos no comprimento total de fio. Com a aplicação de paralelismo no estágio de refinamento, foi possível melhorar o tempo tomado nesta fase em até 20%, com uma melhoria média de 10 %. Com experimentos aplicando paralelismo na fase principal em conjunto com o aumento da caixa de expansão de busca para ligar dois pinos, o tempo de execução dobrou, devido ao comportamento de busca para achar as soluções nesta fase. Ao juntar as duas modificações, aplicando um limiar para aceitação de congestionamento na fase principal e paralelismo em blocos menores nas duas fases sem o aumento da caixa de expansão de busca, foi possível obter melhorias no tempo de execução de até 20%, sem impacto considerável no comprimento total de fio. Explorar estas técnicas de otimização nos algoritmos de roteamento mostrou bons resultados, que podem ser aplicados a diferentes ferramentas de roteamento e de outras etapas da síntese física.

Palavras-chave: EDA, VLSI, Roteamento, Roteamento global, Microeletrônica.

ABSTRACT

SCHMITZ JUNIOR, Jacques de Jesus Figueiredo. **Circuits Global Routing**. 2019. 96 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

Routing is the stage of integrated circuits physical synthesis flow responsible for generating the connections of the circuit after the cell placement. The evolution in transistors technology along with circuits growth, brought new challenges for all physical synthesis stages. Routing a VLSI (Very Large Scale Integration) circuit is a hard task to solve, due to the great amount of data to be processed, and the fact that it is a NP-complete problem, meaning that there is not an optimum solution that can be processed in polynomial time. To solve this problem, it is necessary the use of greedy algorithms, heuristics, machine learning or more specific techniques to work along with the data structures in a more intuitive way and obtain better results.

Even with the satisfactory results generated by the state-of-the-art global routers, it is still possible to do enhancements in those. This paper explores programming optimization techniques applied to a state-of-the-art open-source global router. From the analysis we made in the NTHU Route, we could present enhancements involving the algorithms parameters, and the application of parallelism in some of the router stages, seeking for improvements in the performance and quality of the results. With the application of the strategies involving parameters, more specific with the application of an overflow threshold for accepting overflowed results at the main phase of the router, it was possible to reduce the runtime of all used benchmarks in average in about 9%, being the best case a reduction of about 20% with no further impacts on the total wirelength. With the application of parallelism strategies at the refinement stage, it was possible to improve the runtime of this phase in 20%, with the average improvement by 10%. With experiments applying parallelism at the main phase together with the increment in the bounding-box expansion for connecting two pins, the execution time doubled, because of the behavior for finding solution on this phase. By mixing both modifications, applying the threshold for overflow acceptance on the main phase, and the parallelism in minor blocks in both phases without the increment of the bounding-box expansion, it was possible to obtain improvements in the general runtime in about 20%, with no considerable impact in the total wirelength. Exploring this techniques of optimization on the routing algorithms showed good results, that can be applied in different routing and other physical synthesis tools.

Keywords: EDA, VLSI, Routing, Global Routing, Microelectronics.

LISTA DE FIGURAS

Figura 1	Projeção Lei de Moore 1970 - 2016 (REY, 2017).	16
Figura 2	Ciclo típico de projeto de Circuitos Integrados com divisão da parte física. Adaptado de (TUMELERO; BONTORIN; REIS, 2015).	22
Figura 3	Exemplos de <i>floorplan</i> não-fatiável (a) e fatiável (b) (WANG; CHANG; CHENG, 2009).	23
Figura 4	O processo de roteamento; (a) Resultado recebido de um posicionamento; (b) roteamento global; e (c) roteamento detalhado. Retirado de (WANG; CHANG; CHENG, 2009).	25
Figura 5	Decomposição do chip em global bins, e o grafo correspondente que formula o problema do roteamento global. Adaptado de (MOFFITT; ROY; MARKOV, 2008).	26
Figura 6	Roteamento detalhado lido a partir da ferramenta INNOVUS. Adaptado de (ISPD2018, 2018).	28
Figura 7	(a) ordem que os vértices são percorridos no método BFS, (b) ordem que os vértices são percorridos no método DFS.	33
Figura 8	(a) Fase de propagação da onda. (b) Fase de retração. Os quadradinhos maiores pintados são obstáculos, S é o nodo origem e T o nodo destino (WANG; CHANG; CHENG, 2009).	34
Figura 9	A*: os pontos azuis ilustram a propagação a partir do ponto de origem, a linha preta grossa indica um obstáculo, e a linha vermelha é o caminho traçado após o encontro dos nodos. Adaptado de ((SHERWANI, 2012)).	36
Figura 10	Ilustração do algoritmo <i>line-search</i> de (MIKAMI, 1968), nodos se expandem em linhas retas até encontrar obstáculos. Adaptado de (WANG; CHANG; CHENG, 2009).	38
Figura 11	Uma RSMT e sua grade de Hanan. (a) Uma rede P que consiste de quatro pinos p_1, p_2, p_3, p_4 . (b) A RSMT com dois pontos Steiner(s_1 e s_2). (c) A grade de Hanan de P . Adaptado de (WANG; CHANG; CHENG, 2009).	42
Figura 12	Roteamento de formas padrão: (a) Roteamento de formas padrão em "L"(uma dobra). (b) Roteamento de formas padrão em "Z"(duas dobras) (WANG; CHANG; CHENG, 2009).	43
Figura 13	Roteamento baseado nas diferentes ordens das redes. (a) Uma camada com duas redes de dois pinos A e B. (b) Uma solução ruim obtida por rotear A e depois B. (c) Uma solução melhor obtida por rotear B e depois A (WANG; CHANG; CHENG, 2009).	45

Figura 14	(a) Rede multi-pinos composta por 4 pinos. (b) Rede decomposta em redes de dois pinos utilizando a topologia de Steiner trees. Pontos pretos representam pinos e retângulos vermelhos os pontos Steiner (GAO; WU; WANG, 2008).	50
Figura 15	Fluxo do sistema do roteador NTHU v2. Adaptado de (CHANG; LEE; WANG, 2008).	51
Figura 16	Exemplo da mudança na ordenação das redes na etapa de desfazer e re-rotear. (a) Solução retirada de uma iteração anterior. (b) Solução do roteador NTHU. (c) Solução do roteador NTHU com a modificação na ordenação das redes (CHANG; LEE; WANG, 2008). .	52
Figura 17	Diferentes topologias de árvores de Steiner geradas por uma rede de 6 pinos. Adaptado de (PAN; CHU, 2006).	55
Figura 18	Fluxo seguido pelo FastRoute 4.0. Adaptado de (XU; ZHANG; CHU, 2009).	56
Figura 19	Comparação entre a arquitetura Manhattan (a) e a arquitetura X (b). Adaptado de (LIU et al., 2015).	58
Figura 20	Fluxo do roteador XGRouter. Adaptado de (LIU et al., 2015).	59
Figura 21	Fluxo do roteador NCTU-GR. Adaptado de (LIU et al., 2013).	61
Figura 22	Fluxo do roteador BFG-R. Adaptado de (HU; ROY; MARKOV, 2010). .	64
Figura 23	Fluxo do roteador de (HAN et al., 2014), onde (a) é o processo de inicialização, (b) a fase pós-roteamento, e (c) o processo de desfazer e re-rotear aplicado pelo roteador. Adaptado de (HAN et al., 2014). .	67
Figura 24	Grade de rotamento e redes a serem roteadas, (a) primeira iteração, somente as redes A e D podem ser roteadas ao mesmo tempo, (b) segunda iteração, as redes C, B, F e G podem ser roteadas ao mesmo tempo. Adaptado de (HAN et al., 2014).	69
Figura 25	Diminuição no congestionamento total vs tempo de execução pelo roteador NTHU-Route utilizando o <i>benchmark newblue5</i>	72
Figura 26	Diminuição no congestionamento total vs tempo de execução pelo roteador NTHU-Route utilizando o <i>benchmark newblue4</i>	72
Figura 27	Caracterização das funções mais custosas em termos de execução realizada com a ferramenta Callgrind para o roteador NTHU-Route antes de realizar modificações no <i>benchmark bigblue2</i>	75
Figura 28	Congestionamento total x número da iteração durante a execução do <i>benchmark adaptec1</i> , ...	77
Figura 29	Comparação entre o comprimento total de fio e o tempo que o roteador original leva para chegar à um resultado para os <i>benchmarks</i> aplicados, e após a aplicação da <i>flag</i> de aceitação de congestionamento no estágio principal do do roteador.	80
Figura 30	Comparação entre o comprimento total de fio e o tempo que o roteador leva para completar a fase de refinamento antes e depois da aplicação das melhorias.	82
Figura 31	Comparação entre o comprimento total de fio, tempo total de execução do roteador e tempo no estágio de refinamento antes e depois da aplicação de nossas modificações.	84

LISTA DE TABELAS

Tabela 1	Comparação dos resultados do NTHU-Route nos <i>benchmarks</i> do ISPD08 (onde existem -”a ferramenta não conseguiu encontrar uma solução), adaptado de (CHANG; LEE; WANG, 2008).	53
Tabela 2	Resultados do FastRoute 3.0 e FastRoute 4.0 utilizando os <i>benchmarks</i> do ISPD 08.	57
Tabela 3	Resultados de comprimento total de fio (TOF) dos roteadores NTHU-Router, FastRoute 3.0 e XGRouter aplicados nos <i>benchmarks</i> do ISPD98, adaptado de (LIU et al., 2015).	60
Tabela 4	Resultados obtidos pelo roteador NCTU-GR nos <i>benchmarks</i> do ISPD 2008 (LIU et al., 2013).	62
Tabela 5	Resultados obtidos pelo BFG-R aplicado nos <i>benchmarks</i> do ISPD 2008, adaptado de (HU; ROY; MARKOV, 2010).	64
Tabela 6	Ganhos na diminuição de congestionamento total (<i>TO</i>) das soluções dos roteadores BFG-R e NCTU-GR nos <i>benchmarks</i> difíceis de resolver do ISPD 2008, aplicando o algoritmo proposto por (SIDDIQI; SAIT, 2017).	66
Tabela 7	Resultados do roteador aplicado nos <i>benchmarks</i> do ISPD 2008 com o uso somente de paralelismo a nível de CPU, e CPU + GPU. Adaptado de (HAN et al., 2014).	69
Tabela 8	Características dos <i>benchmarks</i> do ISPD08 selecionados para uso nos experimentos (ISPD2008, 2017).	73
Tabela 9	Resultados envolvendo as métricas de comprimento total de fio, tempo para execução total e tempo para execução da fase de refinamento do roteador NTHU route.	74
Tabela 10	Redução nos resultados gerados pelo roteador NTHU após a adição da flag de aceitação de congestionamento para os <i>benchmarks</i> testados quando comparados como resultado no roteador NTHU original.	80
Tabela 11	Redução nos resultados gerados pelo roteador NTHU após a inserção de paralelismo em blocos menores de código na fase de refinamento quando comparados como resultado no roteador NTHU original.	81
Tabela 12	Comparação entre os resultados gerados pelo roteador NTHU original, e os resultados após a adição de paralelismo em blocos menores no estágio principal dentro do fluxo do roteador.	83

Tabela 13	Comparação entre os resultados gerador pelo roteador NTHU original, e os resultados gerados pelo roteador após a aplicação de todas as modificações nos parâmetros, aplicação da <i>flag</i> para aceitação de congestionamento e aplicação de paralelismo a nível de <i>multi-thread</i> utilizando a biblioteca OpenMP.	84
-----------	---	----

LISTA DE ABREVIATURAS E SIGLAS

ASIC	<i>Application-specific Integrated Circuit</i>
ATPG	<i>Automatic Test Pattern Generator</i>
BFS	<i>Breadth-first-search</i>
BIST	<i>Built-in Self-test</i>
BLMR	<i>Bounded-length Maze Routing</i>
CFOMR	<i>Circular Fixed-ordering Monotonic Routing</i>
CPU	<i>Central Processing Unit</i>
DEF	<i>Design Exchange Format</i>
DFS	<i>Depth-first-search</i>
DFT	<i>Design for Testability</i>
EDA	<i>Electronic Design Automation</i>
ESL	<i>Electronic System Level</i>
GPU	<i>Graphics Processing Unit</i>
GRG	<i>Global Routing Grid</i>
GT	<i>Game Theory</i>
HDL	<i>Hardware Description Language</i>
HPWL	<i>Half-perimeter Wire Length</i>
ILP	<i>Integer Linear Programming</i>
ISPD	<i>International Symposium on Physical Design</i>
LEF	<i>Library Exchange Format</i>
MST	<i>Minimum Cost Spanning Trees</i>
NLC	<i>Net Level Concurrency</i>
OPC	<i>Optical Proximity Correction</i>
PDA	<i>Personal Digital Assistant</i>
RMST	<i>Rectilinear Minimum Spanning Tree</i>
RSMT	<i>Rectilinear Steiner Minimal Tree</i>

RST	<i>Rectilinear Steiner Tree</i>
RTL	<i>Register Transfer Level</i>
R&R	<i>Rip-up and Re-route</i>
TCS	<i>Task-based Concurrency Strategy</i>
TOF	<i>Total Overflow</i>
VLSI	<i>Very Large Scale Integration</i>
WL	<i>Wirelength</i>

SUMÁRIO

1	INTRODUÇÃO	16
1.1	Ciclo de Projeto	17
1.2	Roteamento	18
1.3	Objetivos e Organização do Trabalho	19
2	SÍNTESE FÍSICA	21
2.1	Planejamento Topológico	22
2.2	Posicionamento	23
2.3	Roteamento	24
2.4	Outras ferramentas do fluxo de síntese física	28
2.5	Extensões	29
3	ALGORITMOS E TÉCNICAS DE ROTEAMENTO	32
3.1	Busca em largura e Busca em profundidade	32
3.2	Algoritmos de roteamento	33
3.3	Algoritmos para roteamento global	39
3.4	Roteamento Global e Paralelismo	46
4	ESTADO-DA-ARTE	49
4.1	NTHU-Router	50
4.2	FastRoute	54
4.3	XGRouter	57
4.4	NCTU-GR	60
4.5	BFG-R	62
4.6	GT Router	64
4.7	Explorando Paradigma de Computação de Alto Desempenho para Roteamento Global	66
5	METODOLOGIA	70
5.1	Análise inicial do roteador	71
5.2	Levantamento dos resultados para comparação	72
5.3	Análise avançada do roteador	74
5.4	Aplicação de melhorias	76
6	RESULTADOS	79
6.1	Limiar de Congestionamento	79
6.2	Paralelismo	81
6.3	Paralelismo e Limiar de Congestionamento juntos	83

7	CONCLUSÃO	85
	REFERÊNCIAS	87
ANEXO A	EXEMPLOS DE ALTERAÇÕES REALIZADAS COM O OPENMP	92
ANEXO B	EXEMPLO DE ARQUIVOS LEF/DEF	94
ANEXO C	EXEMPLO DE UM ARQUIVO DE ENTRADA PARA ROTEAMENTO GLOBAL	96

1 INTRODUÇÃO

Através da evolução da automação no processo de fabricação de circuitos integrados e do desenvolvimento de ferramentas de EDA (*Electronic Design Automation*) tornou-se possível a criação de sistemas computacionais cada vez mais robustos, influenciando desta forma progressivamente a vida das pessoas.

A integração de milhares de transistores em um *chip* é possível em consequência da miniaturização dos transistores. A evolução dos circuitos está diretamente ligada à Lei de Moore, que define uma estimativa de que a cada 18 a 24 meses o número de transistores por circuito integrado pode ser dobrado com o mesmo custo (MOORE, 2006). A Figura 1 mostra uma projeção da Lei de Moore de 1970 até os anos atuais, apresentando a densidade de transistores nos *chips* de acordo com o passar dos anos, e também fornecendo exemplos de *chips* a cada período de tempo.

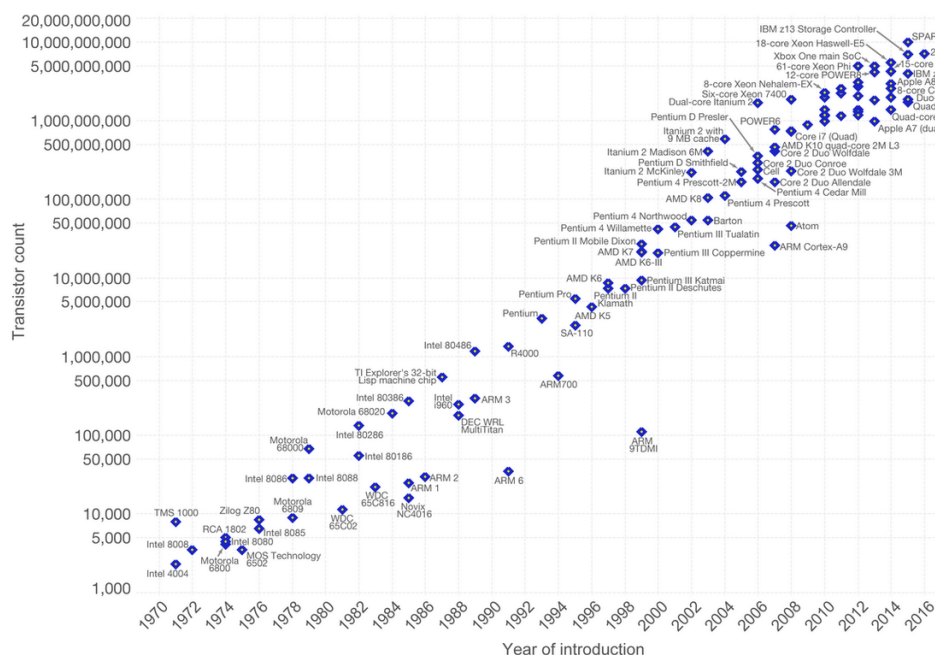


Figura 1: Projeção Lei de Moore 1970 - 2016 (REY, 2017).

A evolução dos circuitos integrados tornou possível a criação de sistemas computacionais cada vez mais eficientes, complexos e robustos, e, consequentemente, diversas

conveniências que acompanham esta evolução foram popularizadas, tais como celulares, sistemas de navegação, PDAs (*Personal Digital Assistant*). Por outro lado, a evolução das tecnologias e a miniaturização dos transistores aumentou a complexidade dos circuitos, aumentando a integração de componentes. Isso implica no aumento da dificuldade de tratar problemas dentro da concepção de circuitos integrados VLSI (*Very Large Scale Integration*).

Dada uma modelagem feita à nível ESL (*Electronic System Level*), a EDA é parte encarregada de automatizar os processos de projeto e teste, se certificando de que as especificações do sistema estão de acordo com o que foi projetado inicialmente. O projeto em nível ESL passa por várias etapas de síntese e verificação, até que, finalmente, o produto seja um circuito integrado que esteja de acordo com as especificações e qualidade desejadas (WANG; CHANG; CHENG, 2009). Grande parte dos problemas relacionados com a concepção de circuitos integrados VLSI, existentes no desenvolvimento de ferramentas de EDA (*Electronic Design Automation*) são NP-completos (WANG; CHANG; CHENG, 2009). Além disso, tratar um grande volume de dados que precisam ser processados em conjunto, torna o problema ainda mais crítico em todas as etapas do ciclo de projeto.

A EDA também abrange outras fases, como o mapeamento tecnológico e o projeto físico propriamente dito (planejamento topológico, posicionamento, roteamento e verificação das regras de projeto). Além disso, ela compreende simulações lógicas e análises de tempo para verificação do projeto antes e depois de passar pela síntese física. Existe também uma grande demanda por ferramentas para testes de manufatura, o que inclui análise de testabilidade, geração automática de padrões de testes (*Automatic Test Pattern Generator* - ATPG), simulação de falhas, projeto para testabilidade (*Design For Testability* - DFT), *Built-In Self-Test* (BIST) e compressão de testes (WANG; CHANG; CHENG, 2009). Em geral, é possível dividir EDA em três categorias maiores, são elas: a automação lógica, a verificação/teste, e a automação do projeto físico.

1.1 Ciclo de Projeto

O projeto de circuitos integrados trata de tarefas complexas, divididas em vários estágios de acordo com a metodologia de projeto adotada. Existem diversas metodologias que surgiram com o intuito de tornar possível a automação do projeto de circuitos VLSI, e estas estão ligadas diretamente a um conjunto de transformações e algoritmos e ao estilo de leiaute, os quais apresentam um ciclo de projeto garantido. Estilo de leiaute, ou estilo de projeto, é a organização geométrica que viabiliza ou provoca o uso de determinada metodologia (JOHANN, 2001). A seguir, são listados os três principais estilos de projeto de circuitos VLSI.

- Projeto *Full-custom*: O projeto *Full-custom* apresenta a liberdade para definição

dos detalhes do leiaute dos transistores e conexões do circuito sem a utilização de padrões.

- Projeto *Macro-cell*: Em um projeto *Macro-cell* o circuito é feito com blocos previamente projetados, os quais recebem o nome de células ou módulos. Neste tipo de estilo é normalmente reaproveitado o que já foi projetado.
- Projeto *Standard-cell*: No projeto *Standard-cell* faz-se a utilização de uma base de células retiradas de uma biblioteca pré projetada. Esta metodologia torna possível um projeto rápido, e torna a automação na parte física simplificada.

No projeto *Standard-cell*, uma das etapas é a síntese física. A síntese física é responsável pela construção do leiaute do circuito integrado. A Síntese Física também é dividida em mais etapas, dentre as três principais etapas desta divisão está o planejamento topológico, onde é realizada a divisão do chip para a colocação de componentes. A segunda etapa maior é o posicionamento, onde é realizado o espalhamento das células lógicas no *chip*, as células são espalhadas globalmente, logo após é realizada uma legalização para que não haja sobreposição de nenhuma célula e então é realizado o posicionamento detalhado para obter um melhor resultado. A terceira maior etapa do fluxo de síntese física é o roteamento dos fios para completar as interconexões entre as células. O produto gerado a partir deste processo é a representação geométrica do *chip*, ou seja, a geração do leiaute final a ser enviado para a fábrica.

1.2 Roteamento

O problema de roteamento existe desde o uso de leiaute das primeiras placas de circuito impresso, e conforme a tecnologia evoluiu trazendo transistores cada vez menores, foi possível aumentar o fator de integração nos circuitos integrados. Desta forma, aumentando o número de funcionalidades disponíveis e, consequentemente, o número de dispositivos em um único projeto. O roteamento é tradicionalmente dividido em duas etapas: O roteamento global e o roteamento detalhado.

De acordo com (TUMELERO; BONTORIN; REIS, 2015), apesar do roteamento ser uma das primeiras áreas de EDA (*Electronic Design Automation*) a ser automatizada, ele continua sendo uma área com quantidade significativa de esforço em pesquisas e desenvolvimento, e isto se deve ao fato de que o roteamento é um problema NP-completo, sem uma solução ótima em tempo polinomial (SZYMANSKI, 1985). Nos tempos atuais, um *chip* chega a possuir milhões ou mesmo bilhões de transistores, o que torna o problema do roteamento de difícil processamento computacional, podendo demandar muito tempo para processar uma solução dependendo do algoritmo que foi aplicado e da solução que é esperada como resultado final (SHYAMALA; LATHA, 2016). Este problema é tratado

na maioria das vezes com heurísticas e algoritmos de busca de menor caminho em grafos se tratando de algoritmos.

Soma-se os fatos do roteamento ser um problema NP-completo e, da quantidade de transistores que um circuito pode chegar a ter nos tempos atuais a partir do crescimento rápido de circuitos lógicos e de capacidade de memória. Isso torna o problema de roteamento ainda mais difícil de ser tratado de uma maneira otimizada. Os algoritmos eficientes para o processo de roteamento global em circuitos VLSI estão em demanda devido a estes fatores (SHYAMALA; LATHA, 2016).

Mesmo nos tempos atuais existindo diversas técnicas sofisticadas para concluir o roteamento, algoritmos antigos de uso geral continuam sendo utilizados dentro do processo de roteamento (UDGIRKAR; INDUMATHI, 2016). Com a evolução da tecnologia, a implementação de algoritmos com o propósito de roteamento pode mudar muito. Porém, existe uma classe de algoritmos de roteamento chamada de algoritmos de roteamento de propósito geral, a qual o principal objetivo é o de encontrar um caminho entre dois pontos, desconsiderando qualquer possível restrição (GONÇALVES; ROSA; SOUZA MARQUES, 2014).

Se tratando de otimização, é possível otimizar o processo de roteamento global de duas formas: 1) no campo de algoritmos, buscando implementar heurísticas melhores, e algoritmos mais eficientes ; e 2) em termos de processamento de CPU (*Central Processing Unit*) ou outras arquiteturas computacionais mais robustas, como processamento distribuído ou clusterização, buscando fazer o melhor uso dos recursos disponíveis pelo *hardware* para cumprir a tarefa do roteamento, como por exemplo utilizando o paralelismo (TUMELERO; BONTORIN; REIS, 2015).

1.3 Objetivos e Organização do Trabalho

O objetivo deste trabalho é identificar possibilidades de otimização de algoritmos de roteamento, e implementar as mesmas no roteador global NTHU-route (CHANG; LEE; WANG, 2008). Para este fim, realizou-se inicialmente um estudo, e foram levantadas possíveis melhorias a serem aplicadas. Dentre estas melhorias, levou-se em consideração a aplicação de limiares nos resultados preliminares em tempo de execução, e a paralelização de blocos de código dentro da estrutura do roteador. Para os estudos e mensuração dos ganhos após aplicação das otimizações, foram utilizados os *benchmarks* do ISPD 2008 (ISPD2018, 2018).

Esta dissertação está organizada da seguinte forma. No Capítulo 2 é abordada a síntese física de projeto, apresentando as três principais etapas da síntese física e as principais extensões utilizadas como entradas em ferramentas *place-and-route* e competições de EDA. O Capítulo 3 aborda os principais algoritmos utilizados no roteamento global, além de técnicas mais avançadas. O Capítulo 4 apresenta roteadores estado-da-arte, e aborda

técnicas utilizadas por estes, assim como os resultados deles quando aplicados nos *benchmarks* dos ISPD's 2007 e 2008. O Capítulo 5 apresenta a metodologia aplicada para a realização do trabalho, o Capítulo 6 apresenta os resultados obtidos nos experimentos, e finalmente, o Capítulo 7 apresenta a conclusão do trabalho, e ideias para trabalhos futuros.

2 SÍNTESE FÍSICA

Neste capítulo será abordada a síntese física de circuitos VLSI e suas principais etapas. Antes do ciclo de projeto entrar na etapa da síntese física, são realizadas as etapas de especificação, projeto funcional e síntese lógica, conforme ilustra a Figura 2. A especificação do projeto é o primeiro passo no processo de projeto do circuito. Nesta etapa são planejadas todas as especificações, é realizada uma representação de alto nível do projeto, levando em consideração as funcionalidades esperadas, tais como desempenho, tecnologia de fabricação e técnicas de projeto. Os resultados desta etapa são as especificações de tamanho, desempenho, consumo de energia e funcionalidades do sistema (SHERWANI, 2012).

A partir da especificação do projeto, é realizada uma modelagem em nível comportamental utilizando descrições ESL (*Electronic System Level*) ou HDL (*Hardware Description Language*) a partir de linguagens específicas, tais como *SystemC*, VHDL, Verilog, ou mesmo C e C++. O objetivo desta etapa é o de descrever o comportamento do *hardware* em um nível de modelagem em que seja possível simular para realizar a verificação do projeto, e também seja possível realizar a tradução para para RTL (*Register Transfer Level*). A descrição RTL determina o fluxo de controle, largura de palavras, alocação de registradores, operações aritméticas e operações lógicas que representem o projeto funcional (WANG; CHANG; CHENG, 2009). A saída do projeto funcional/arquitetural é a descrição RTL, a qual é utilizada na etapa da síntese lógica.

Durante a síntese lógica, as expressões Booleanas da especificação funcional são convertidas em uma representação em forma de circuito, onde são levadas em consideração informações como energia e atrasos. Esta representação possui os elementos do circuito, tais como portas lógicas, transistores, macros, etc.; além das interconexões entre estes elementos. Nesta etapa também acontece o mapeamento tecnológico, que é uma das etapas mais importantes em otimização dependente de tecnologia. O mapeamento tecnológico recebe como entrada uma *netlist* independente de tecnologia já otimizada (portas lógicas simples), e substitui estas portas lógicas por portas lógicas que estão presentes na biblioteca de células que será utilizada pelo projeto, dando como resultado a representação do projeto em forma de circuito, denominada *netlist* (WANG; CHANG; CHENG, 2009).

A *netlist* possui a descrição do circuito após a realização do mapeamento tecnológico, ela especifica interconexões entre as células e provê as informações de cada célula. Estas informações podem conter o tipo de célula, o tamanho e a altura, e as localizações dos pinos de entrada e saída (*I/O pins*). A *netlist* é utilizada como entrada para a etapa seguinte, a síntese física.

A síntese física é parte integrante do fluxo de projeto de circuitos integrados. Ela pode ser dividida em três passos principais: planejamento topológico (*floorplanning*), posicionamento (*placement*) e roteamento (*routing*) (NUNES; REIS, 2013). A Figura 2 apresenta os níveis típicos de projeto de circuitos integrados. Abrindo o leque da síntese física, são apresentadas todas as etapas típicas pelas quais o projeto passa dentro do nível de síntese física, incluindo, além das etapas citadas, o planejamento de energia, e a geração da árvore de *clock*.

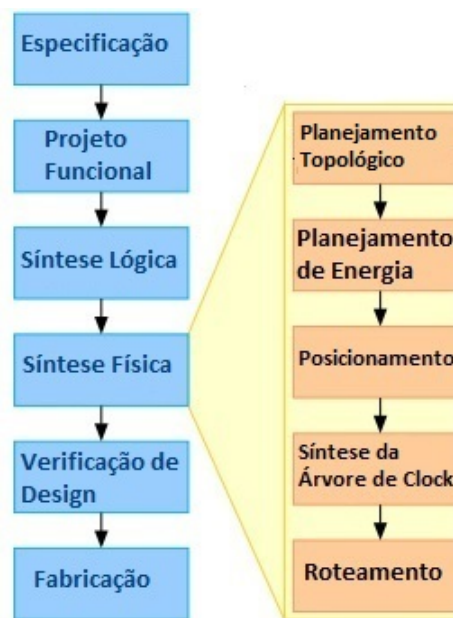


Figura 2: Ciclo típico de projeto de Circuitos Integrados com divisão da parte física. Adaptado de (TUMELERO; BONTORIN; REIS, 2015).

O escopo maior da síntese física é, dada uma relação de componentes e interligações, realizar a geração do leiaute do circuito, passando por todas as fases citadas. O produto esperado é a representação geométrica do mesmo. Neste capítulo, os passos de planejamento topológico, posicionamento e roteamento serão abordados, assim como as principais etapas e sub-divisões deles.

2.1 Planejamento Topológico

O planejamento topológico é considerado o primeiro estágio do projeto físico de circuitos VLSI. Neste estágio, a partir de um conjunto de blocos rígidos (os quais os tamanhos não podem ser ajustados), blocos maleáveis (os quais os tamanhos podem ser

ajustados), e uma *netlist*, são determinadas as formas dos blocos, e então os blocos são posicionados no *chip*. Nesta etapa já é possível ter um retorno sobre a necessidade de possíveis alterações no *design* (WANG; CHANG; CHENG, 2009).

Um *floorplan* F é uma dissecção de uma região retilínea R em um conjunto de retângulos, onde cada um destes retângulos representa um módulo (SARRAFZADEH, 1993). O *floorplanning* pode ser dividido em duas categorias: 1. *floorplans* fatiáveis e 2. *floorplans* não-fatiáveis. O fatiável pode ser obtido cortando horizontalmente ou verticalmente os blocos do *chip*, já o não-fatiável não permite estes cortes (WANG; CHANG; CHENG, 2009). Na Figura 3 (a) é apresentado o *floorplan* sem a possibilidade de ser fatiado, e a Figura 3 (b) mostra o mesmo *floorplan*, com a diferença que o bloco 2 foi movido para o topo do bloco 1, tornando possível que ele seja fatiado verticalmente. O próximo passo dado na síntese física após o planejamento topológico é o posicionamento.

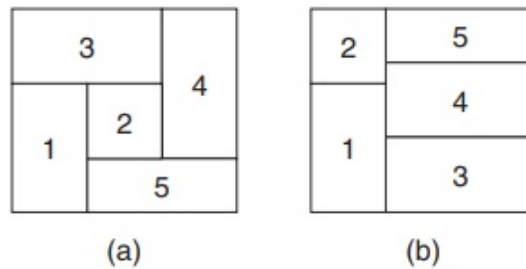


Figura 3: Exemplos de *floorplan* não-fatiável (a) e fatiável (b) (WANG; CHANG; CHENG, 2009).

2.2 Posicionamento

Assim como alguns outros problemas de EDA, o posicionamento das células nos circuitos VLSI é NP-completo, e a implementação de algoritmos que produzam boas soluções em um circuito com um número muito grande de células, e em um tempo aceitável, também é uma tarefa difícil (CHEN; ZHU, 2012). O problema de posicionamento das células em circuitos VLSI é formulado a partir de uma *netlist*, tendo como objetivo determinar a localização de cada célula, de forma que não haja sobreposição entre as células, e de forma que as métricas de custo sejam otimizadas.

Existem diversos estilos de projeto de circuitos, tais como *macro cell*, *standard cell* e *gate array*. Devido às diferentes células e diferentes estilos de projeto, que podem levar à diferentes limitações, existem muitas variações deste problema (SHERWANI, 2012). Visto que o estilo *standard cell* não demanda muito tempo para implementar o projeto e apresenta bons resultados de atrasos e potência (intermediários entre o *Full Custom* e *Macro cell*), atualmente este estilo de projeto é o mais popular para ASICs (*Application-Specific Integrated Circuit*), e a maior parte dos algoritmos com o propósito de cumprir a

tarefa do posicionamento apresentam soluções para este estilo (SHERWANI, 2012).

Por diversas razões, o posicionamento das células pode ser visto como uma parte crítica do projeto de circuitos integrados. É possível dizer que, nesta etapa, é determinado o desempenho do circuito. O posicionamento exerce forte influência no roteamento, pois afeta diretamente o comprimento dos fios que serão conectados na fase posterior, na distribuição do aquecimento que ocorrerá em certas regiões do chip após concluído, e também no consumo de energia (CHEN; ZHU, 2012).

Com a conclusão do posicionamento, é definida a posição de cada elemento do circuito (células, pinos, etc), e também é determinado o tamanho estimado de fios que serão utilizados para interconectar elementos. O próximo passo que deverá ser dado após este processo é o roteamento dos fios para realizar estas interconexões.

2.3 Roteamento

O roteamento é uma etapa importante no ciclo de projeto físico de circuitos integrados, pois é através desta etapa que as conexões dos fios de mesmo potencial são realizadas entre os componentes do circuito. O aumento dos transistores em um *chip* passa a casa dos milhões/bilhões faz aumentar cada vez mais os desafios na área de *Electronic Design Automation*. Com a redução nas dimensões dos transistores e o número de dispositivos (densidade) nos circuitos aumentando, o problema de roteamento tem se tornado cada vez mais desafiador (OZDAL, 2009).

Após o processo do posicionamento ser finalizado, é gerado como resultado a localização exata dos blocos e pinos fixados, ou seja, é disposto um leiaute pronto, porém sem possuir as conexões entre os componentes ainda estabelecidas. Além do leiaute, recebemos a informação da *netlist*, a qual contém a lista de conexões que indica quais pinos devem ser conectados formando um conjunto de redes (*nets*).

O objetivo ao receber estas informações é realizar as conexões entre os componentes de forma a estabelecer ligações entre componentes do circuito que utilizam mesmo potencial, respeitando as regras de projeto e restrições impostas pela tecnologia, e buscando melhorar métricas de custo, tais como a redução no congestionamento de rotas. Dada a complexidade do problema de roteamento, este normalmente é dividido em duas etapas para facilitar o tratamento do problema, estas etapas são o roteamento global e o roteamento detalhado. O roteamento global é encarregado de aproximar os caminhos das interconexões, e o roteamento detalhado é encarregado de especificar as conexões físicas de fato, tratando das rotas, vias e camadas de metal para realizar as conexões de forma a respeitar o roteamento global (SIDDIQI; SAIT, 2017).

Na Figura 4 é apresentado o processo de roteamento descrito acima, onde (a) apresenta uma solução recebida da etapa de um posicionamento, onde os retângulos são as células ou blocos posicionados dentro do *chip*; (b) apresenta o processo do roteamento global,

onde as linhas tracejadas indicam o caminho por onde as redes deverão passar durante o processo de roteamento detalhado, e, finalmente, (c) apresenta o processo do roteamento detalhado, onde os fios (linhas) são traçados da forma final em que deverão conectar as redes.

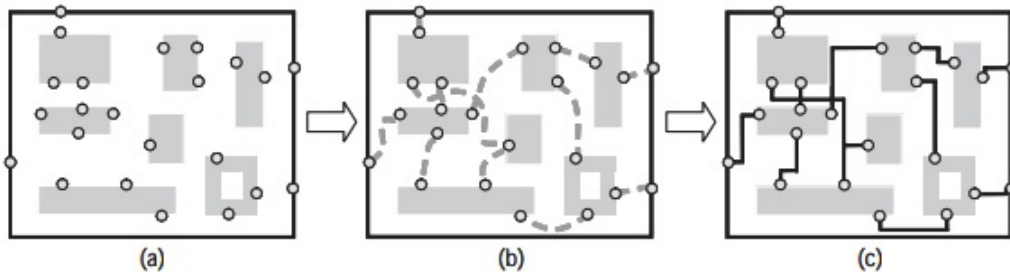


Figura 4: O processo de roteamento; (a) Resultado recebido de um posicionamento; (b) roteamento global; e (c) roteamento detalhado. Retirado de (WANG; CHANG; CHENG, 2009).

Roteamento Global

De acordo com (JOHANN, 2001), o roteamento global pode ser visto como um problema de otimização combinatória. O roteamento global é geralmente formulado como um problema de grafos, e resolvido com algoritmos aplicáveis em grafos. O problema é normalmente formulado como um grafo G , com um conjunto de vértices V e um conjunto de arestas E , onde cada vértice $vi \in V$ corresponde a uma região retangular ou célula do chip, e cada aresta $aij \in E$ corresponde a uma fronteira entre os vértices adjacentes $vievj$. Além disto, existe um conjunto de *nets* N , no qual cada *net* $ni \in N$ é composta por um conjunto de pinos Pi , com cada um destes pinos correspondendo a um vértice vi (MOFFITT; ROY; MARKOV, 2008).

Na Figura 5 é apresentado um circuito dividido em *Global bins* (retângulos no chip), na Figura 5 (a) são apresentados as arestas que ficam entre os *global bins* e as células dentro dos *global bins*, e na Figura 5 (b) é apresentado o grafo correspondente.

Ao construir um roteador global é preciso levar em consideração normalmente três métricas (MOFFITT; ROY; MARKOV, 2008):

1. congestionamento (congestionamento): situação causada pelo excesso de rotas em uma mesma região, acontece quando existem muitos fios passando perto um do outro e isto pode gerar diversos problemas, tais como capacitância indesejada;
2. comprimento total dos fios (*total wirelength*): O comprimento total dos fios é a soma dos comprimentos de cada uma das redes;
3. o tempo para chegar a uma boa solução (*runtime*).

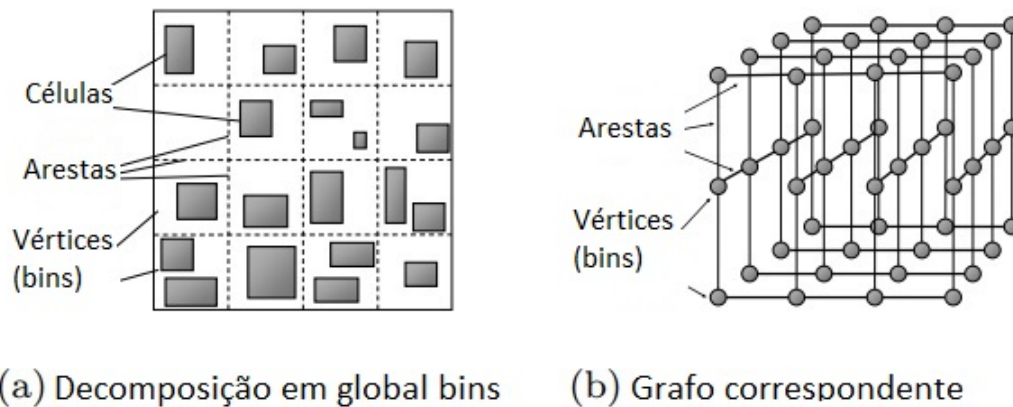


Figura 5: Decomposição do chip em global bins, e o grafo correspondente que formula o problema do roteamento global. Adaptado de (MOFFITT; ROY; MARKOV, 2008).

Além disto, existem três objetivos específicos que devem ser cumpridos por um roteador global (JOHANN, 2001):

1. A distribuição das conexões no espaço com o gerenciamento do congestionamento;
2. A definição da forma das redes maiores com múltiplos pinos;
3. A facilitação e orientação sobre caminhos a serem seguidos no roteamento detalhado.

Segundo (SHERWANI, 2012), o roteamento global pode ser dividido em três etapas:

1. Definição das regiões: Todo o espaço de roteamento é dividido em regiões menores de roteamento. Isto inclui espaço entre os blocos e em cima dos blocos. Entre os blocos existem dois tipos de regiões de roteamento: de canal e *2D-switchboxes*, e acima dos blocos, onde todo o espaço de roteamento é livre, particionando este espaço também em regiões menores, chamadas *3D-switchboxes*. Para cada região definida é atribuída uma capacidade, que é a quantidade de redes que poderão passar por esta região;
2. Assinalamento das regiões: nesta etapa, são identificadas as sequências das regiões por onde uma rede irá passar (ser roteada). O congestionamento de cada região deve ser levado em consideração;
3. Assinalamento dos pinos: Depois de concluído o assinalamento das regiões, é assinalado um pino nas fronteiras de cada região, o que permite que as regiões sejam independentes durante o processo de roteamento.

Os resultados do roteamento global são as regiões pelas quais uma rede r deve ser roteada, ou seja, é esperada uma orientação a ser seguida pelo roteamento detalhado para a geração do leiaute com as conexões de cada rede realizadas. Para isto, são levadas em consideração as métricas de custo já citadas para tornar o circuito mais próximo do otimizado, buscando evitar que a rota da(s) rede(s) passe(m) por regiões congestionadas. Em caso de haver regiões muito congestionadas, algumas técnicas podem ser aplicadas para tentar diminuir o congestionamento, tais como a técnica de desfazer e re-rotear, a qual é uma técnica algorítmica aplicada para rotear novamente as redes que estão congestionadas ou causando congestionamento para outras redes. Os algoritmos serão discutidos no Capítulo 3.

Roteamento Detalhado

Depois de uma rota genérica ser determinada para cada rede durante o roteamento global, o próximo passo é a execução do roteamento detalhado para encontrar as rotas exatas para todas as redes (OZDAL, 2009). Dada uma solução de roteamento global, o roteamento detalhado decide as conexões físicas exatas das redes (*nets*), incluindo as camadas (*layers*) e conexões entre as mesmas utilizando as *vias*.

Em resumo, o roteador detalhado é encarregado de especificar as rotas de cada conexão, incluindo os seus materiais, posições e dimensões (JOHANN, 2001). A Figura 6 apresenta um exemplo de roteamento detalhado, onde são apresentadas ligações entre células e pinos através dos fios (que estão ilustrados em rosa e ciano, indo na vertical e horizontal), em diferentes camadas de metal.

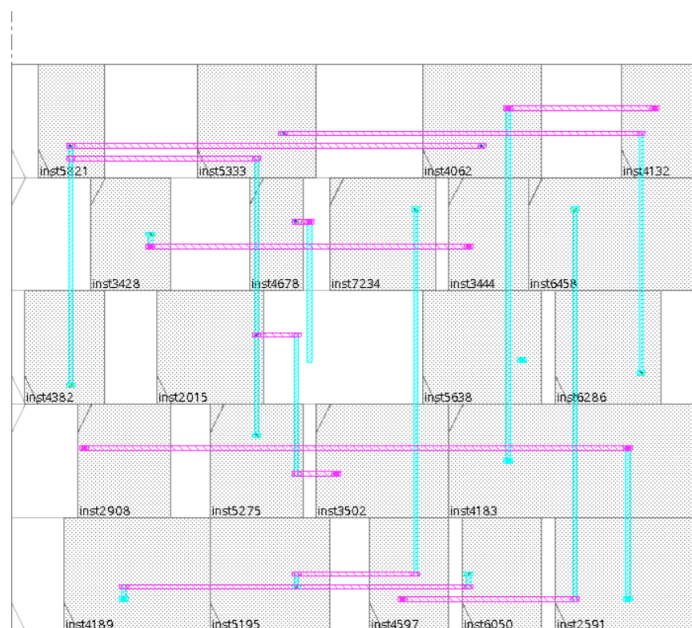


Figura 6: Roteamento detalhado lido a partir da ferramenta INNOVUS. Adaptado de (ISPD2018, 2018).

Em tecnologias mais antigas, quando o número de camadas existentes em um *chip* não passavam de duas ou três, o roteamento de canal era muito utilizado. Neste tipo de roteamento, os fios eram roteados no espaço livre entre as células lógicas, necessitando assim de área extra no *chip* para o roteamento das redes.

Normalmente, os tamanhos das grades geradas durante o roteamento detalhado são muito maiores do que as geradas no roteamento global, e o espaço para as redes individuais é muito maior. As regras de projeto para fabricação fazem com que o roteamento detalhado seja ainda mais complexo. Diferente do roteamento global, no caso do roteamento detalhado, é difícil a utilização de algoritmos sofisticados para tratar todas as redes simultaneamente de forma a aproximar-se de uma solução boa. No caso do roteamento detalhado, é realizado o roteamento de uma rede por vez, e a solução para cada uma destas redes é restringida às regiões que foram determinadas no roteamento global (OZDAL, 2009).

O roteamento detalhado é aplicado em partições menores do *chip*, as quais são definidas no roteamento global ou, em alguns casos, após o posicionamento das células. O principal objetivo deste roteamento é realizar as conexões das rotas sem permitir que fios se cruzem em uma mesma camada e melhorar métricas estipuladas, tais como diminuição dos desvios nas rotas e diminuição das redes.

2.4 Outras ferramentas do fluxo de síntese física

Após a etapa de roteamento ser finalizada, ainda são utilizadas ferramentas de verificação física, as quais buscam cobrir uma verificação de regras de projeto, verificação de desempenho e verificação de confiabilidade. Estas ferramentas são utilizadas para ter certeza de que os padrões geométricos definidos foram cumpridos, o tempo de propagação dentro do circuito e as regras de eletricidade satisfazem as especificações do projeto proposto.

A síntese de *clock* e *power/ground* são também diferentes da aplicada nas redes de sinais normais. Roteadores de propósito geral não conseguem lidar com os requisitos exigidos em redes de *clock* e *power/ground*. Em roteadores de *clock* por exemplo, o *clock skew*, que é a diferença máxima no tempo de chegada do sinal nos componentes do circuito, deve ser otimizado para evitar que um sinal chegue atrasado ou mais rápido em determinado componente, o que pode comprometer a confiança ou o funcionamento do circuito. No caso das redes *power/ground*, estas são normalmente mais largas do que as redes de sinal, pois conduzem quantidades muito maiores de corrente, então o tamanho das redes de *power/ground* devem ser considerados para o tamanho total da área utilizada.

2.5 Extensões

As extensões *place-and-route* são as entradas de dados utilizadas em diferentes etapas do ciclo de projeto VLSI. Aqui são apresentadas três principais tipos de extensão, incluindo o *Bookshelf*, LEF/DEF e a entrada de arquivos auxiliares para roteamento global.

Bookshelf

O *Bookshelf* é uma extensão muito utilizada pela academia por prover uma melhoria na comunicação e competição entre grupos de pesquisa, além da adoção e reconhecimento mais rápido de avanços em pesquisas realizadas na academia pela indústria (CALDWELL ANDREW; KAHNG, 2017). Este formato é muito utilizado para realização de posicionamento de células. Após as células estarem posicionadas, as informações de localizações destas são salvas em um arquivo com extensão .pl, e as conexões são especificadas em um arquivo .nets. Esta e as demais extensões serão explicadas a seguir.

Benchmarks Bookshelf normalmente possuem seis arquivos, sendo estes:

- .aux: O arquivo com esta extensão possui as informações dos nomes dos demais arquivos (nets, nodes, etc.).
- .nets: O arquivo .nets possui informações sobre as conexões das redes do circuito, dizendo de onde saem e para onde vão, podendo as conexões serem entre pinos/células ou células/células.
- .nodes: Descreve a lista de objetos e, para cada objeto, descreve sua altura e largura.
- .slc (*Standard cell layout*): Dita a quantidade de linhas do circuito, para cada linha sua posição e altura, além do número de divisões (*sites*) de cada linha, e a largura de cada um destes *sites*.
- .pl (*Placement and orientations*): Possui informações sobre posição e orientação de cada célula e cada pino. Antes de passar por um posicionamento a posição de cada célula é informada como 0.
- .wts: Lista os pinos e células, e o peso que estes possuem relacionados à rede a qual pertencem.

LEF/DEF

O arquivo com extensão LEF (*Library Exchange Format*) é utilizado para descrever uma biblioteca de células padrão. Esta extensão inclui regras de projeto e informações abstratas das células (sem informação adicional do *netlist* interno destas).

A extensão LEF contém uma biblioteca de informações para uma classe de *designs*. Estas incluem descrições de camadas de metal, vias, tipos de *sites* para posicionamento

e definições das células. Um único arquivo LEF pode conter todas as informações sobre a biblioteca. Porém, isto torna o arquivo de difícil manuseio. A melhor abordagem é dividir as informações em dois arquivos, sendo estes um arquivo de tecnologia que contém todas informações sobre a tecnologia utilizada (regras de projeto, definições das vias, capacitância dos fios, etc.) e um arquivo de células que contém as informações sobre as células do circuito (descrições, dimensões, leiaute dos pinos, bloqueios, etc) (SYSTEMS, 2009).

A extensão DEF (*Design Exchange Format*) contém as informações do leiaute físico do circuito e da *netlist*. Este formato é utilizado em conjunto com o LEF para obter a completa representação do leiaute físico do circuito enquanto este é projetado. A extensão dá informações sobre posicionamento, localizações de pinos, restrições de metal e orientação dos objetos.

O arquivo DEF normalmente é o resultado de ferramentas de síntese física (*place-and-route tools*), e é utilizado posteriormente em ferramentas de análises. As informações do *netlist* podem estar no próprio arquivo DEF ou em arquivos separados como por exemplo em um Verilog, ou mesmo em outro arquivo DEF à parte (SYSTEMS, 2009).

Extensão Global Routing

Em competições como o ISPD (*International Symposium on Physical Design* 2007 e ISPD 2008 (ISPD2008, 2017) foram utilizados como entrada para os *benchmarks* apenas um arquivo de texto que é uma variação do formato *Labyrinth* (ISPD2008, 2017). Neste arquivo são especificadas nas linhas iniciais, as grades com suas posições x e y , o número de camadas de metal, a capacidade vertical, a capacidade horizontal, o espaçamento mínimo, os espaçamento de vias e o espaçamento das colunas (ISPD2008, 2017). Também são especificadas para cada rede o número da rede, o nome da rede, o número de pinos o espaçamento mínimo, posição x e y e a camada da ligação, e a estrutura das redes descrita se repete para o número de redes existentes.

3 ALGORITMOS E TÉCNICAS DE ROTEAMENTO

Este capítulo apresenta algoritmos e técnicas utilizadas para realizar o roteamento de circuitos em roteadores globais estado-da-arte acadêmicos. De acordo com (JOHANN, 2001), podemos classificar os algoritmos de roteamento pelo modo com que tratam a solução aplicada para rotear as diversas redes que precisam ser conectadas no mesmo espaço. Desta forma, os algoritmos podem ser classificados como **sequenciais** (realizam as conexões uma a uma), **paralelos/concorrentes** (consideram todas as conexões necessárias a serem realizadas) e **iterativos** (solucionam e depois refinam a solução). Neste capítulo soluções algorítmicas para o problema de roteamento serão apresentadas com o foco em algoritmos de roteamento genéricos e com aplicação na etapa de roteamento global.

3.1 Busca em largura e Busca em profundidade

Os métodos mais utilizados para percorrer grafos são a busca em largura (*Breadth-first search*) e a busca em profundidade (*Depth-first search*), os quais formam a base de diversos algoritmos aplicáveis em grafos (WANG; CHANG; CHENG, 2009). A Figura 7 ilustra a diferença na ordem em que os vértices são percorridos utilizando o método BFS e DFS. Na Figura 7 (a) é apresentada a ordem em que os vértices são percorridos no método BFS, e na Figura 7 (b) é apresentada a ordem em que os vértices são percorridos no método DFS.

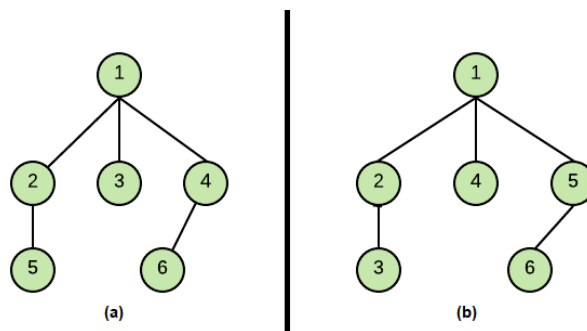


Figura 7: (a) ordem que os vértices são percorridos no método BFS, (b) ordem que os vértices são percorridos no método DFS.

Na Busca em largura (BFS), dado um vértice v , a pesquisa visita todos os vértices que são adjacentes à este, e então, continua sua execução visitando os vértices adjacentes aos vértices que eram adjacentes ao vértice v , até que tenha visitado todos os vértices possíveis dado o vértice v como origem.

A pesquisa BFS utiliza uma fila, a qual guarda os índices dos vértices cujos vértices adjacentes ainda não foram visitados. Esta técnica pode ser implementada com um procedimento do tipo FIFO (*First In First Out*) (JOHANN, 2001). Os caminhos do mesmo tamanho são levados em consideração ao mesmo tempo, e a técnica FIFO garante que cada vértice seja visitado uma vez, e seus vértices adjacentes sejam visitados na sequência, explorando a largura (WANG; CHANG; CHENG, 2009). Um exemplo de algoritmo aplicado ao roteamento que utiliza a técnica de busca em largura é o algoritmo de Lee (LEE, 1961).

Na técnica de busca em profundidade (DFS), é dado um vértice fonte v , o grafo é explorado pesquisando o mais longe possível até encontrar as folhas do grafo, e então retorna em um processo de *backtracing* e explora os vértices anteriores para ter certeza de que não existem vértices inexplorados (WANG; CHANG; CHENG, 2009). Um exemplo de algoritmo utilizado para roteamento que utiliza busca em profundidade (em conjunto com busca em largura) é o algoritmo de (SOUKUP, 1978).

3.2 Algoritmos de roteamento

Existem dois tipos principais de algoritmos de roteamento: os baseados em pesquisa de caminhos e os baseados em geometria. Os algoritmos de pesquisa de caminhos baseiam-se na pesquisa de caminhos em grafos, denominados na área de roteamento de *maze-routers*. Os algoritmos baseados em geometria utilizam as posições reais dos elementos, e não requerem um espaço discreto, porém a liberdade que isto gera pode afetar as qualidades do resultado (JOHANN, 2001). Ambos os algoritmos genéricos podem ser aplicados tanto no roteamento global quanto no detalhado, desde que a estrutura de roteamento seja simplificada para isto.

Algoritmos de labirinto (*Maze Routing*)

O primeiro algoritmo de busca de caminho entre dois pontos apresentado e publicado foi o algoritmo de Lee, apresentado em (LEE, 1961). É, possivelmente, o algoritmo mais utilizado para encontrar um caminho entre dois pontos, e garante encontrar uma solução ótima, se esta existir. A técnica utilizada na fase de exploração deste algoritmo é pesquisa em largura (BFS).

Algoritmos baseados em busca em labirinto utilizam força bruta. Nesta abordagem, são procurados os menores caminhos entre os dois pontos origem (*source*) e destino (*target*), onde são aplicadas duas fases para a busca deste caminho, a dissipação seguida da

retração, onde "ondas" são propagadas partindo do nodo origem até encontrarem o nodo destino. Quando o nodo destino é encontrado, acontece então a fase de retração, onde o menor caminho é retornado do nodo destino até o nodo origem. O ponto de origem da onda é marcado como S, e cada ponto visitado posteriormente (vizinhos) é marcado com o próximo número em sequência. O processo de visitar os vizinhos e marcar com números continua até que o ponto destino seja encontrado, ou as tentativas atinjam um limite. A Figura 8 apresenta uma ilustração do funcionamento do algoritmo de Lee, onde na Figura 8 (a) acontece a fase de propagação da onda, e na Figura 8 (b) é apresentada a fase de retração. Na figura, os quadrados maiores pintados são obstáculos, S é o nodo origem, e T é o nodo destino (WANG; CHANG; CHENG, 2009).

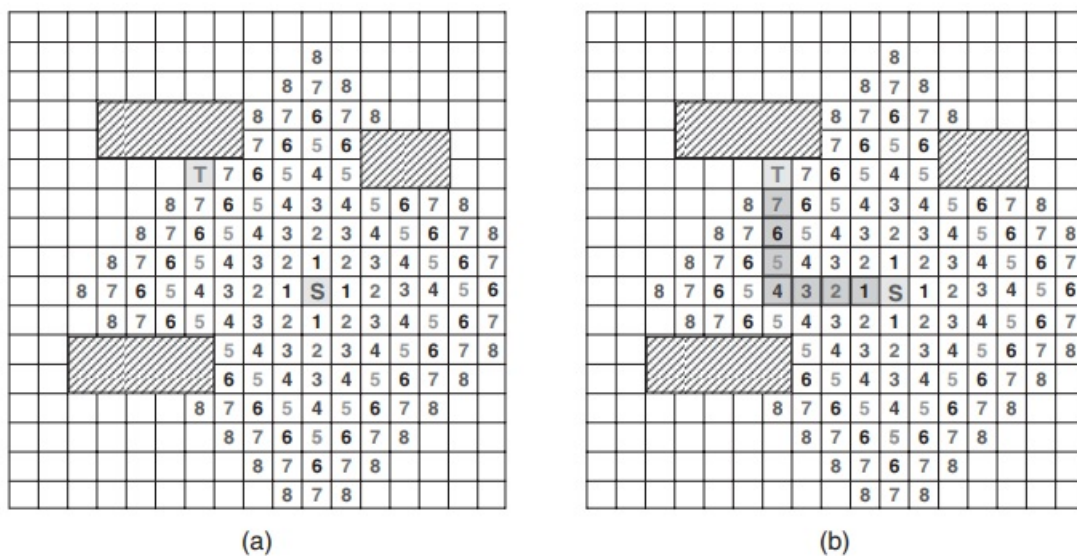


Figura 8: (a) Fase de propagação da onda. (b) Fase de retração. Os quadrados maiores pintados são obstáculos, S é o nodo origem e T o nodo destino (WANG; CHANG; CHENG, 2009).

O Algoritmo 3.2 ilustra o algoritmo de Lee (LEE, 1961). No algoritmo, são instanciados o ponto de origem (origem) e o ponto de destino (destino), um *array* de pontos para visitar B e um *array* do caminho percorrido para chegar do ponto de origem até o ponto de destino L. O *array* L é utilizado na fase de retração para formar um caminho C (saída do algoritmo). Existem duas listas encadeadas para manter um histórico da onda de propagação *listapropagacao* e dos vizinhos dos nós *listavizinhos*.

O algoritmo de Lee também é chamado de **busca cega**, pois ele procura pela rota sem prioridades. Ele garante encontrar um caminho se ele existir, porém possui algumas limitações, tais como a grande demanda de tempo para encontrar uma solução final e a quantidade de memória utilizada no processo. Dadas estas limitações, foram criadas versões melhoradas do algoritmo, tais como o algoritmo A*. A complexidade de tempo e espaço deste algoritmo é $O(mn)$, onde m e n são os números das células horizontais e verticais da grade.

Algoritmo 1: ALGORITMO DE LEE - MAZE ROUTING, ADAPTADO DE (SHERWANI, 2012).

Entrada: $B, origem, destino$

Saída: Caminho

```

1  início
2      listapropagacao  $\leftarrow$  origem;
3      listavizinhos  $\leftarrow$  null;
4      temp  $\leftarrow$  1;
5      caminhoexiste  $\leftarrow$  FALSE;
6      enquanto (listapropagacao  $\neq$  null) faça
7          para cada vertice  $v_i$  em listapropagacao faça
8              para cada vertice  $v_j$  vizinho de  $v_i$  faça
9                  se  $B[v_j] \neq \text{BLOQUEADO}$  então
10                      $L[v_j] \leftarrow$  temp; listavizinhos.INSERT( $v_j$ );
11                     se  $v_j = \text{destino}$  então
12                         caminhoexiste  $\leftarrow$  TRUE; exitwhile;
13                     fim
14                 fim
15             fim
16         fim
17         temp  $\leftarrow$  temp + 1;
18         listapropagacao  $\leftarrow$  listavizinhos;
19         listavizinhos  $\leftarrow$  null;
20     fim
21     se caminhoexiste então
22         RETROCEDER( $L, \text{Caminho}$ );
23     fim
24     senão
25         return FALSE;
26     fim
27 fim
28
```

Algoritmo A*

O algoritmo A* foi proposto por (HART; NILSSON; RAPHAEL, 1968). Ele é um algoritmo de pesquisa em grafos que utiliza uma pesquisa heurística, significando que faz o uso de informações adicionais para tentar descobrir quais os nodos podem levar à uma melhor solução final, e em um tempo otimizado. Também é chamado de **primeiro melhor caminho**, pois a cada iteração ele escolhe as rotas que indicam ser o melhor caminho do nodo origem até o nodo destino. A complexidade de tempo e espaço deste algoritmo depende da heurística aplicada.

O algoritmo utiliza a função $f(x) = g(x) + b(x)$, onde $g(x)$ é o custo do nodo origem até o nodo atual x enquanto percorrendo o caminho, e $b(x)$ é o custo estimado do nodo atual x até o nodo destino, sendo a informação de $b(x)$ cedida juntamente com o grafo onde a solução será trabalhada. Desta forma, o algoritmo seleciona sempre o nodo com o melhor custo estimado para se propagar. Quanto menor o $f(x)$, maior as chances de propagação. O algoritmo A* também pode ser classificado como um algoritmo *maze routing*, porém ele utiliza técnicas diferentes para atingir o objetivo do roteamento. A Figura 9 ilustra o funcionamento do algoritmo A*, onde existe um obstáculo ilustrado como uma linha, o ponto de origem é dado como S, e o ponto de destino é dado como T, os círculos azuis são os caminhos que já foram percorridos, e os círculos brancos são caminhos que ainda não foram percorridos.

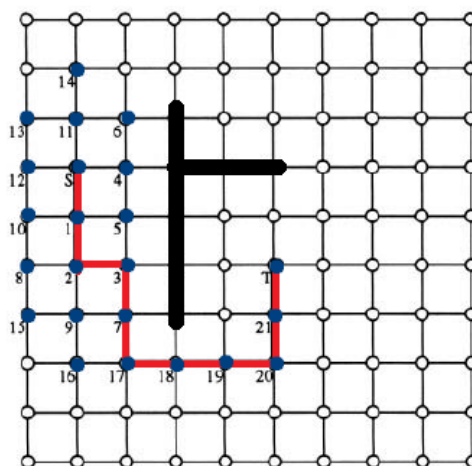


Figura 9: A*: os pontos azuis ilustram a propagação a partir do ponto de origem, a linha preta grossa indica um obstáculo, e a linha vermelha é o caminho traçado após o encontro dos nodos. Adaptado de ((SHERWANI, 2012)).

Algoritmo *Line-search*

Os algoritmos baseados em geometria traçam rotas de uma fonte S até um destino T . Basicamente, estes algoritmos percorrem linhas e caso encontrem obstáculos procuram uma saída alternativa e traçam novas linhas até que a linha do nodo origem S encontre a

linha do nodo destino T , traçando assim uma rota a partir do ponto de encontro entre os dois nodos.

Para lidar com as limitações do algoritmo de Lee, Mikami (MIKAMI, 1968) propôs um algoritmo denominado *Line-search algorithm*. Ao contrário do algoritmo de Lee, que utiliza a técnica de pesquisa em largura, o algoritmo de *line search* utiliza a pesquisa em profundidade (DFS) e é um algoritmo baseado em geometria. O algoritmo *line-search* é apresentado no Algoritmo 3.2 (SHERWANI, 2012).

A partir dos nodos origem e destino são criados quatro segmentos, dois verticais e dois horizontais, passando pelos nodos que são tidos como nodos base inicialmente. Os segmentos então são expandidos até que encontrem obstáculos ou os limites dos caminhos no espaço utilizado para a solução. Então, cada ponto criado no caminho dos segmentos são iterativamente atribuídos como pontos base, e é gerado um novo segmento através de cada um dos novos pontos base. Isto acontece até que um segmento gerado pelo nodo origem se encontre com um segmento gerado pelo nodo destino. Neste momento, uma conexão é encontrada a partir da intersecção dos dois segmentos.

A maior diferença entre o algoritmo *maze routing* e os baseados em geometria é que, diferente do algoritmo de Lee que faz o uso do segmento de grade para a expansão do caminho de busca, os algoritmos baseados em geometria utilizam os segmentos de linhas, de forma a diminuir a complexidade de busca e memória utilizada no processo, porém com a possibilidade de ocorrerem perdas no resultado. A Figura 10 ilustra o algoritmo *line-search* de (MIKAMI, 1968). Na Figura 10 é apresentado o nodo de origem S , e o nodo de destino T , os obstáculos são representados como blocos pintados, e o caminho dos nodos até se encontrarem é ilustrado como asteriscos nos fios.

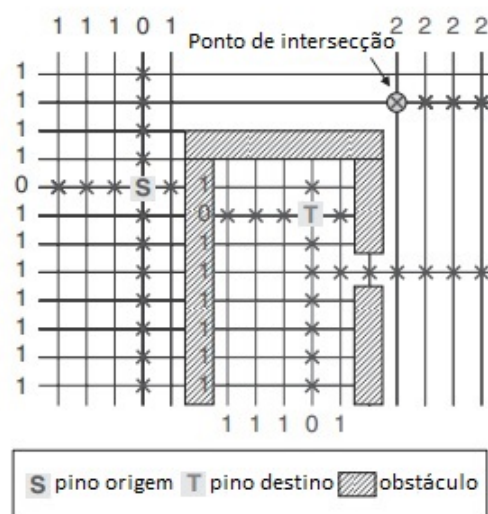


Figura 10: Ilustração do algoritmo *line-search* de (MIKAMI, 1968), nodos se expandem em linhas retas até encontrar obstáculos. Adaptado de (WANG; CHANG; CHENG, 2009).

Assim como o algoritmo de Lee, o *line-search* garante encontrar um caminho entre os pontos se ele existir e é melhor se comparados os tempos de execução e memória utilizada, possuindo complexidade de tempo e espaço $O(L)$, onde L é o número de segmento de linhas gerado pelo algoritmo. Como limitações para este algoritmo, o caminho encontrado pode não ser o mais curto ou melhor possível na solução.

Algoritmo 2: ALGORITMO LINE-SEARCH, ADAPTADO DE (SHERWANI, 2012).

Entrada: *origem, destino*

Saída: *Caminho*

```

1  início
2  neworiginlist ← segmentosgeradospororigem;
3  newdestinlist ← segmentosgeradospordestino;
4  enquanto (neworiginlist ≠ null e newdestinlist ≠ null) faça
5      originlist ← neworiginlist;
6      destinlist ← newdestinlist;
7      para cada segmento de linha  $l_i$  em originlist faça
8          para cada segmento de linha  $l_j$  em destinlist faça
9              se INTERSECT( $l_i, l_j$ ) = TRUE então
10                 caminho.existe ← TRUE;
11                 Caminho ← destinlist;
12                 sai do loop enquanto;
13             fim
14         fim
15     fim
16     neworiginlist ← null;
17     para cada segmento de linha  $l_i$  em originlist faça
18         para cada ponto de escape  $e$  em  $l_i$  faça
19             GERASEGMENTO( $l_k, e$ ); INSERE( $l_k, neworiginlist$ );
20         fim
21     fim
22     newdestinlist ← null;
23     para cada segmento de linha  $l_i$  em destinlist faça
24         para cada ponto de escape  $e$  em  $l_i$  faça
25             GERASEGMENTO( $l_k, e$ ); INSERE( $l_k, newdestinlist$ );
26         fim
27     fim
28 fim
29 se caminho.existe = TRUE então
30     RETROCEDER(Caminho);
31 fim
32 senão
33     return FALSE;
34 fim
35 fim

```

3.3 Algoritmos para roteamento global

Os algoritmos de roteamento podem ser classificados pela forma como tratam o problema, podendo ser de forma sequencial ou de forma concorrente. Ambas as formas serão explicadas a seguir.

Abordagem sequencial

As redes são ordenadas, e de forma sequencial são roteadas uma a uma na ordem pré-estabelecida. Para estabelecer a ordem com a qual as redes ficarão ordenadas, normalmente leva-se em consideração métricas como número de terminais e sua criticidade. De acordo com (WANG; CHANG; CHENG, 2009), a abordagem sequencial normalmente leva à uma solução ruim, por exemplo, uma rede que foi roteada pode bloquear outra rede ou até mais de uma rede, e a solução então fica dependendo da forma com que as redes forem ordenadas.

Nos dias atuais, as abordagens sequenciais utilizam heurísticas para a ordenação das redes, e, em conjunto com a técnica de desfazer e re-rotear, é possível reformular a solução do roteamento, buscando desbloquear caminhos que foram bloqueados por uma má escolha de ordenação das redes, por exemplo.

A técnica de desfazer e re-rotear consiste de dois passos: (1) identificar as regiões congestionadas e desfazer algumas das redes que já foram roteadas naquela região, e (2) rotear as conexões bloqueadas e re-rotear as redes que foram desfeitas. Este processo é então repetido iterativamente até que todas as redes sejam roteadas, ou que alguma outra condição de parada seja satisfeita.

O maior problema da abordagem sequencial é a ordenação das redes. Quando uma ordem é pré-estabelecida, caso bloqueios sejam estabelecidos nas redes iniciais, torna-se mais difícil efetuar o roteamento das redes posteriores. Outro problema da abordagem sequencial é que caso uma solução não tenha sido encontrada, não é possível saber ao certo se o motivo foi o de não existir uma solução ou se a ordenação das redes foi ruim. Além disto, caso uma solução seja encontrada, não sabemos ao certo o quão longe da solução ótima podemos estar (WANG; CHANG; CHENG, 2009).

Abordagem concorrente

Ao contrário da abordagem sequencial, a abordagem concorrente consegue responder se há uma solução, e o quão perto da solução ótima a solução atual se encontra. Uma possível formulação para o problema do roteamento global utilizando abordagem concorrente é com a formulação do problema como programação linear inteira (*0-1 integer linear programming*). O problema de buscar uma solução utilizando esta técnica é que a programação linear inteira é um problema NP-completo, e o tempo gasto ao procurar uma solução limita o tamanho do problema.

Dito isto, outras técnicas são utilizadas para encontrar uma solução, tais como relaxar o problema para torná-lo em um problema de programação linear, pois estes podem ser resolvidos em tempo polinomial. O trabalho de (BEHJAT; VANNELLI; ROSEHART, 2006) é um exemplo desta técnica, onde antes de executar a programação linear, é realizado um relaxamento linear progressivamente até concluir o roteamento.

Normalmente, as técnicas de programação linear inteira são embutidas em *frameworks* com propósito de roteamento global com técnicas de divisão e conquista hierárquicas para resolver problemas menores possíveis de serem resolvidos computacionalmente (WANG; CHANG; CHENG, 2009).

Algoritmos baseados em *Steiner Trees*

Diferentemente dos algoritmos que tratam caminhos entre dois pinos, a construção de árvores de Steiner é projetada para multi-terminais (três ou mais pinos). Quando temos redes com duas ou mais conexões multi-terminais, uma boa solução é a decomposição da rede em diversas redes menores de ligações de dois pinos. E fazer o posterior roteamento destas um por um.

Um método popular para esta decomposição é encontrar a *minimum spanning tree* (MST) para os pinos de cada rede, a qual é a árvore de menor tamanho conectando todos os pinos. Este método pode ser computado em tempo polinomial utilizando algoritmos como o Kruskal ou o Prim-Dijkstra (WANG; CHANG; CHENG, 2009). A qualidade do roteamento gerada com a utilização desta técnica (MST) depende da forma em que as redes foram decompostas, e torna difícil a produção de resultados melhores, pois não existem iterações entre a fase de decomposição e a fase de roteamento posterior a esta.

A utilização de árvores de Steiner é normalmente a escolha de técnica para o roteamento de redes multi-terminais, e normalmente são utilizadas árvores de Steiner retilíneas (*Rectilinear Steiner Trees* - RST), pois estas obedecem as direções vertical e horizontal utilizadas pela arquitetura Manhattan em roteamento. A arquitetura Manhattan permite que o roteamento seja realizado somente em linhas verticais e horizontais, e a distância Manhattan é o percurso necessário para um ponto chegar ao outro utilizando estes passos verticais e horizontais.

O problema de encontrar uma RST de custo mínimo (RSMT) é *NP-hard* (GAREY; JOHNSON, 1977), e dito isto, para encontrar uma solução foram propostas diversas heurísticas. A maioria destas heurísticas dependem da busca da construção da árvore de Steiner retilínea de menor comprimento de fio (*Rectilinear Steiner Minimal Tree* - RSMT) através da combinação das árvores de Steiner e árvores de extensão de custo mínimos (*Minimum Cost Spanning Trees* (MST)). A RSMT é uma árvore com o menor comprimento total das arestas em distância Manhattan para conectar uma lista de nodos possivelmente através da utilização de alguns pontos Steiner extras (CHU; WONG, 2008).

A construção da árvore de Steiner retilínea de peso mínimo possui diversas aplicações

no projeto físico de circuitos VLSI. Na fase de *floorplanning*, ela pode ser utilizada para estimar o comprimento de fios, congestionamento no roteamento e *delay* das interconexões. No caso do roteamento global e detalhado, ela é utilizada para gerar a topologia de roteamento de cada rede.

Pelo fato da geração da RSMT ser um problema *NP-hard*, na prática é mais comumente utilizada a árvore de extensão mínima retilínea (*rectilinear minimum spanning tree* - RMST) ao invés da RSMT. A maior diferença na utilização de uma RMST ao invés de uma RSMT é que o tamanho da RMST é normalmente muito maior do que da RSMT, pois pontos Steiner não são permitidos nesta abordagem.

A abordagem RSMT conecta todos os pontos no plano com linhas retilíneas com a utilização de pontos extras chamados de pontos Steiner com o intuito de encontrar o menor comprimento de fio das arestas retilíneas. Podem existir infinitos pontos Steiner, os quais precisam ser considerados na construção da RSMT. A grade de Hanan é formada pela construção de linhas horizontais e verticais passando por cada pino de um *array* de pinos P , e existe uma RSMT de P com todos os pontos Steiner na grade de Hanan (WANG; CHANG; CHENG, 2009). A Figura 11 apresenta uma RSMT e a sua grade de Hanan, onde (a) é uma rede P que consiste de quatro pinos p_1, p_2, p_3, p_4 . (b) A RSMT de (a) com os dois pontos Steiner s_1 e s_2 e (c) A grade de Hanan de P . É possível notar que todos os pontos Steiner (s_1 e s_2) em (b) foram escolhidos da grade de pontos na grade de Hanan.

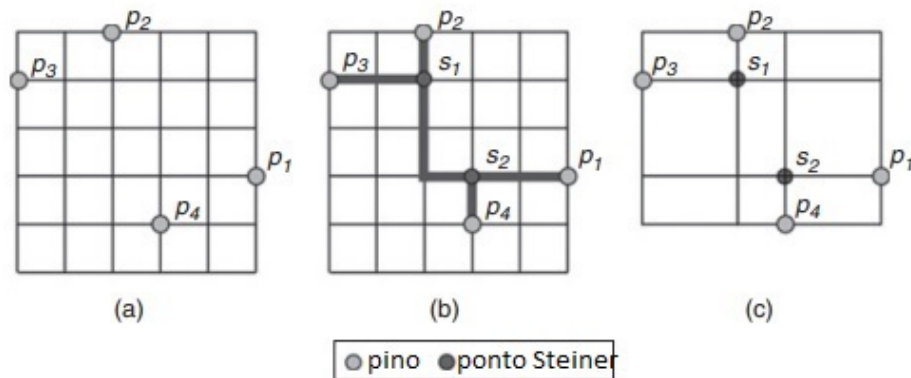


Figura 11: Uma RSMT e sua grade de Hanan. (a) Uma rede P que consiste de quatro pinos p_1, p_2, p_3, p_4 . (b) A RSMT com dois pontos Steiner (s_1 e s_2). (c) A grade de Hanan de P . Adaptado de (WANG; CHANG; CHENG, 2009).

Outras técnicas aplicadas no roteamento global

Além dos algoritmos que são mais comumente utilizados para realizar o roteamento, existem diversas outras técnicas que podem ser aplicadas nesta etapa. Nesta seção serão apresentadas algumas técnicas aplicadas por roteadores globais atuais.

Roteamento orientado à desempenho: O atraso das interconexões é de extrema importância no projeto de circuitos digitais de alto desempenho e, o atraso das interconexões é extremamente relevante dentro do atraso total da rede. Além disso, a proximidade entre módulos e suas interconexões resultam em um ruído *crosstalk*.

Os ruídos são variações indesejadas, as quais trazem modificações no comportamento esperado do circuito. Eles podem ocorrer por defeitos de fabricação do circuito ou por erros no sincronismo dos sinais, sendo que o maior ruído é proveniente do efeito *crosstalk*, o qual é causado pela capacitância acoplada entre os fios das interconexões (WANG; CHANG; CHENG, 2009).

Em (ZHOU; WONG, 1999), foi considerado começar a evitar o *crosstalk* ainda na etapa de roteamento global. Para isto, utilizaram formulação de árvores de Steiner e *La-grange relaxation* na fase inicial de roteamento. Porém, durante a etapa do roteamento global, não existem informações exatas sobre as posições relativas e ordenação das redes. Então, para conseguir tratar do *crosstalk* durante o roteamento global, são criadas estimativas para auxiliar, com o objetivo de que seja mais difícil que uma rede entre em regiões não desejadas.

Roteamento orientado à litografia OPC: *Optical Proximity Correction* (OPC) é o processo de modificar os padrões de leiaute na máscara para compensar as propriedades que não podem ser atingidas pelo processo de litografia *sub-wavelength* (WANG; CHANG; CHENG, 2009). Temos como exemplo o trabalho de (CHEN; CHANG, 2007), onde foi proposto um roteador para reduzir os requerimentos de padrões OPC. Neste trabalho, as distorções de padrões foram classificadas em três tipos maiores, e para cada tipo de distorção foram aplicadas técnicas para melhorar suas formas, como por exemplo, aumentar as larguras das linhas dos fios para compensar o seu encolhimento.

Roteamento de formas padrão (Pattern Routing): O roteamento de formas padrão realiza o roteamento de dois pinos em padrões pré-definidos, como em formato de "Z" e "L". Este método é mais rápido e eficiente se comparado com *maze routing*. Porém, a solução pode não ser ideal se todas as rotas para os dois pinos não forem consideradas (UDGIRKAR; INDUMATHI, 2016). A solução ótima do caminho escolhido não é atendida somente com o uso deste algoritmo. Por este motivo, normalmente são utilizados outros algoritmos em conjunto. A Figura 12 ilustra o roteamento de formas padrão em "L"(a) e em "Z"(b).

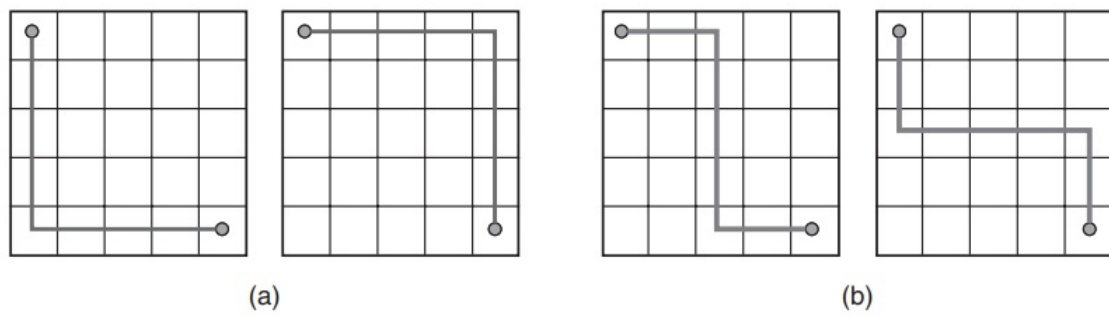


Figura 12: Roteamento de formas padrão: (a) Roteamento de formas padrão em "L"(uma dobra). (b) Roteamento de formas padrão em "Z"(duas dobras) (WANG; CHANG; CHENG, 2009).

Roteamento Monotônico (Routing Monotically): O roteamento monotônico é empregado em redes de dois pinos, e consiste em expandir do nodo origem sempre em direção ao nodo destino, esta técnica pode ser vista também como uma forma melhorada de *maze routing*. Em resumo, o roteamento monotônico pode ser dito como um roteamento guiado, onde a grade de expansão do pino de origem é expandida sempre em direção ao pino de origem, não expandindo em direções contrárias, e desta forma, acaba reduzindo o espaço de busca (PAN; CHU, 2007).

Desfazer e re-rotear (Rip-up and Re-route): *Rip-up and re-route.* ou desfazer e re-rotear, é uma técnica que faz melhorias na solução inicial gerada de forma iterativa. Em resumo, os roteadores globais modernos normalmente consistem em dois maiores passos: um roteamento inicial de todas as redes, com pouco ou nenhum esforço para minimizar o congestionamento, onde é gerado um mapa de congestionamento das redes. O segundo passo é um processo de desfazer e re-rotear, onde o congestionamento é minimizado iterativamente, desfazendo e re-roteando cada rede que passa pelas regiões mais congestionadas, as redes que passam por estas regiões são desfeitas e re-roteadas até que haja pouco ou nenhum congestionamento no resultado (SIDDIQI; SAIT, 2017). A qualidade da solução gerada com a utilização deste método depende de como as redes foram ordenadas para o processamento do roteamento.

De acordo com (SIDDIQI; SAIT, 2017), o processo de desfazer e re-rotear convencional consiste em, sequencialmente, desfazer e re-rotear as redes em uma ordem pré-determinada. Esta técnica auxilia na otimização dos resultados de roteamento, porém, os problemas de roteamento global estão ficando cada vez mais complexos, o que demanda a implementação de diversas técnicas de melhorias para trabalharem junto deste processo. Através da utilização do processo de desfazer as redes e re-rotear, espera-se como resultado um roteamento com pouco, ou nenhum congestionamento restante, além da diminuição do comprimento total de fios utilizado na solução. Ainda segundo (SIDDIQI; SAIT, 2017), soluções com nenhum congestionamento no resultado possuem melhores chances de serem roteadas com sucesso no processo posterior de roteamento

detalhado.

A Figura 13 apresenta o roteamento de duas redes baseado em diferentes ordenamentos destas. É possível observar que se a rede B for ordenada antes da rede A, a solução do roteamento é melhor, pois é utilizado menos fio para realizar as ligações entre os pontos.

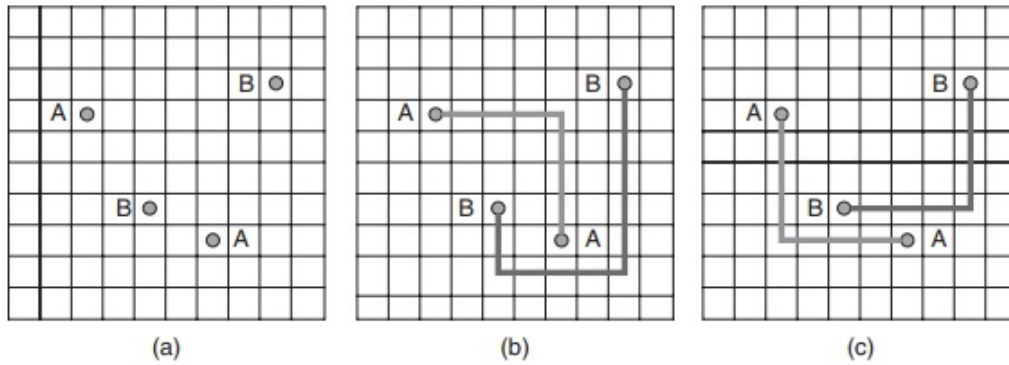


Figura 13: Roteamento baseado nas diferentes ordens das redes. (a) Uma camada com duas redes de dois pinos A e B. (b) Uma solução ruim obtida por rotear A e depois B. (c) Uma solução melhor obtida por rotear B e depois A (WANG; CHANG; CHENG, 2009).

Roteamento baseado em negociação (Negotiated Congestion Routing): Proposto inicialmente por (MCMURCHIE; EBELING, 1995), este método foi criado com a intenção de equalizar os dois objetivos de melhorar a performance afetada pelos tempos de sinais e eliminação de congestionamento. Para isto, a função de custo para rotear uma rede é definida pela Equação 1, apresentada abaixo:

$$custo_a = (B_a + H_a) \times P_a \quad (1)$$

Na Equação 1, B_a é o custo base para usar a , H_a é a variável que faz relação com o congestionamento das iterações anteriores, e P_a é a variável que faz relação com a quantidade das outras redes que passam por a no momento. Se existem congestionamentos no vértice durante iterações anteriores, H_a desencoraja o algoritmo a rotear através de a , e preserva os recursos de a para redes com menos escolhas de rotas. A variável H_a é atualizada iterativamente, onde caso a tenha congestionamento é incrementado por um parâmetro definido pelo usuário, caso contrário é mantido o valor normalmente (UDGIRKAR; INDUMATHI, 2016).

Multi-sink Multi-source Maze Routing: Um algoritmo de *maze routing* tradicional começa a procurar os caminhos levando em consideração a origem inicial dos pinos origem e destino de uma rede. O algoritmo multi-sink multi-source pode ser visto como uma melhoria realizada no *maze routing* original. Neste algoritmo as redes multi-pinos são divididas em duas sub-árvores, então, todos os pontos da primeira árvore são tratados como nodos de origem, e todos os pontos da segunda árvore são tratados como pontos de destino, permitindo desta forma que ocorra uma conexão entre os pinos da rede diminuindo

o comprimento de fio. Este algoritmo possui o problema de quando as redes multi-pinos possuírem muitos nodos, aumentar a quantidade de tempo para identificar todos os pontos da grade nas sub-árvores (CHANG; LEE; WANG, 2008).

3.4 Roteamento Global e Paralelismo

O roteamento global com múltiplas camadas de metal (3D) possui um espaço de busca relativamente maior em comparação com o roteamento global restrito a duas camadas de metal (2D), o que acaba demandando um tempo muito maior para o processamento. Arquiteturas *multi-core* e GPU (*Graphics Processing Unit*) tornam possível o processamento paralelo, que pode ser utilizado para acelerar na resolução deste problema (QI et al., 2014). A paralelização é um candidato promissor para aumentar a velocidade na resolução de problemas de roteamento, e gerar resultados rápidos sem impacto na qualidade destes é um desafio (HAN; CHAKRABORTY; ROY, 2013).

Um dos principais gargalos, que dificulta a paralelização dos problemas de roteamento é a concorrência entre as redes para a utilização dos recursos disponíveis para o roteamento (HAN; CHAKRABORTY; ROY, 2013). Diversos trabalhos recentes buscaram o uso de estratégias de paralelismo para resolver ou melhorar o desempenho para a solução de problemas de roteamento global e detalhado. O processamento paralelo foi utilizado em alguns destes trabalhos com recursos de CPU *multi-thread* (LIU et al., 2013) (QI et al., 2014), GPU (HAN; CHAKRABORTY; ROY, 2013), e também CPU em conjunto com GPU (HAN et al., 2014).

Existem duas principais abordagens para empregar paralelismo na solução de problemas de roteamento global (QI et al., 2014) (TUMELERO; BONTORIN; REIS, 2015):

- Dividir o mapa de roteamento em regiões menores: consiste em dividir o circuito em partes menores para serem processadas em paralelo posteriormente;
- Dividir as redes independentes: consiste em dividir redes independentes (distantes umas das outras, e/ou que não utilizem os mesmos recursos), e então processar em paralelo o roteamento destas.

No caso do roteamento paralelo, é dito que ocorreu um conflito se múltiplas *threads* demandam o uso de um mesmo recurso de roteamento. Dentro do contexto da divisão do mapa de roteamento em regiões menores para a realização do roteamento, existe a dificuldade de encontrar pontos de particionamento do circuito de forma que conflitos sejam evitados, e também, na resolução dos conflitos quando ocorrerem, pelo motivo das regiões de roteamento de uma rede acabam sobrepondo o de outras redes, tornando difícil conseguir um bom candidato a região independente (LIU et al., 2013) (TUMELERO; BONTORIN; REIS, 2015).

Já no caso da divisão das redes independentes, é possível a exploração de paralelismo de uma forma melhor e um mais equilibrado balanceamento de carga, porém esta possui uma complexidade de implementação maior, e pode levar a uma geração de resultados com maior congestionamento caso as redes não tenham acesso às informações sobre colisões na utilização de recursos (HAN et al., 2014). Tanto a estratégia da divisão do mapa de roteamento em regiões menores para o processamento quanto a estratégia da utilização das redes independentes ainda requer processamento sequencial em partes do fluxo de roteamento (TUMELERO; BONTORIN; REIS, 2015).

Em (QI et al., 2014) podemos ver o uso de ambas estratégias de divisão de partes do *chip* e divisão de redes independentes, com o roteamento sendo realizado em um nível *multi-thread*. A arquitetura *multi-core* se tornou o principal foco no *design* de CPUs, isto torna o desenvolvimento de algoritmos voltados para EDA promissor dentro deste contexto, e a implementação de algoritmos voltados para arquitetura *multi-core* pode melhorar a performance e a qualidade na resolução dos problemas de EDA (LIU et al., 2013).

De acordo com (HAN et al., 2014) diversos problemas relacionados ao projeto de circuitos VLSI já foram resolvidos com o auxílio do uso de GPU, resultando em diminuições de até 100 vezes no tempo de execução. Um dos maiores gargalos nas aplicações envolvendo GPU é a comunicação entre GPU e CPU através do canal PCI-E, o que se torna um dos primeiros obstáculos. Uma outra barreira que pode gerar dificuldades na abordagem de paralelismo através do uso de GPU para a resolução de problemas de roteamento é o já citado fato de que algoritmos de roteamento normalmente utilizam estruturas de dados compartilhadas, e desta forma, há uma grande dependência entre os dados, o que não se encaixa bem em um dos principais requisitos de soluções que envolvem arquiteturas de GPUs, a independência de dados (HAN; CHAKRABORTY; ROY, 2013) (HAN et al., 2014).

4 ESTADO-DA-ARTE

A partir da competição ISPD 2007, a qual teve como tema roteamento global de circuitos, surgiram diversos roteadores globais que até os tempos atuais constituem o estado-da-arte (SHYAMALA; LATHA, 2016). Os resultados que estes roteadores geram a partir dos *benchmarks* do ISPD07, ISPD08 e ISPD98 são satisfatórios (conseguem soluções sem congestionamento nos *benchmarks* solucionáveis, e com otimização no comprimento total de fio utilizado). Até mesmo nos *benchmarks* identificados como difíceis de resolver pelos provedores dos *benchmarks*, os resultados obtidos pelos mesmos são bons ou próximos do desejado. Mesmo com os bons resultados gerados a partir destes roteadores, pesquisas e desenvolvimentos atuais mostram que ainda é possível gerar melhorias em seus resultados. Por exemplo, no trabalho de (SIDDIQI; SAIT, 2017) é realizada uma remodelagem do processo de desfazer e re-rotear dos roteadores NCTU-Gr (LIU et al., 2013) e BFG-R (HU; ROY; MARKOV, 2010), enfatizando que ainda é possível gerar melhorias, não somente em tempo de processamento, mas também em resultados para o roteamento global.

Neste capítulo são apresentados roteadores estado-da-arte. Foram buscados roteadores criados durante as competições do ISPD2007 e ISPD2008. Os trabalhos apresentados a seguir são citados até os dias atuais como estado-da-arte em roteamento global de circuitos acadêmicos (SIDDIQI; SAIT, 2017) (SHYAMALA; LATHA, 2016). Entre os trabalhos estudados temos os participantes das competições ISPD07 e ISPD08 NTHU Router (GAO; WU; WANG, 2008), FastRoute (XU; ZHANG; CHU, 2009), BFG-R (HU; ROY; MARKOV, 2010) e NCTU-GR (LIU et al., 2013). Além dos participantes das competições ISPD07 e ISPD08, também foram estudados roteadores que surgiram depois, como XGRouter (LIU et al., 2015), GT Router (SIDDIQI; SAIT, 2017) e (HAN et al., 2014).

4.1 NTHU-Router

O roteador global NTHU-router (GAO; WU; WANG, 2008) (CHANG; LEE; WANG, 2008) (LEE; CHANG; WANG, 2010), é atualmente um dos roteadores estado-da-arte, e

vencedor do concurso ISPD 2008 com a sua versão 2.0 da ferramenta. A última melhoria realizada neste roteador foi em 2010 em (LEE; CHANG; WANG, 2010). Porém, ainda em 2016, este trabalho foi citado como estado da arte em (UDGIRKAR; INDUMATHI, 2016). Para gerar uma solução para o roteamento global, este roteador aplica o processo de desfazer e re-rotear, em conjunto com outras técnicas.

Inicialmente, o roteador usa a técnica de *edge shifting* e aplica o algoritmo FLUTE (CHU; WONG, 2008) em cada rede multi-pinos, gerando sua RSMT (*Rectilinear Steiner Minimal Tree*) equivalente. Então, cada uma destas redes é decomposta em uma lista de redes de dois pinos, conforme mostra o exemplo da Figura 14. Para cada rede gerada, é aplicado o roteamento de formas padrão em forma de "L" para realizar a primeira versão do roteamento, e assim construir o primeiro mapa de congestionamento.

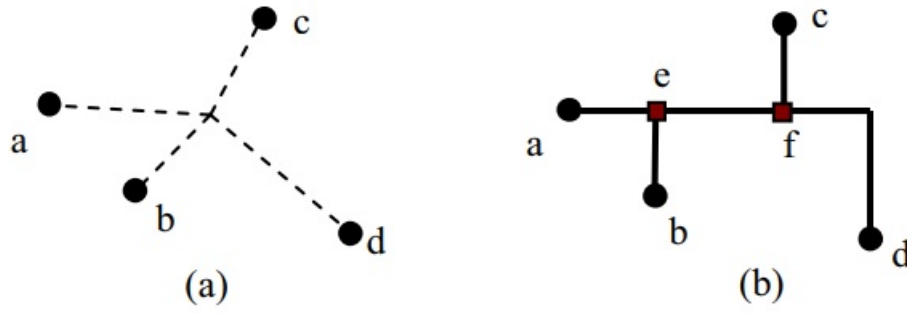


Figura 14: (a) Rede multi-pinos composta por 4 pinos. (b) Rede decomposta em redes de dois pinos utilizando a topologia de Steiner trees. Pontos pretos representam pinos e retângulos vermelhos os pontos Steiner (GAO; WU; WANG, 2008).

Na segunda fase, foi proposta uma função de custo baseada no histórico, que é utilizada para o processo de desfazer e re-rotear iterativo posterior. A função é apresentada na Equação 2, onde $custog$ é o custo gerado como resultado da equação, bg é o custo básico para o uso de uma aresta g , $hg \times pg$ é o custo de congestionamento de uma aresta g , e vcg é o custo da via ao usar a aresta g . Esta função é a combinação do comprimento do fio e do congestionamento, onde o primeiro termo da função é setado como um, correspondendo a uma unidade de comprimento. O termo de penalização pg é atualizado conforme uma função de histórico. Quando mais iterações completas, as arestas que tendem a ficar congestionadas ficam com um custo maior.

$$custog = bg + hg \times pg + vcg \quad (2)$$

Na terceira fase, as regiões congestionadas são localizadas, e então a técnica de desfazer e re-rotear é aplicada. As redes de dois pinos são desfeitas e então re-roteadas utilizando roteamento monotônico seguido de *maze routing* adaptativo com múltiplas origens ou múltiplos destinos.

A fase final do roteador consiste em aplicar um processo de refinamento iterativo para

lidar com o congestionamento ainda existente caso esse exista, nesta fase é realizado o roteamento sem levar em consideração constantes que são levadas na fase anterior, como por exemplo, a ordenação das redes.

A Figura 15 apresenta o fluxo do sistema do roteador NTHU, onde o fluxo é dividido em Estágio Inicial, Estágio Principal, Estágio de Refinamento, e finalmente Estágio de Assinalamento de Camadas.

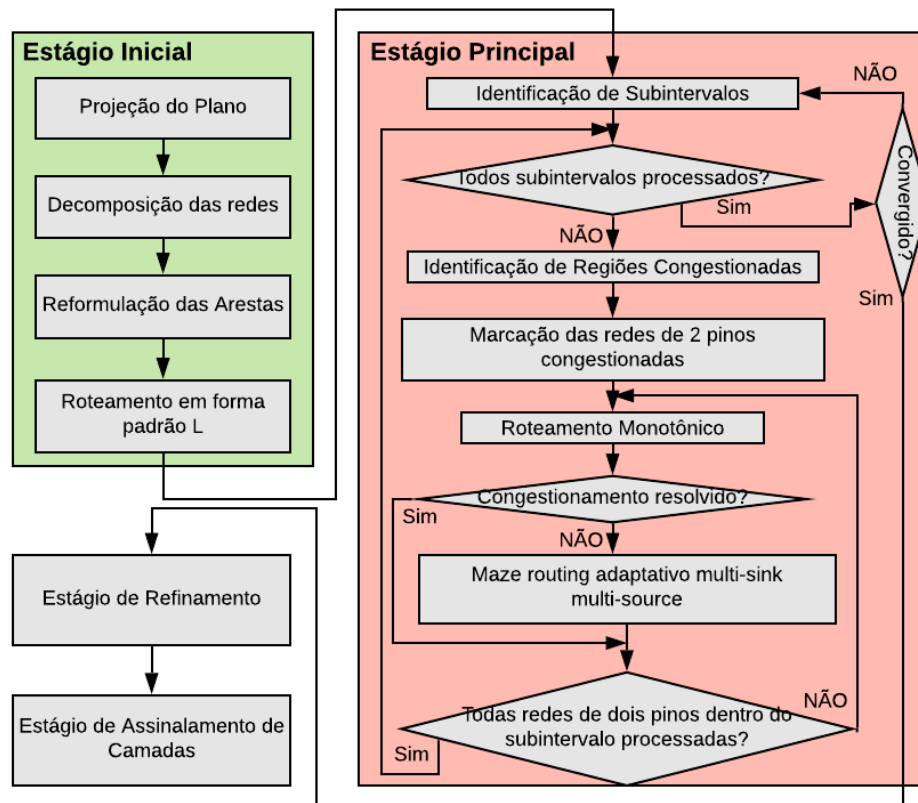


Figura 15: Fluxo do sistema do roteador NTHU v2. Adaptado de (CHANG; LEE; WANG, 2008).

A versão 2.0 do roteador NTHU(CHANG; LEE; WANG, 2008) apresentou algumas melhorias aplicadas no roteador inicial. Foram realizadas as seguintes melhorias:

1. Uma nova função de custo baseada em histórico (agora dividida em função de custo base, de congestionamento e de via).
2. Novos métodos para ordenação para o processo de desfazer e re-rotear, e identificação de regiões congestionadas.
3. Duas novas técnicas para redução de tempo de execução.

A função de custo base apresentada na Equação 3 se tornou adaptativa, onde β e γ são parâmetros definidos pelo usuário, e i é o contador de iterações. No roteador NTHU 2.0

β foi atribuído como 5 e γ foi atribuído como 0.1, e como resultado b_g ficará entre 0 e 1.

$$b_g = 1 - a^{-\beta e^{-\gamma i}} \quad (3)$$

A função de congestionamento sofreu adaptações para simular predições, e a função de via foi adaptada para suportar projetos com mais de duas camadas de metal.

Os métodos de ordenamento para o processo de desfazer e re-rotear e identificação do congestionamento no primeiro roteador NTHU eram iniciados com o ordenamento pela rede com menor perímetro. Na versão 2.0 do roteador, a ordenação foi revertida, começando pela rede com maior perímetro. Na Figura 16 é apresentado um exemplo desta reversão da ordem de ordenação aplicada, onde (a) apresenta uma solução de roteamento retirada de uma iteração anterior, (b) a solução obtida com o roteador NTHU e (c) a solução obtida com o roteador NTHU com o novo método aplicado.

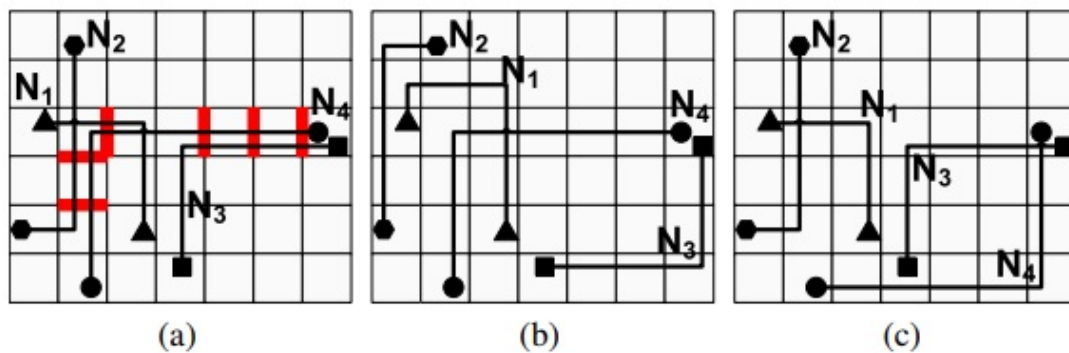


Figura 16: Exemplo da mudança na ordenação das redes na etapa de desfazer e re-rotear. (a) Solução retirada de uma iteração anterior. (b) Solução do roteador NTHU. (c) Solução do roteador NTHU com a modificação na ordenação das redes (CHANG; LEE; WANG, 2008).

A redução do tempo de execução para encontrar as soluções na ferramenta partiu da identificação de dois gargalos. Um destes é a necessidade de calcular o custo das arestas quando está procurando caminhos para redes de dois pinos congestionadas. Na primeira versão da ferramenta, o cálculo do custo das arestas era realizado em tempo de execução. Sempre que era necessário calcular, a ferramenta o fazia no momento em que precisava. Para diminuir este gargalo foi realizada a pré-computação e armazenamento do custo das arestas previamente. Além disto, na segunda ferramenta são calculados e armazenados a soma da base de custo e o custo do congestionamento para cada aresta, e somente é realizada a alteração destes se necessário.

O segundo gargalo é determinar se uma parte de uma rede multi-pinos já passa por uma aresta. Na primeira ferramenta cada aresta possuía uma tabela para informar quais redes de dois pinos passam por esta aresta, tabela esta que é usada frequentemente e é implementada por uma árvore de busca balanceada. Para realizar a melhoria neste

gargalo, na versão 2.0 da ferramenta, os autores realizaram a mudança da árvore de busca balanceada para uma tabela *hash*, o que tornou possível a ferramenta ser mais rápida, porém consumindo mais memória.

A Tabela 1 apresenta os resultados de congestionamento total (TOF), comprimento total de fio (WL) e tempo de execução (CPU) obtidos pelo roteador NTHU e pelo NTHU v 2.0 utilizando os *benchmarks* do ISPD 2008.

Tabela 1: Comparação dos resultados do NTHU-Route nos *benchmarks* do ISPD08 (onde existem -"a ferramenta não conseguiu encontrar uma solução), adaptado de (CHANG; LEE; WANG, 2008).

Benchmark	NTHU-Route			NTHU-Route 2.0		
	TOF	WL(e5)	CPU (min)	TOF	WL(e5)	CPU (min)
adaptec1	0	54.18	93.6	0	53.44	10.0
adaptec2	0	52.77	16.8	0	52.29	2.1
adaptec3	0	132.85	64.9	0	131.01	10.9
adaptec4	0	122.65	10.1	0	121.69	2.6
adaptec5	0	157.30	268.4	0	155.38	23.0
bigblue1	0	56.23	171.97	0	55.95	13.7
bigblue2	198	90.23	73.17	0	90.59	8.5
bigblue3	18	131.62	93.05	0	130.68	5.0
bigblue4	-	-	-	162	231.04	131.6
newblue1	352	46.17	38.0	0	46.53	6.2
newblue2	0	76.25	4.3	0	75.70	1.2
newblue3	31800	106.98	357.7	31454	106.53	122.4
newblue4	448	129.10	144.89	138	130.46	96.7
newblue5	-	-	-	0	231.58	19.4
newblue6	0	177.97	244.32	0	176.89	18.1
newblue7	-	-	-	62	353.35	143.9

Existe ainda uma terceira versão do roteador NTHU (LEE; CHANG; WANG, 2010). Nesta, foram realizadas três melhorias se comparadas com a versão 2.0. As melhorias foram:

1. Roteamento em forma padrão de "L" considerando temperatura.
2. Alteração nas funções de custo na fase de desfazer e re-rotear, agora considerando temperatura.
3. Três técnicas para redução do uso de memória.

Para levar a temperatura em consideração no estágio inicial, o método de roteamento padrão em formato de "L" foi modificado da seguinte forma: Se todos os caminhos passam pelo mesmo número de *hot spots*, a função original é utilizada para escolher um dos caminhos. Se não, é escolhido o caminho que passa por menos *hot spots*. As funções de

custo receberam um termo térmico adicional para penalizar os caminhos das redes por *hot spots*.

Para a redução da memória, os tamanhos das tabelas *hash* foram estimados mais cuidadosamente, diminuindo as entradas iniciais de 128 na fase principal e 32 no assinalamento de camadas para 32 e 5 respectivamente. Além disto, é utilizado um vetor para guardar o caminho de uma rede de dois pinos, e para eliminar caminhos antigos era utilizada uma função *clear()*, porém esta função não desalocava a memória que estava sendo utilizada. Para resolver isto, foi adotado o método de colocar a estrutura em uma variável vazia, assim desalocando a memória utilizada. Finalmente, removendo estruturas de dados que não estavam sendo utilizadas resultou em uma diminuição ainda maior da memória utilizada, somando as três técnicas foi obtida uma redução do uso de memória de 44.4%.

A última versão do roteador desviou 14% das redes que passam por *hot spots*. Porém, aumentando em 4% o comprimento dos fios, e com um tempo de execução 4.5x menor que o do roteador NTHU 2.0.

4.2 FastRoute

O roteador global FastRoute possui uma abordagem diferente da tradicionalmente adotada por outros roteadores. Seu objetivo é o de ser aplicado tanto na etapa de posicionamento, quanto na de roteamento. O uso de *maze routing* em excesso é evitado por este roteador. Devido ao tempo consumido para executar algoritmos de *maze routing*, ele adota uma abordagem diferente.

O funcionamento do roteador pode ser dividido em três fases. Na fase inicial é aplicado o algoritmo FLUTE criando árvores de Steiner para todas as redes (CHU; WONG, 2008). Então, todas árvores de Steiner geradas são reformuladas em redes de dois pinos, e o roteamento padrão em forma de "L" é executado para realizar o roteamento inicial destas redes. Após a execução deste roteamento inicial nesta primeira fase, é obtido o mapa de congestionamento.

Na segunda fase, para reduzir o congestionamento e construir árvores de Steiner ótimas, são aplicadas duas técnicas. A primeira consiste em construir topologias de árvores de Steiner com intuito de diminuir o congestionamento utilizando de um algoritmo orientado à construção de topologias baseado no congestionamento. O algoritmo estende o procedimento de diminuição de congestionamento aplicado no pacote FLUTE.

A Figura 17 apresenta diferentes topologias de árvores de Steiner geradas por uma rede de 6 pinos. O resultado desta abordagem é a diminuição de arestas nas regiões congestionadas. A segunda técnica é uma troca de arestas (*swap*), buscando diminuir ainda mais o congestionamento após montada a nova topologia da árvore. Sem causar mudanças no comprimento total de fio da árvore as arestas são trocadas, e ao mesmo

tempo a demanda por roteamento é também trocada de regiões com congestionamento para regiões sem congestionamento, de forma a diminuir o congestionamento local. Estas duas técnicas são aplicadas para cada rede que contenha mais de quatro pinos, e o mapa de congestionamento é modificado a cada modificação nas redes.

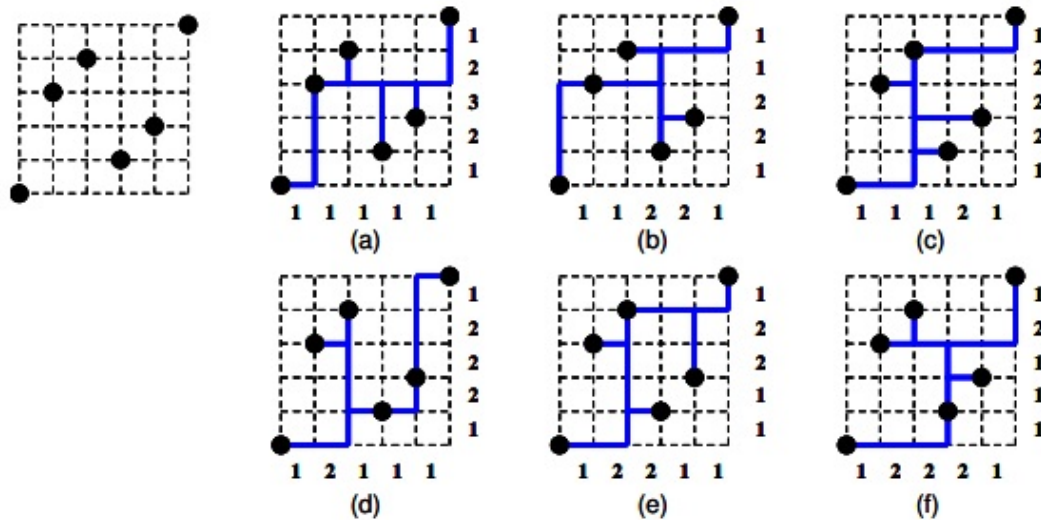


Figura 17: Diferentes topologias de árvores de Steiner geradas por uma rede de 6 pinos. Adaptado de (PAN; CHU, 2006).

A última etapa consiste em partir as redes multi pinos em redes de dois pinos, e então no roteamento das redes de dois pinos geradas utilizando de *maze routing* e roteamento de formas padrão. Primeiro o roteamento de formas padrão é aplicado para rotear todas as redes de dois pinos, nesta etapa é utilizado o padrão em forma de "Z". Depois de executar o roteamento inicial, a técnica de desfazer e re-rotear é aplicada utilizando roteamento de labirinto. Para encontrar as regiões menos congestionadas nesta etapa, uma função logística é aplicada.

A versão 2.0 do FastRoute (PAN; CHU, 2007) apresentou duas melhorias: roteamento monotônico e *maze routing multi-sink multi-sources*. O roteamento monotônico substituiu o roteamento de formas padrão que era utilizado inicialmente na primeira versão do FastRoute, e o *maze routing* multi-fonte é uma técnica que produz melhores resultados no roteamento.

A ideia de utilizar roteamento monotônico foi encontrar um meio-termo entre o roteamento de formas padrão e o *maze routing*. O roteamento de formas padrão é muito mais rápido que o *maze routing*. Porém, a qualidade é inferior se considerado com o roteamento monotônico. A segunda técnica proposta foi a de respeitar a árvore de roteamento. Quando uma aresta para multi-pinos é roteada, os pontos de término originais do caminho de roteamento não são restringidos a serem os pontos originais da aresta a ser roteada.

A terceira versão do roteador FastRoute (ZHANG; XU; CHU, 2008) foi a terceira colocada no ISPD 2008. Esta versão introduziu uma nova técnica chamada capacidade

virtual. A capacidade virtual guia o roteador no estágio de *maze routing*, ajudando a atingir melhores resultados em termos de tempo de execução e congestionamento.

Esta técnica pode ser dividida em inicialização da capacidade virtual, e posterior atualização desta. Antes de inicializar o *maze routing*, o roteador inicializa a capacidade virtual subtraindo a previsão do congestionamento gerado por estimativa de congestionamento adaptativa através da capacidade real da aresta. Então, nas iterações posteriores da fase do *maze routing*, a capacidade virtual é reduzida pela quantidade existente de congestionamento para as arestas que ainda estão congestionadas.

Em sua versão 4.0, o roteador FastRoute (XU; ZHANG; CHU, 2009) buscou diminuir o uso de vias durante todo o fluxo do roteamento global, aplicando três principais técnicas: (1) Geração de árvores de Steiner orientadas à diminuição de vias, (2) roteamento de três curvas (*3-bend routing*) e (3) assinalamento de camadas com ordenação de redes e arestas melhoradas. A Figura 18 apresenta o fluxo seguido pelo roteador FastRoute 4.0.

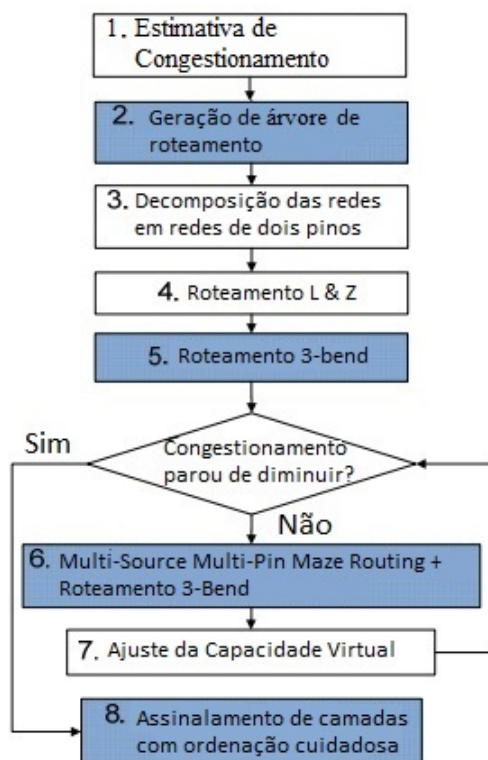


Figura 18: Fluxo seguido pelo FastRoute 4.0. Adaptado de (XU; ZHANG; CHU, 2009).

A geração das árvores de Steiner orientadas à diminuição das vias produzem uma topologia que diminui o número de vias baseado no congestionamento e nas informações sobre as camadas.

O roteamento de três curvas é um algoritmo de roteamento eficiente com opções de desvios. Ele possui capacidade de diminuir o congestionamento assim como *maze routing*, e ao mesmo tempo mantêm um controle eficiente do número de vias utilizadas como

roteamento por formas padrão.

O algoritmo de assinalamento de camadas utilizado é um algoritmo sequencial que ordena tanto as redes quanto as arestas de cada uma destas redes. Primeiramente, as redes são ordenadas considerando o comprimento total de fio e o número de pinos. Então, as arestas de cada rede são ordenadas de acordo com suas localizações na rede. Finalmente, programação dinâmica é utilizada para assinalar camadas aresta por aresta, e rede por rede.

A Tabela 2 apresenta os resultados de congestionamento total (TOF), comprimento total de fio (WL) e tempo de execução (CPU) obtidos nos trabalhos (ZHANG; XU; CHU, 2008) e (XU; ZHANG; CHU, 2009) utilizando os *benchmarks* do ISPD 2008. A medição do comprimento de fio e tempo de CPU utilizam métricas diferentes nos dois roteadores, as diferenças são apontadas na segunda coluna da tabela.

Tabela 2: Resultados do FastRoute 3.0 e FastRoute 4.0 utilizando os *benchmarks* do ISPD 08.

Benchmark	FastRoute 3.0			FastRoute 4.0		
	TOF	WL(e5)	CPU(min)	TOF	TWL	CPU(s)
adaptec1	0	55.5	2	0	9100K	190
adaptec2	0	53.1	1	0	9157K	45
adaptec3	0	133	2	0	20461K	300
adaptec4	0	122	1	0	18695K	40
adaptec5	0	161	5	0	27046K	568
bigblue1	0	58.3	4	0	5789K	423
bigblue2	142	98.2	11	0	9559K	913
bigblue3	0	132	3	0	13070K	278
bigblue4	206	243	41	194	24165K	674
newblue1	76	49	13	0	9170K	283
newblue2	0	76.2	1	0	13563K	22
newblue3	31650	109	31	31710	18211K	1280
newblue4	226	136	10	144	13394K	1135
newblue5	0	241	5	0	23672K	607
newblue6	0	187	4	0	18262K	574
newblue7	588	359	190	58	35691K	11060

4.3 XGRouter

Diferente da maioria dos roteadores que fazem o uso da arquitetura Manhattan, o que significa realizar conexão de fios somente na vertical e horizontal, este roteador faz o uso da arquitetura X. O XGRouter (LIU et al., 2015) foi o primeiro roteador a utilizar um algoritmo concorrente para resolver o problema de roteamento global na arquitetura X. A arquitetura X é uma arquitetura de fiação baseada no uso difusivo de fios diagonais. De acordo com (LIU et al., 2015), a arquitetura X é totalmente suportada por quase todas

tecnologias de produção atuais. A Figura 19 apresenta uma comparação entre a arquitetura Manhattan e a arquitetura X, onde na grade da esquerda é ilustrada a arquitetura Manhattan (a), com os fios passando somente na vertical e horizontal, e na grade da direita é ilustrada a arquitetura X (b), onde além de permitir as conexões na vertical e horizontal, também é possível realizar conexões em X.

Neste roteador foi proposta uma nova formulação ILP (*Integer Linear Programming*) que consegue atingir soluções de roteamento mais uniformes. Assim como a maioria dos roteadores globais estado-da-arte, o XGRouter não possui congestionamento na maioria de seus resultados.

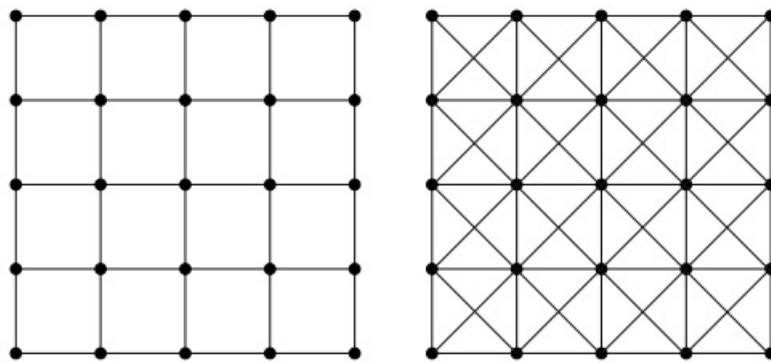


Figura 19: Comparação entre a arquitetura Manhattan (a) e a arquitetura X (b). Adaptado de (LIU et al., 2015).

Para que o roteador gere sua solução, inicialmente uma estratégia de particionamento é adotada, onde primeiramente o roteador faz o roteamento das regiões mais congestionadas no *chip*, e expande a região até cobrir todo o *chip*. O método baseado em formulações ILP consegue encontrar boas soluções sem considerar a ordem de roteamento das redes.

O método ILP utilizado neste trabalho é baseado em O-ILP. Esta formulação considera a uniformidade do congestionamento para realizar as distribuições do roteamento mais balanceadas e não causar muitos *hot spots*. Cada candidato a roteamento possui um peso, e os demais objetivos de otimização podem ser facilmente incorporados. Além disto, o método ILP utilizado possui uma restrição rigorosa com as capacidades das arestas, o que implica em uma solução sem arestas com congestionamento.

O fluxo deste roteador pode ser dividido em três estágios principais: estágio inicial, estágio principal e estágio pós-processamento. O fluxo do roteador é apresentado na Figura 20.

No estágio inicial, as redes multi-pinos são decompostas em diversas redes de dois pinos, para isto é utilizada uma SMT (*Steiner Minimal Tree*) para a arquitetura X, nomeada XSMT. Então é calculada a inclinação (*slope*) para cada linha, que é composta por dois pontos finais (*endpoints*) de cada rede de dois pinos. As redes de dois pinos são cheçadas para saber se satisfazem a condição de pré-roteamento. Se a condição for satisfeita, os

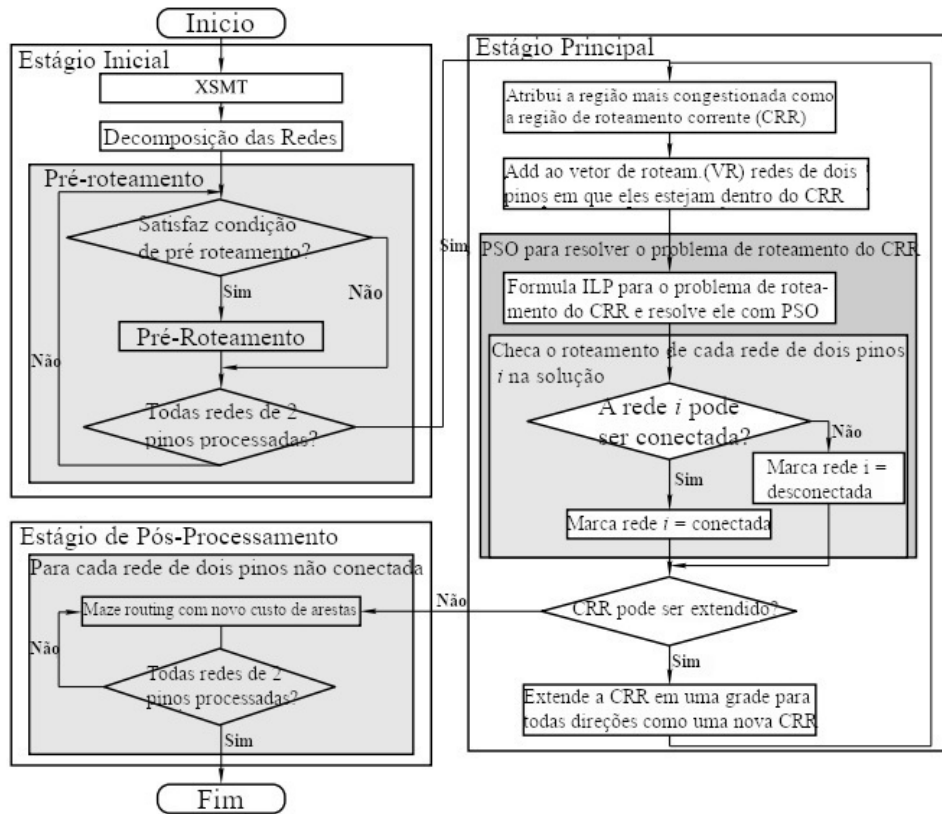


Figura 20: Fluxo do roteador XGRouter. Adaptado de (LIU et al., 2015).

pinos são conectados e a rede é marcada para ser conectada. Senão, a checagem continua na próxima rede de dois pinos até que todas as redes de dois pinos sejam processadas. No final deste estágio, todas as redes de dois pinos podem ser processadas.

No estágio principal, a estratégia de partição citada anteriormente é adotada. É escolhida uma região da grade GRG (*Global Routing Grid*) 2x2 com o maior congestionamento, então O-ILP é formulada para resolver o roteamento. Após rotear cada sub-região, uma nova região de roteamento será obtida expandindo a região de roteamento em uma grade a mais para cada direção (a região 2x2 vira uma região 3x3), e este procedimento é repetido até que a região de roteamento cubra toda a área do *chip*.

Depois de passar pelo estágio principal, podem existir redes de dois pinos que ainda não foram conectadas. Para isto, é implementado o estágio pós-processamento. Este estágio é necessário para que todas as redes sejam roteadas, para isto foi proposto um roteamento *maze routing* baseado no novo custo de roteamento das arestas para conectar todas as redes remanescentes.

Uma comparação deste roteador com outros roteadores estado-da-arte, incluindo alguns que já foram abordados nas Seções anteriores utiliza os *benchmarks* do ISPD 98. A Tabela 3 apresenta os resultados obtidos por este roteador com a métrica de comprimento total de fio (WL), e a comparação com os resultados obtidos pelo NTHU-Router 2.0 e

pelo FastRoute 3.0. Em (LIU et al., 2015) é possível encontrar comparações do XGRouter com outros roteadores que utilizam a arquitetura X, roteadores que utilizam abordagem concorrente e alguns que utilizam abordagem retilínea e participaram do ISPD 2008.

Tabela 3: Resultados de comprimento total de fio (TOF) dos roteadores NTHU-Router, FastRoute 3.0 e XGRouter aplicados nos *benchmarks* do ISPD98, adaptado de (LIU et al., 2015).

Benchmark	NTHU-Router	FastRoute 3.0	XGRouter
ibm01	62 498	62 659	57 989
ibm02	169 881	171 110	158 049
ibm03	146 458	146 634	136 284
ibm04	166 452	167 275	153 425
ibm06	277 696	277 913	261 729
ibm07	366 133	365 790	341 794
ibm08	404 976	405 634	379 902
ibm09	414 738	413 862	389 367
ibm10	579 870	590 141	542 568

4.4 NCTU-GR

O roteador NCTU-GR, em sua primeira versão (DAI; LIU; LI, 2012), inicia seu fluxo realizando a decomposição das redes multi-pinos em diversas redes de dois pinos. Para isto, ele realiza a decomposição através de MST (*Minimum Spanning Tree*), e então aplica o algoritmo FLUTE (CHU; WONG, 2008) para entregar RSMT como a solução inicial.

Para realizar o roteamento inicial, o roteador aplica uma técnica chamada roteamento monotônico circular com ordenamento fixo (*circular fixed-ordering monotonic routing* (CFOMR)). Esta técnica consiste em desfazer e re-rotear as redes de forma repetitiva na ordem decrescente de tamanho total de fio da rede, realizando roteamento monotônico um número n (constante) de vezes definida pelo usuário. De acordo com os autores, adotar a ordem decrescente de comprimento de fio em cada ciclo melhora a performance do roteamento quando processando fios grandes.

Na terceira parte, acontece o processo de desfazer e re-rotear baseado em evolução simulada, onde são aplicadas duas funções de custo em dois estágios diferentes para ajudar a resolver redes difíceis de rotear. Caso no processo anterior o congestionamento não consiga ser minimizado depois de muitas iterações, é possível que o processo de roteamento tenha ficado preso em uma máxima local, então o processo R&R entra em seu segundo estágio, e aplica uma nova função de custo para completar o roteamento das redes remanescentes.

A função *score* é modificada afim de selecionar as redes para o re-roteamento, de acordo com os autores, no processo inicial as redes com mais violações são desfeitas e re-roteadas primeiro. Porém, estas redes normalmente estão alocadas em uma área

com congestionamento, e rotear estas redes primeiro pode utilizar recursos que poderiam ser melhor aproveitados. Já na função proposta por este processo, as redes com poucas violações possuem pontuações maiores que as redes com muitas violações nas primeiras iterações, resultando em serem re-roteadas primeiro.

Em sua segunda versão (LIU et al., 2013), o roteador adotou uma técnica de BLMR (*Bounded-Length Maze Routing*), que possui o intuito de aumentar a velocidade de processamento do *maze routing* através da limitação do campo de busca dentro do grafo, e realizar uma melhor utilização dos recursos através da utilização de menos fios que se tornam redundantes. A Figura 21 mostra o fluxo do roteador NCTU-GR (LIU et al., 2013).

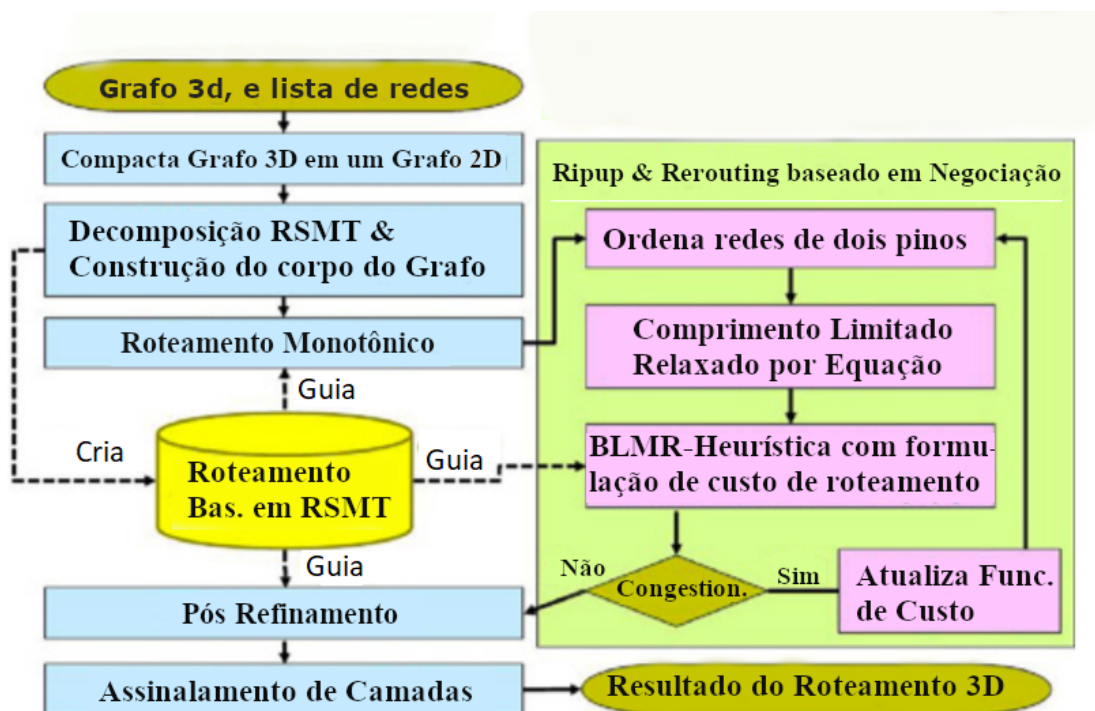


Figura 21: Fluxo do roteador NCTU-GR. Adaptado de (LIU et al., 2013).

O BLMR foi formulado como uma 4-tupla (s, t, G, L) , onde s e t são os pontos de origem e destino, G é o grafo, e L é a limitação do comprimento a ser utilizado, não podendo ser menor que a distância Manhattan computada entre s e t . O objetivo do BLMR é de encontrar o caminho entre os dois pontos, de forma mais ágil e utilizando menos comprimento de fio, porém respeitando a distância Manhattan entre os pontos. O roteador adota o BLMR heurístico, o que não garante uma solução ótima, porém é muito mais rápido do que o BLMR que garante a solução ótima, sendo a grande diferença entre os dois a de que o BLMR heurístico preserva somente um caminho do nodo origem até o nodo destino, descartando os demais, e assim, a maior dificuldade é determinar qual caminho deve ser preservado.

Uma outra melhoria foi implementada na decomposição das redes multi-pinos em

redes de dois pinos pelo roteador. Inicialmente, o roteador realiza a decomposição das redes multi-pinos em redes de dois pinos utilizando RMST, então, o roteamento orientado a RSMT constrói a RSMT para referência, e encoraja que o roteamento de cada sub-rede passe pelas regiões em que a RSMT passa.

Em (LIU et al., 2013) foram implementadas diversas versões do roteador, onde foram utilizadas diversas combinações de técnicas para escolher a melhor configuração. Para a comparação com outros roteadores, foi escolhida a configuração que usava o BLMR heurístico e o roteamento orientado à RSMT para reduzir comprimento de fio e tempo de execução. A Tabela 4 apresenta os resultados obtidos pelo NCTU-GR aplicado nos *benchmarks* do ISPD 2008.

Tabela 4: Resultados obtidos pelo roteador NCTU-GR nos *benchmarks* do ISPD 2008 (LIU et al., 2013).

Benchmark	TOF	WL(e5)	CPU(min)
adaptec1	0	53.04	2.52
adaptec2	0	51.51	0.65
adaptec3	0	129.24	3.40
adaptec4	0	120.53	1.23
adaptec5	0	154.01	6.03
bigblue1	0	56.29	4.09
bigblue2	0	88.66	4.21
bigblue3	0	129.35	1.91
bigblue4	194	225.0	60.23
newblue1	0	45.76	2.37
newblue2	0	74.51	0.64
newblue3	31710	105.36	144.34
newblue4	144	127.7	17.33
newblue5	0	228.68	5.26
newblue6	0	175.49	4.44
newblue7	58	342.90	85.67

4.5 BFG-R

O roteador BFG-R (ROY; MARKOV, 2008) (HU; ROY; MARKOV, 2010) inicialmente realiza a decomposição das redes multi-pinos em redes de dois pinos, então é realizado um roteamento inicial com o intuito de gerar uma solução em uma grade 2D.

Em resumo, existem dois diferentes fluxos dependendo da solução inicial gerada pelo roteador. Caso não existam violações nesta solução, o roteador realiza o assinalamento de camadas, projetando a grade de 2D para 3D, realizando um procedimento posterior para diminuir o comprimento total de fio.

Caso existam violações na solução inicial, então o roteador trata de atualizar os multiplicadores Lagrange afim de atualizar o custo das redes. O segundo passo no caso de

existência de violações é o processo de desfazer e re-rotear. Todas as redes que contêm violações são desfeitas e re-roteadas até que acabem as violações ou o tempo de execução estoure, com base nos custos de suas arestas em uma ordem pré-determinada. Finalmente, o processo de assinalamento de camadas e diminuição do comprimento de fio também é aplicado.

Durante o processo de R&R, é aplicada uma técnica chamada de agrupamento de redes *edge clustering*, onde são encontradas todas as redes que possuem ao menos uma violação, e realizam o agrupamento destas de acordo com a sua localização, desta forma melhorando a utilização da memória. Além disto, outra técnica é aplicada pelo roteador no processo de R&R, fora observado pelos autores que a maior parte do *runtime* era demandado na parte do roteamento de redes com maiores *bounding boxes*, para auxiliar na diminuição deste tempo de processamento, o roteador utiliza uma técnica chamada de bloqueio de redes cíclico (*cyclic net locking* - CNL), onde é realizado um maior controle da quantidade de vezes que as redes maiores serão roteadas. A Figura 22 apresenta o fluxo do roteador BFG-R.

Para a realização de testes e gerar resultados experimentais para comparação com outros roteadores estado-da-arte, o BFG-R, com seu código fonte em C++, foi compilado utilizando o g++ 4.3.2, e rodado em um processador *quad core* 2.83GHz, e 8 GB de memória RAM. Os resultados obtidos pelo roteador aplicado nos *benchmarks* do ISPD 2008 são apresentados na Tabela 5.

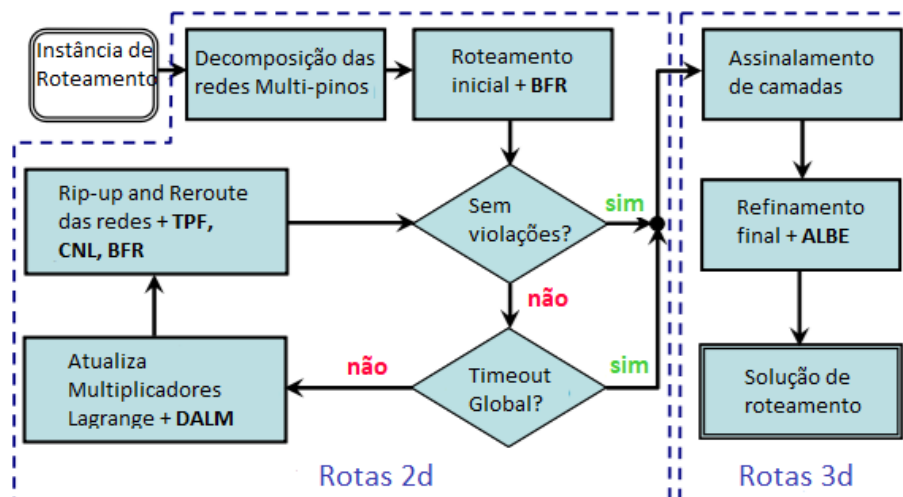


Figura 22: Fluxo do roteador BFG-R. Adaptado de (HU; ROY; MARKOV, 2010).

Tabela 5: Resultados obtidos pelo BFG-R aplicado nos *benchmarks* do ISPD 2008, adaptado de (HU; ROY; MARKOV, 2010).

Benchmark	TOF	WL(e6)	CPU(min)
adaptec1	0	5.43	8.4
adaptec2	0	5.23	3.7
adaptec3	0	13.14	16.0
adaptec4	0	12.16	5.2
adaptec5	0	15.67	15.5
bigblue1	0	5.72	10.2
bigblue2	0	9.11	40.8
bigblue3	0	13.18	20.6
bigblue4	434	23.20	1416.6
newblue1	0	4.68	256.9
newblue2	0	7.57	1.5
newblue3	33900	10.64	1420.9
newblue4	218	13.08	1413.3
newblue5	0	23.30	47.6
newblue6	0	18.01	15.7
newblue7	606	35.21	1421.1

4.6 GT Router

Este trabalho (SIDDIQI; SAIT, 2017) baseou-se em realizar uma re-modelagem no processo de R&R de dois roteadores estado-da-arte abordados anteriormente neste capítulo: BFG-R e NCTU-GR. Os autores identificaram que durante o processo de R&R, os roteadores globais normalmente diminuem o congestionamento total e comprimento total de fio durante as primeiras horas de execução, mas durante o restante do tempo as melhorias são poucas ou até mesmo nenhuma, o que demanda uma análise mais detalhada do processo com o intuito de buscar melhorias.

Foi proposto no trabalho um algoritmo baseado em GT (*Game Theory*) para ser aplicado no processo de R&R, com a capacidade de minimizar o congestionamento total, e o comprimento total de fio nas soluções geradas pelos roteadores. O Algoritmo 4.6 apresenta o algoritmo proposto pelos autores, no algoritmo são recebidas como parâmetro de entrada o conjunto de todas as redes (N), o grafo de roteamento (G) e a solução gerada pelos roteadores (T). A modelagem do processo de R&R então é realizada como um jogo, em primeiro passo é realizada a permutação da ordem em que as redes serão selecionadas para "jogar" suas estratégias. O segundo passo consiste em um *loop* que simula o jogo, dentro deste *loop* a rede selecionada no processo de fila seleciona uma estratégia, e aplica esta estratégia com o objetivo de melhorar sua árvore de abrangência. A simulação do jogo continua até que aconteça um ganho esperado (NE), ou chegue a um tempo limite de execução (SIDDIQI; SAIT, 2017).

Algoritmo 3: Algoritmo proposto para realizar a melhoria no processo de desfazer e re-rotear (SIDDIQI; SAIT, 2017).

```

1 Entrada:  $N$ : conjunto de todas as redes,  $G(V, E)$ : Grafo da grade de roteamento,
   T: Resultado do roteamento global, objetivo={congestionamento-total,
   comprimento-de-fio}
2 Saída: T solução do roteamento global após as melhorias no comprimento total de
   fio ou no congestionamento remanescente.
3 enquanto ( $NE$  ou limite de tempo alcançado) faça
4   Permuta a ordem das redes em  $N$ 
5   para cada rede  $n_i \in N$  faça
6     Seleciona uma estratégia para a rede  $n_i$  que possa ser representada como
        $s * (n_i)$ 
7     Aplica a estratégia  $s * (n_i)$  na rede  $n_i$  para melhorar a sua árvore de
       abrangência  $t_i$ 
8   fim
9 fim
10 retorna ( $T$ )

```

A partir da execução de experimentos, o processo de desfazer e re-rotear remodelado com o algoritmo baseado em GT apresentou ganhos na diminuição de congestionamento total de *benchmarks* do ISPD 2008 difíceis de resolver, e diminuição no comprimento total de fio dos demais *benchmarks*. A Tabela 6 apresenta os ganhos na diminuição de congestionamento total das soluções geradas em *benchmarks* difíceis de resolver pelos roteadores BFG-R e NCTU-GR quando aplicado o algoritmo proposto por (SIDDIQI; SAIT, 2017).

Tabela 6: Ganhos na diminuição de congestionamento total (TO) das soluções dos roteadores BFG-R e NCTU-GR nos *benchmarks* difíceis de resolver do ISPD 2008, aplicando o algoritmo proposto por (SIDDIQI; SAIT, 2017).

BFG-R			Remodelagem R&R GT		
Benchmark	CPU(hora)	TO	+CPU(hora)	TO (depois do GT)	Ganho
newblue3	6	33864	3.33	31752	2074 (6.13%)
newblue4	12	220	9.46	180	38 (17.43%)
newblue7	18	484	5.85	410	66 (13.87%)
bigblue4	9	438	8,98	330	102 (23.61%)
NCTU-GR			Remodelagem R&R GT		
Benchmark	CPU (hora)	TO	+CPU(hora)	TO (depois do GT)	Ganho
newblue3	1.67	37118	0.8	31786	4264 (12.69%)
newblue4	0.56	162	6.04	146	10 (5.13%)
newblue7	1.11	66	1.43	58	0
bigblue4	1.11	214	8.37	176	34 (16.19%)

O trabalho abordado aqui comprovou através de uma remodelagem no processo de R&R, e realização de experimentos envolvendo roteadores modernos, que é possível realizar melhorias nos roteadores estado-da-arte acadêmicos existentes atualmente, tanto

nos resultados de comprimento total de fio para todos *benchmarks* do ISPD 2008 quanto nos resultados de congestionamento total quando se tratando de *benchmarks* de difícil resolução.

4.7 Explorando Paradigma de Computação de Alto Desempenho para Roteamento Global

O trabalho de (HAN et al., 2014) fez a proposta de um roteador híbrido que fizesse o uso de CPU e GPU, onde foi implementada a utilização do paralelismo no nível de dados das GPUs e o paralelismo no nível de *threads* dos múltiplos núcleos das CPUs. O roteador proposto pelos autores fez o uso de ambas CPU e GPU no paralelismo no nível das redes, onde foi proposto um algoritmo *A*-search maze routing* no nível da CPU, e um algoritmo BFS (*Breadth First Search*) similar ao algoritmo de Lee no nível da GPU.

Para realizar o roteamento global, o roteador realiza um fluxo onde inicialmente as redes multi-pinos são divididas em redes de dois pinos a partir da utilização do FLUTE, então é aplicada uma técnica de *edge shifting* em todas as redes, com o intuito de modificar suas topologias iniciais, e finalizando a fase inicial, é realizado um roteamento inicial para gerar o mapa de congestionamento. O roteamento inicial divide as instâncias em todos os cores disponíveis na CPU para utilizar roteamento paralelo, ignorando a dependência de recursos entre as redes para minimizar a sobrecarga no sincronismo dos núcleos. Feito isto, os custos de cada região são calculados com base na separação de duas categorias: com congestionamento e sem congestionamento.

Após a fase do roteamento inicial, o roteador entra em um processo de R&R, onde é aplicado um algoritmo baseado em negociação com o objetivo de melhorar o roteamento das redes que passam por regiões congestionadas, onde a ordem das redes que serão roteadas é determinada com base nas regiões congestionadas. A fase final do roteador é aplicada quando não existe mais congestionamento detectado, neste ponto é aplicado um refinamento final seguido de um assinalamento de camadas para retornar do plano 2-D para o plano multicamadas. A Figura 23 apresenta o fluxo descrito acima seguido pelo roteador, onde as partes com o fundo verde-escuro são apresentadas como as principais contribuições.

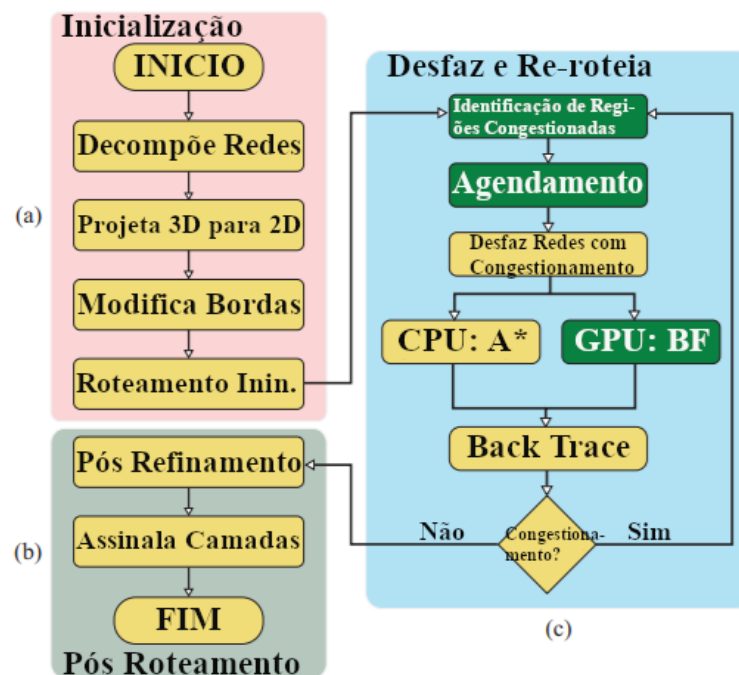


Figura 23: Fluxo do roteador de (HAN et al., 2014), onde (a) é o processo de inicialização, (b) a fase pós-roteamento, e (c) o processo de desfazer e re-rotear aplicado pelo roteador. Adaptado de (HAN et al., 2014).

Para que o roteador execute o procedimento descrito, ele faz uso de uma técnica chamada NLC (*Net Level Concurrency*), a técnica de NLC consiste em utilizar um escalonador que cria grupos de redes que podem ser processadas em paralelo. O escalonador de redes trata de produzir tarefas de roteamento concorrentes com base na dependência entre as redes. A partir da identificação das redes concorrentes acontece então a divisão entre CPU ou GPU (de acordo com os critérios estabelecidos no roteador) para a realização do roteamento das redes que podem ser processadas em paralelo sem problemas de dependência.

A dependência entre as redes é identificada a partir dos recursos que as mesmas compartilham dentro de uma região, a Figura 24 apresenta um exemplo de identificação de dependência entre 7 redes em uma grade, na Figura 24 (a) é possível analisar que somente as redes A e D poderiam ser roteadas ao mesmo tempo, dado que as demais redes dependem da rede D, e na Figura 24 (b) é possível analisar que após acontecer o roteamento da rede D, as redes C, B, F e G podem ser roteadas ao mesmo tempo por não apresentarem dependências.

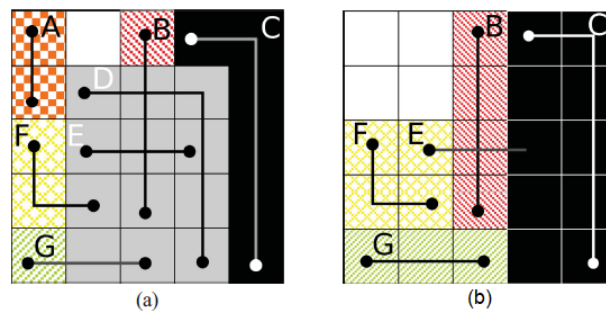


Figura 24: Grade de rotamento e redes a serem roteadas, (a) primeira iteração, somente as redes A e D podem ser roteadas ao mesmo tempo, (b) segunda iteração, as redes C, B, F e G podem ser roteadas ao mesmo tempo. Adaptado de (HAN et al., 2014).

A Tabela 7 apresenta os resultados obtidos pelo roteador nas métricas de comprimento total de fio e tempo de execução fazendo o uso da CPU de 4 núcleos e da GPU quando aplicado nos *benchmarks* do ISPD 2008. É possível analisar que o ganho em diminuição de tempo de execução utilizando paralelismo no nível de CPU + GPU não é tão grande comparado aos resultados utilizando apenas o paralelismo no nível de CPU, além de acontecerem pequenas perdas nos ganhos na diminuição do comprimento total de fio. Na Tabela, os resultados representados com -"são experimentos que não foram aplicados nos *benchmarks* sem solução livres de congestionamento, dados como *benchmarks hard-to-route*.

Tabela 7: Resultados do roteador aplicado nos *benchmarks* do ISPD 2008 com o uso somente de paralelismo a nível de CPU, e CPU + GPU. Adaptado de (HAN et al., 2014).

Benchmark	CPU (Quatro Núcleos)		CPU (Quatro Núcleos) + GPU	
	WL(e6)	CPU(min)	WL(e6)	CPU(min)
adaptec1	5.43	2.90	5.44	2.07
adaptec2	5.29	0.70	5.30	0.63
adaptec3	13.10	3.53	11.31	3.05
adaptec4	12.40	0.95	12.40	0.65
adaptec5	15.50	9.98	15.50	8.12
bigblue1	5.63	3.10	5.63	2.86
bigblue2	9.10	2.31	9.05	2.09
bigblue3	13.00	1.09	13.00	1.01
bigblue4	-	-	23.29	~60
newblue1	4.70	2.87	4.70	1.88
newblue2	7.79	0.66	7.81	0.65
newblue3	-	-	10.85	~60
newblue4	-	-	13.89	~60
newblue5	23.80	5.38	23.80	4.42
newblue6	18.00	4.12	18.00	3.78
newblue7	-	-	35.23	~60

5 METODOLOGIA

Este capítulo apresenta a metodologia aplicada neste trabalho. Para atender ao nosso objetivo de realizar melhorias em um roteador global estado-da-arte de código fonte aberto, foi aplicada uma metodologia que pode ser dividida de forma geral em quatro principais etapas, e são elas:

1. Análise inicial do roteador, das métricas utilizadas para comparação de resultados e algoritmos aplicados por ele;
2. Levantamento dos resultados gerados pelo roteador sem modificação alguma em sua estrutura;
3. Análise avançada do roteador e de sua estrutura geral; e
4. Aplicação de melhorias guiadas nos experimentos com o intuito de trazer ganhos no desempenho geral ou nos resultados gerados pelo roteador.

Os experimentos foram executados em um computador com o Sistema Operacional Linux 64-bit, e um processador Intel Core i7-7700HQ, com 16GB de memória RAM 2.80 GHz. A organização de cachê é composta por um cachê L1 com 256 KiB de instrução e L1 dados 128KB(4x32 KB) *8-way set associative*. Os cachês L1, L2 e L3 adotam políticas de memória write-back. L2 possui 1 MB (4x256 KB) *4-way set associative*, e L3 possui 6 MB (4x1.5 MB) *12-way set associative*.

Os *benchmarks* adotados para a realização dos experimentos foram os *benchmarks* do ISPD2008 (ISPD2008, 2017), alguns dos *benchmarks* desta competição foram selecionados com o pré requisito de não possuírem impedimentos nas soluções geradas pelo roteador, de forma a possibilitar um comparativo para os experimentos realizados. Desta forma, os *benchmarks hard-to-route* não foram utilizados, os *benchmarks* utilizados são apresentados na Tabela 8.

5.1 Análise inicial do roteador

A partir da análise realizada sobre a estrutura do roteador NTHU (Figura 15), foi possível constatar que seu processo em estágio inicial é dado pela decomposição inicial

das redes multi-pinos em redes de dois pinos, seguido de um roteamento em forma de "L" para gerar o primeiro mapa de congestionamento. No estágio principal, é realizada a identificação das regiões congestionadas, e as redes de dois pinos são marcadas para re-rotear, a partir disto, é realizado um re-roteamento destas redes. No re-roteamento das redes identificadas como congestionadas, é aplicado o roteamento monotônico, seguido de um processo adaptativo de *maze routing multi-source multi-sink*. O estágio principal que foi descrito é repetido até que todas as redes sejam processadas. Então, é identificada a existência ou não de congestionamento remanescente. Caso não haja congestionamento, ou depois de muitas iterações (número de iterações definido pelo usuário), o roteador avança para a etapa de processamento final.

No estágio do processamento final, é realizado um processo de refinamento seguido do assinalamento das camadas, gerando assim o resultado do roteamento. O processo descrito foi apresentado na Figura 15 no Capítulo 4.

Através da execução de experimentos utilizando o roteador NTHU-Route (LEE; CHANG; WANG, 2010) para resolver *benchmarks* que estão fora do escopo de serem de difícil solução, como o *newblue5*, foi possível analisar que, nos primeiros momentos, o roteador apresenta uma grande redução no congestionamento total das redes (*TO*). Após esta diminuição, o processo leva a maior parte do tempo para realizar poucas melhorias posteriores. A Figura 25 apresenta o gráfico gerado a partir da execução do *benchmark newblue5*, mostrando o tempo de execução e a diminuição do congestionamento total durante a janela de tempo.

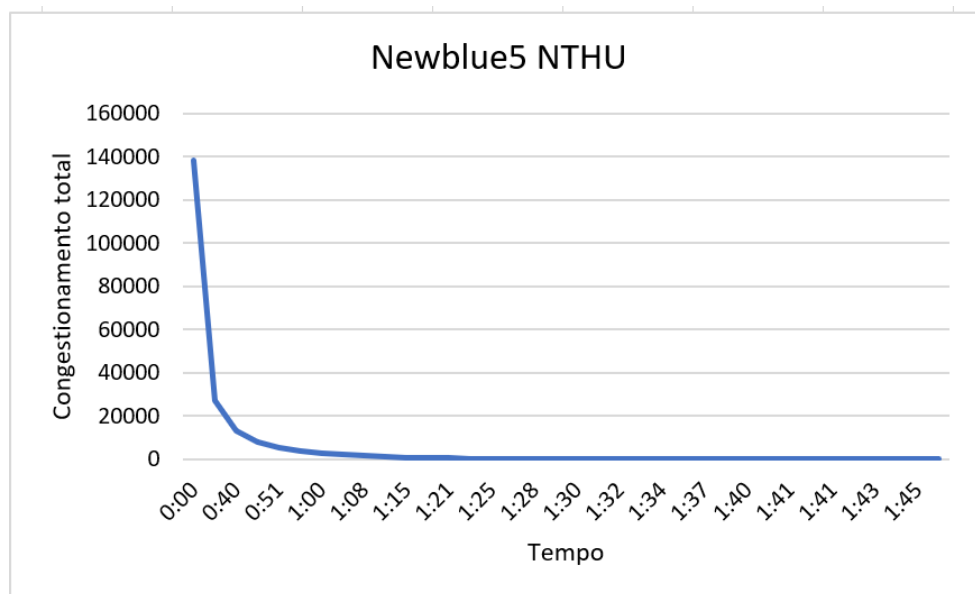


Figura 25: Diminuição no congestionamento total vs tempo de execução pelo roteador NTHU-Route utilizando o *benchmark newblue5*.

O mesmo comportamento acontece quando falamos de *benchmarks* de difícil resolução, como por exemplo o *newblue4*. Após os primeiros momentos em que é gerado

o mapa de congestionamento e realizadas as primeiras iterações, o congestionamento total cai drasticamente. Entretanto, nas horas posteriores, as melhorias realizadas pelo processo de desfazer e re-rotear são mínimas, ou mesmo nulas. A Figura 26 apresenta o gráfico gerado a partir da execução do *benchmark* de difícil resolução *newblue4*, apresentando tempo de execução e congestionamento total durante a execução, sendo o eixo vertical o congestionamento total, o eixo horizontal o tempo, e as caixas de texto o tempo seguido do congestionamento total naquela janela de tempo. É possível verificar a partir da execução deste *benchmark* que depois de algumas iterações, os ganhos na diminuição total de congestionamento das redes é praticamente nulo.

Este comportamento envolvendo o congestionamento total e o tempo de execução foi levado em consideração para a aplicação de uma técnica envolvendo a aplicação de um limite de aceitação do congestionamento, que será explicado na sequência deste texto.

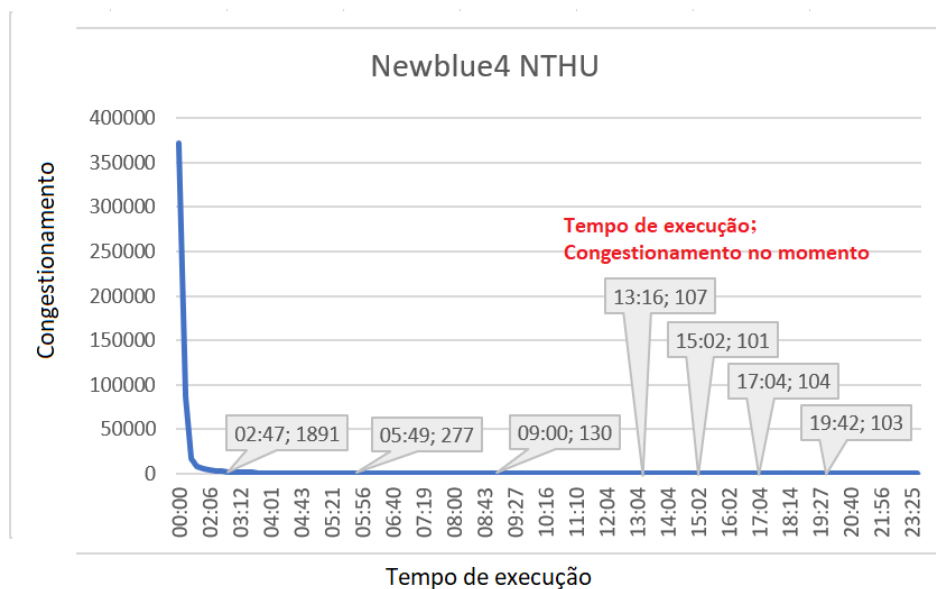


Figura 26: Diminuição no congestionamento total vs tempo de execução pelo roteador NTHU-Route utilizando o *benchmark newblue4*.

5.2 Levantamento dos resultados para comparação

A Tabela 8 apresenta os *benchmarks* selecionados para a realização dos experimentos, e as características dos mesmos. Na tabela é possível visualizar o *benchmark*, a quantidade de redes (Redes), o tamanho das grades (Grade), e o número de obstáculos.

Tabela 8: Características dos *benchmarks* do ISPD08 selecionados para uso nos experimentos (ISPD2008, 2017).

Benchmark	Redes	Grades	Número de Obstáculos
adaptec1	219794	227x227	84731
adaptec2	260159	424x424	207257
adaptec3	466295	774x779	770558
adaptec4	515304	774x779	618473
adaptec5	867411	465x468	264323
bigblue1	282974	227x227	16856
bigblue2	576816	468x471	192068
newblue2	463213	557x463	449965
newblue5	1257555	637x640	449189
newblue6	1286452	463x464	174709

De forma a buscar um comparativo no mesmo contexto computacional para os experimentos realizados, após a seleção dos *benchmarks* a serem usados nos experimentos, foi realizada uma execução sem modificações na estrutura geral do roteador. Inicialmente, os *benchmarks* foram executados com as configurações padrão na versão do código disponibilizada, e os resultados foram guardados para futuras comparações com os experimentos a serem realizados.

A Tabela 9 apresenta os resultados obtidos para os *benchmarks* selecionados, na tabela é possível visualizar o tempo total levado para chegar ao resultado final (Tempo), o tempo levado no estágio de refinamento (Tempo Ref.), e o comprimento total de fio nos resultados (WL). Todos os resultados obtiveram um resultado de congestionamento 0.

Tabela 9: Resultados envolvendo as métricas de comprimento total de fio, tempo para execução total e tempo para execução da fase de refinamento do roteador NTHU route.

Benchmarks	Resultados do NTHU-R		
	WL	Tempo(s)	Tempo Refin.(s)
adaptec1	5516103	660	11.40
adaptec2	5237170	154.8	10.10
adaptec3	13127908	571.8	29.60
adaptec4	12203734	159	29.39
adaptec5	15845294	1425	33.61
bigblue1	5887720	1318.8	13.39
bigblue2	7996087	993	23.57
newblue2	7594019	73.8	14.63
newblue5	23352537	1239	55.48
newblue6	18215066	2409.6	48.00

5.3 Análise avançada do roteador

Foi realizado um *profiling* mais detalhado no roteador utilizando as ferramentas Gcov (GCOV, 2019) e Valgrind (VALGRIND, 2019), para obter uma análise mais detalhada do comportamento do roteador. Nesta análise, foram levadas em conta as execuções dos trechos de código com o intuito de entender as partes mais e menos críticas.

Dentro do contexto da análise avançada, três principais ações foram tomadas:

- Levantamento de código morto ou passível de pequenas melhorias, como instruções de decisão que nunca entrariam em determinado caso;
- Análise das chamadas de funções no fluxo geral do roteador, para conhecimento de possíveis partes a serem otimizadas;
- Busca do entendimento do impacto dos parâmetros dos algoritmos nos resultados de roteamento, e a possível aplicação de métodos de paralelismo na estrutura atual do roteador.

A ferramenta de caracterização Gcov foi utilizada para entender a execução do roteador. Dentro deste contexto, o objetivo foi o de identificar quais blocos de código eram mais, menos, e nunca executados. A ferramenta de caracterização Valgrind foi utilizada através de uma sub-ferramenta chamada Callgrind, e o intuito principal de usar estas ferramentas foi o de entender o fluxo do roteador no sentido de execução de ponta a ponta, e também quais partes poderiam ser otimizadas. A Figura 27 apresenta a caracterização inicial realizada no roteador sem nenhuma alteração em sua estrutura. A figura apresenta as funções que possuem um custo maior, e seriam boas candidatas a otimização, além da quantidade de vezes que determinada função é chamada dentro da janela de execução para o *benchmark bigblue2*.

Os algoritmos utilizados no estágio principal do roteador *multi-sink multi-source maze routing* e *monotonically routing* possuem grande dependência de dados. Esta dependência trouxe para os primeiros experimentos envolvendo paralelismo resultados negativos. Sem realizar modificações nos algoritmos implementados pelo roteador atualmente não seria possível obter ganhos significativos no tempo de execução aplicando técnicas de paralelismo.

Para entender o impacto dos parâmetros aplicados aos algoritmos mencionados, foi criado um *script* em Ruby com o objetivo de estressar as execuções, alternando os parâmetros de entrada e analisando as saídas dos resultados. Basicamente, para cada um dos *benchmarks* testados, o programa tratou de mudar parâmetros como a expansão da caixa-delimitadora (*bounding-box*) para busca e conexão de dois pinos no algoritmo *multi-sink multi-source maze routing*, e a aceitação de congestionamento no estágio principal do roteador (de forma que em uma segunda etapa, o congestionamento aceito pudesse ser resolvido).

Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000001090	ld-2.27.so
100.00	0.00	1	_start	ISPD2008-NTHU-R-CodeRelease-Updated
100.00	0.00	1	(below main)	libc-2.27.so: libc-start.c
100.00	0.00	1	main	ISPD2008-NTHU-R-CodeRelease-Updated
74.05	0.03	1	NTHUR::Construct_2d_tree:...	ISPD2008-NTHU-R-CodeRelease-Updated
44.25	0.00	13	NTHUR::Route_2pinnets::ro...	ISPD2008-NTHU-R-CodeRelease-Updated
43.96	0.24	13	NTHUR::RangeRouter::spec...	ISPD2008-NTHU-R-CodeRelease-Updated
41.55	0.06	6 971 299	NTHUR::RangeRouter::rang...	ISPD2008-NTHU-R-CodeRelease-Updated
39.86	15.59	59 315	NTHUR::Multisource_multis...	ISPD2008-NTHU-R-CodeRelease-Updated
25.11	0.06	1	NTHUR::Construct_2d_tree:...	ISPD2008-NTHU-R-CodeRelease-Updated
21.60	5.85	747 580	NTHUR::Construct_2d_tree:...	ISPD2008-NTHU-R-CodeRelease-Updated
21.36	0.00	1	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
21.31	0.00	1	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
21.20	0.01	373 790	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
17.32	0.84	97 233 409	boost::heap::pairing_heap...	ISPD2008-NTHU-R-CodeRelease-Updated
16.73	0.03	373 790	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
16.11	1.05	28 068 989	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
14.82	7.38	99 621 883	boost::heap::pairing_heap...	ISPD2008-NTHU-R-CodeRelease-Updated
13.38	3.69	10 469 746	NTHUR::Layer_assignment:...	ISPD2008-NTHU-R-CodeRelease-Updated
10.82	5.53	756 171 586	std::_Function_handler<voi...	ISPD2008-NTHU-R-CodeRelease-Updated
7.44	7.44	265 720 946	boost::heap::pairing_heap...	ISPD2008-NTHU-R-CodeRelease-Updated
6.68	6.68	1 512 343 172	NTHUR::EdgePlane<int>::e...	ISPD2008-NTHU-R-CodeRelease-Updated
6.05	0.15	331 441 638	operator delete(void*)	libstdc++.so.6.0.25

Figura 27: Caracterização das funções mais custosas em termos de execução realizada com a ferramenta Callgrind para o roteador NTHU-Route antes de realizar modificações no *benchmark bigblue2*.

A aplicação de um limiar de aceitação de congestionamento contribui principalmente para a diminuição do tempo de execução geral do roteador. Durante o estágio principal do roteador, o mesmo tenta encontrar um resultado já livre de congestionamento, de forma que o estágio de refinamento somente realize pequenas melhorias no resultado já aceitável gerado pelo estágio principal. O estágio de refinamento aplica uma forma diferente de roteamento e em partes menores do espaço de roteamento, sem levar em consideração algumas constantes que são aplicadas no estágio principal, desta forma conseguindo resolver pequenos congestionamentos de forma mais rápida. Porém, dependendo da quantidade de congestionamento aceito, a fase de refinamento pode não conseguir lidar e gerar um resultado válido, desta forma invalidando a execução.

A partir da análise das execuções de *benchmarks* roteáveis e dos testes realizados com a aceitação dos resultados com congestionamento, foi atribuído como constante a *flag* de limiar para a aceitação de resultados não completos ainda na fase principal. A Figura 28 apresenta o gráfico que detalha uma execução de um *benchmark* roteável (*adaptec1*). Nesta figura, podemos perceber que o roteador consegue achar uma solução com pouco congestionamento ainda nas primeiras iterações, e que, após isto, ele pode ficar preso por um grande número de iterações até ser liberado para a próxima fase. Em um determinado momento durante o fluxo de execução, é encontrado um resultado com pouco congestionamento, porém, logo em seguida, (destacado na imagem por uma seta) o valor aumenta consideravelmente, desta forma estendendo o tempo total de execução.



Figura 28: Congestionamento total x número da iteração durante a execução do *benchmark adaptec1*,

5.4 Aplicação de melhorias

Podemos dividir a aplicação das melhorias propostas em duas principais propostas: (1) Limiar de congestionamento, e (2) Paralelismo. Todas as melhorias e experimentos para estas aplicações foram aplicadas dentro do estágio principal e do estágio de refinamento do roteador, entrando dentro deste contexto o processo de desfazer e re-rotear, o refinamento de resultados, e o processo de assinalamento de camadas.

1 - Limiar de congestionamento

Após a análise mais detalhada do fluxo do roteador, foi percebido que os algoritmos aplicados para solucionar o roteamento no estágio principal ficavam presos com pequenos congestionamentos por um tempo considerável em alguns casos (conforme é possível acompanhar na Figura 28), e que o estágio de refinamento, por aplicar um algoritmo que não considera constantes que são exploradas no estágio principal, consegue lidar com os congestionamentos de forma mais rápida.

Dito isto, após executar os *benchmarks* selecionados para os testes de forma incremental, aumentando a cada iteração em cinco a aceitação de congestionamento a partir do *script* criado para a execução contínua do roteador, foi selecionado um valor limiar que se enquadrava no limite entre diminuição de tempo de execução, e geração de solução válida. Após realizações de testes, constatou-se que aceitação de um congestionamento acima de 25 na fase principal acarretava na não solução do *benchmark* no fim. Então, por uma questão de margem de segurança, o limiar de congestionamento foi definido como 20 para a geração dos resultados finais.

2 - Paralelismo

Inicialmente, a ideia do uso de paralelismo para melhorar o desempenho do roteador foi estudada para ser aplicada a nível algorítmico. Os principais algoritmos que o NTHU Route faz uso para gerar soluções de roteamento possuem grande dependência de dados. Assim, para chegar à uma solução em que fosse possível obter grandes ganhos, seria necessário re-estruturar ou implementar algoritmos novos, somente aproveitando a estrutura do roteador. Os primeiros testes envolvendo o paralelismo foram aplicados em blocos menores de código, e não nas principais soluções algorítmicas do roteador. Foram selecionados trechos de código que eram passíveis de paralelização sem comprometer a lógica ou o fluxo geral do roteador, e então através do uso da biblioteca OpenMP (OPENMP, 2019), os trechos de código foram paralelizados a nível de *threads*.

A exploração de paralelismo utilizando uma biblioteca em C++ voltada para este fim foi aplicada principalmente para responder a duas perguntas:

- Podemos aplicar paralelismo na estrutura atual do roteador, sem tornar os algoritmos que realizam as principais funções no roteamento obsoletos?
- Se aplicarmos paralelismo em blocos menores de código que não envolvem os principais fluxos, é possível gerar melhorias?

Para selecionar onde as funcionalidades da biblioteca seriam aplicadas, foi considerada a dependência de dados entre os algoritmos e as fases de roteamento, e também o possível ganho real no tempo de execução, caso fosse aplicado paralelismo em alguns dos processos do roteador.

Para validar estas aplicações, foram separados blocos de código menores dentro da fase principal e da fase de refinamento do roteador. Os blocos menores selecionados dentro do estágio principal foram sub-algoritmos, como buscas e/ou outras operações em vetores, os laços de repetição para que se encarregavam de buscar por um caminho válido para conexões de dois pinos, e também o método para alocar as grades de roteamento.

Para a fase de refinamento, todas as modificações foram aplicadas em procedimentos envolvendo vetores. Todas as modificações tiveram o intuito de entender se seria possível gerar ganhos a partir delas. As modificações implicaram na modificação do roteador para utilizar as *flags* que configuram o uso de paralelismo através do OpenMP nos blocos selecionados de código. O Anexo A deste trabalho apresenta exemplos de uso do OpenMP para a paralelização de estruturas de repetição *while* e *for*.

Os resultados envolvendo as melhorias aplicadas dentro do contexto de parâmetros (aceitação de congestionamento) e paralelismo (uso da biblioteca OpenMP para paralelizar blocos de código) são apresentados no Capítulo 6.

6 RESULTADOS

Experimentos após as modificações no roteador NTHU Route foram realizados com o intuito de mensurar os resultados de tempo total de execução, e avaliar o aspecto no comprimento total de fio. Os resultados são apresentados em três sub-sessões:

1. Experimentos realizados com base na inserção de um limiar de congestionamento no estágio principal do roteador;
2. Experimentos realizados após a aplicação de paralelismo em blocos menores do código no estágio principal (*Main stage*) e no estágio de refinamento do resultado (*Refinement stage*), e
3. Experimentos realizados após juntar as duas melhorias antes citadas, juntando a solução com a inserção do limiar de congestionamento e a aplicação de métodos de paralelismo usando a biblioteca OpenMP no estágio principal e de refinamento do roteador.

6.1 Limiar de Congestionamento

Na Tabela 10 são apresentados os resultados das execuções do roteador NTHU Route sem nenhuma alteração, e os resultados após a inserção o limiar de congestionamento para aceitar valores de congestionamento no estágio principal do roteador. Podemos ver ganhos no tempo de execução total em todos os *benchmarks* testados, e para o comprimento total de fio, podemos acompanhar pequenos ganhos na maioria dos casos testados.

A Figura 29 mostra a comparação entre o comprimento total de fio e o tempo que o roteador original leva para resolver os *benchmarks*, e o comprimento total de fio e o tempo que o roteador leva para resolver os *benchmarks* após a inserção do limiar de congestionamento no estágio principal do fluxo do roteador. Em média, com a adição da *flag* para pular de fase com aceitação de congestionamento, foi possível diminuir em 9% o tempo total que o roteador leva para encontrar resultados válidos. Os melhores resultados foram nos *benchmarks adaptec1* e *adaptec5*, com 20,7% e 17% de redução no tempo total de execução, respectivamente. O caso com menor aumento de desempenho foi encontrado

no *benchmark bigblue1*, com 4,6% de melhoria. Em termos de alterações no *wirelength*, os *benchmarks adaptec1* e *adaptec5* também apresentaram as maiores alterações, com uma redução de 2,3% e 1,5%, respectivamente. De forma geral, uma redução menor que 0,1% foi observada nos outros circuitos *benchmarks*, salvo no *newblue2* que teve um incremento também menor que 0,1%.

Tabela 10: Redução nos resultados gerados pelo roteador NTHU após a adição da flag de aceitação de congestionamento para os *benchmarks* testados quando comparados como resultado no roteador NTHU original.

Benchmarks	NTHU-R Original		Aceitação Congestion.	
	WL	Tempo(s)	WL	Tempo(s)
adaptec1	5516103	660	5366404	523
adaptec2	5237170	155	5236763	138
adaptec3	13127908	572	13127287	523
adaptec4	12203734	159	12203640	144
adaptec5	15845294	1425	15614366	1182
bigblue1	5887720	1319	5886156	1257
newblue2	7594019	74	7594401	68

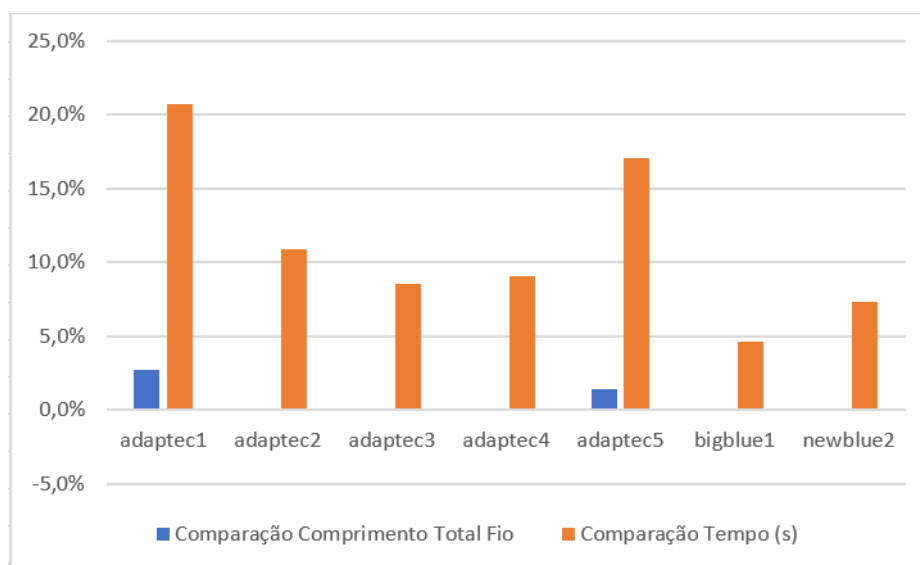


Figura 29: Comparação entre o comprimento total de fio e o tempo que o roteador original leva para chegar à um resultado para os *benchmarks* aplicados, e após a aplicação da *flag* de aceitação de congestionamento no estágio principal do do roteador.

6.2 Paralelismo

A Tabela 11 apresenta os resultados gerados a partir do roteador após aplicar técnicas de paralelismo no estágio de refinamento comparados aos resultados originais do roteador sem modificações. São apresentados o comprimento total de fio, e também o tempo que o

roteador levou para finalizar a parte de pós-processamento/refinamento. A versão modificada do roteador apresenta resultados de comprimento total de fio próximos do roteador NTHU original, com poucas melhorias nesta métrica para a maior parte dos *benchmarks* (a melhoria média foi de 0,05%). Os melhores resultados foram encontrados no *benchmark adaptec1*, com 2,97% de melhoria no comprimento total de fio, e no *benchmark bigblue1*, com 5,14 %.

Além disso, foi possível diminuir o tempo total tomado na fase de pós-processamento/refinamento. Em média, foi possível reduzir o tempo que a fase de refinamento leva para processar uma saída em 9,7%, sendo os melhores casos encontrados nos benchmarks *bigblue1* e *adaptec4*, com 20,4% e 15,4% de redução no tempo para chegar à um resultado, respectivamente. O *benchmark adaptec2* apresentou os menores resultados, mesmo assim, obteve 5,8% de melhoria no tempo. A Figura 30 apresenta um gráfico comparando os resultados do roteador original, e o resultado após as aplicações das melhorias no estágio de pós-processamento/refinamento.

Tabela 11: Redução nos resultados gerados pelo roteador NTHU após a inserção de paralelismo em blocos menores de código na fase de refinamento quando comparados como resultado no roteador NTHU original.

Benchmarks	NTHU-R original		Nossa versão	
	WL	Tempo Refinamento (s)	WL	Tempo Refinamento (s)
adaptec1	5516103	11.40	5352269	9.85
adaptec2	5237170	10.10	5237931	9.51
adaptec3	13127908	29.60	13138825	26.91
adaptec4	12203734	29.39	12203539	24.85
adaptec5	15845294	33.61	15539837	30.35
bigblue1	5887720	13.39	5585291	10.65
newblue2	7594019	14.63	7590191	13.61

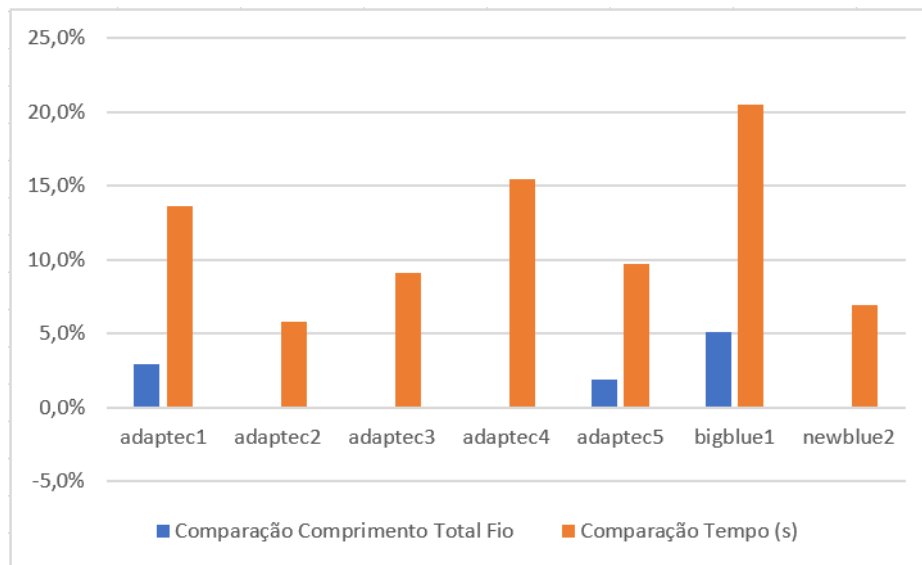


Figura 30: Comparação entre o comprimento total de fio e o tempo que o roteador leva para completar a fase de refinamento antes e depois da aplicação das melhorias.

Para o estágio principal do roteador, ao aplicar a mesma estratégia de paralelismo obtivemos aumento no tempo total de execução. O principal algoritmo utilizado no roteador atualmente possui um comportamento de tentar exaustivamente encontrar soluções livres de congestionamento já no estágio principal, e a tendência foi do roteador ficar um tempo consideravelmente maior tentando encontrar uma solução livre de congestionamento. Desta forma, se mantendo preso por grande parte dentro do estágio principal. Para todos os *benchmarks* em que aplicamos os experimentos, obtivemos aumento no tempo de execução. Dado que os experimentos envolvendo a fase principal e a fase de refinamento foram aplicados em conjunto, o comprimento total de fio das soluções e o tempo de processamento da fase de refinamento são os mesmos apresentados na Tabela 11.

A Tabela 12 apresenta a comparação do tempo levado para resolver os *benchmarks* entre o roteador original, e o tempo levado para resolver os *benchmarks* após as nossas modificações, ressaltando as métricas de comprimento total de fio e tempo total para encontrar as soluções. O pior caso foi o *benchmark bigblue1*, o qual obteve um aumento de 1,92x no tempo de execução, porém o comprimento total de fio para este *benchmark* foi melhorado em 5,14%. Em média, o aumento do tempo de execução geral foi de 116%.

Tabela 12: Comparação entre os resultados gerados pelo roteador NTHU original, e os resultados após a adição de paralelismo em blocos menores no estágio principal dentro do fluxo do roteador.

Benchmarks	NTHU-R original		Nossa versão	
	WL	Tempo (s)	WL	Tempo (s)
adaptec1	5516103	660	5352269	1428
adaptec2	5237170	155	5237931	417
adaptec3	13127908	572	13138825	1161
adaptec4	12203734	159	12203539	237
adaptec5	15845294	1425	15539837	3678
bigblue1	5887720	1319	5585291	5180
newblue2	7594019	74	7590191	122

6.3 Paralelismo e Limiar de Congestionamento juntos

Para os últimos experimentos, foram aplicadas as duas modificações antes apontadas em conjunto, a limiar de aceitação de congestionamento foi aplicada na fase principal do roteador, e o processamento paralelo foi aplicado principalmente no estágio de refinamento. A Tabela 13 apresenta a comparação entre a execução do roteador original e a execução após todas as mudanças de parâmetros e aplicação de soluções envolvendo a biblioteca OpenMP. Nesta tabela são apresentados os resultados de comprimento total de fio (WL), tempo total de execução (Tempo) e tempo total para execução da fase de refinamento (Tempo Ref.).

Para estes experimentos, a biblioteca OpenMP foi utilizada para realização de paralelismo no nível de *threads* em alguns dos laços de repetição para em um nível não-algoritmico, buscando a melhoria do desempenho geral. O *flag* limiar de aceitação de congestionamento foi setado como 20, e o parâmetro de expansão de caixa-delimitadora no algoritmo *core* do estágio principal como 2. A escolha destes parâmetros foi definida pela realização de várias execuções para comparações via *scripts*, os quais buscaram testar parâmetros variantes em todos os *benchmarks*, de modo a encontrar um valor médio que se adaptasse bem em todos os casos.

No caso do uso do OpenMP para a aplicação de execuções *multi-thread*, aplicamos nas estruturas de repetição para que não obtiveram impactos negativos a nível algorítmico. Nosso melhor resultado envolvendo melhoria no comprimento total de fio foi no *benchmark newblue5*, onde a diferença foi de 7,7%. Nosso melhor resultado envolvendo o tempo de execução foi no *benchmark adaptec1*, onde a melhoria no comprimento total de fio foi de 2,7%, e a melhoria no tempo de execução total foi de 19,7%. O pior cenário foi no *benchmark bigblue1*, onde obtivemos 0,4% de aumento no comprimento total de fio, e 52% de aumento no tempo de execução total. Para o tempo levado na fase de refinamento, o melhor resultado foi a diminuição de 18,3% no *benchmark adaptec4*.

Na média de todos os *benchmarks* testados, a melhoria no comprimento total de fio

foi de 1,75%, a penalidade no tempo de execução foi de 0,04%, e a melhoria no tempo de execução da fase de refinamento foi de 4,7%. Na Tabela 13 os resultados são melhor detalhados, e a Figura 31 ilustra os ganhos e perdas. Como é possível acompanhar na Figura 31, foram obtidos resultados negativos envolvendo a diminuição do tempo no estágio de refinamento em apenas um caso (*benchmark bigblue2*). Além disso, o tempo para completar o roteamento aumento nos *benchmarks adaptec5* e *bigblue1*. Nos demais casos, o tempo para completar o roteamento e o tempo para completar o refinamento foi positivo.

Tabela 13: Comparação entre os resultados gerador pelo roteador NTHU original, e os resultados gerados pelo roteador após a aplicação de todas as modificações nos parâmetros, aplicação da *flag* para aceitação de congestionamento e aplicação de paralelismo a nível de *multi-thread* utilizando a biblioteca OpenMP.

Benchmarks	NTHU-R original			NTHU-R depois de todas modificações		
	WL	Tempo(s)	Tempo Ref.(s)	WL	Tempo(s)	Tempo Ref.(s)
adaptec1	5516103	660	11.4	5364875	529	10.57
adaptec2	5237170	155	10.1	5234881	145	9.93
adaptec3	13127908	572	29.6	13115690	546	26.84
adaptec4	12203734	159	29.39	12202228	143	24.9
adaptec5	15845294	1425	33.61	15805585	1578	30.66
bigblue1	5887720	1319	13.39	5915261	2004	11.59
bigblue2	7996087	993	23.57	7921938	893	29.76
newblue2	7594019	74	14.63	7590223	73	13.73
newblue5	23352537	1239	77.40	21546474	1180	72.40
newblue6	18222064	2410	48.00	17107562	2250	47.00

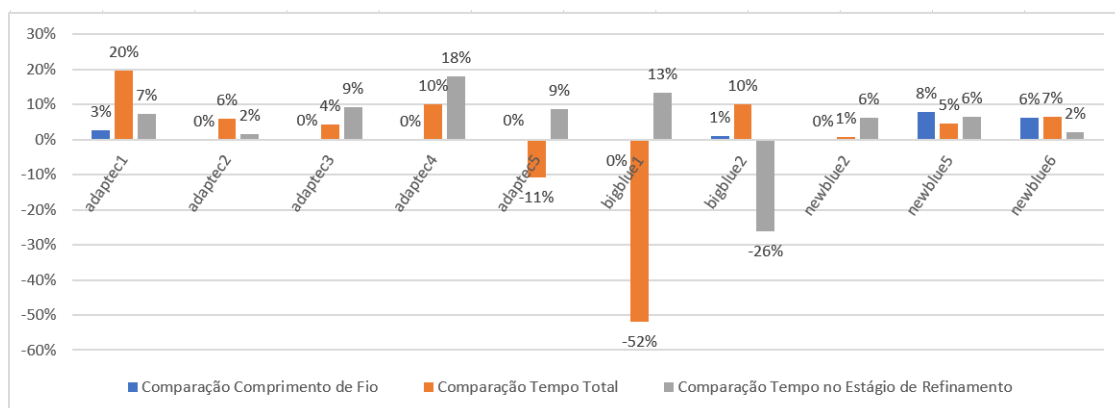


Figura 31: Comparação entre o comprimento total de fio, tempo total de execução do roteador e tempo no estágio de refinamento antes e depois da aplicação de nossas modificações.

7 CONCLUSÃO

Este trabalho disserta sobre a síntese física de circuitos integrados com um enfoque na etapa de roteamento global. Algoritmos e técnicas que normalmente são aplicados na resolução de problemas de roteamento foram abordados. Trabalhos recentes realizaram a implementação de melhorias em roteadores globais estado-da-arte. Da mesma forma, neste trabalho, buscou-se, através do estudo de roteadores globais modernos, a escolha de um roteador de código fonte aberto para a realização de investigações mais aprofundadas, e consequente aplicação de melhorias no mesmo.

O Roteador NTHU Route foi o ganhador do concurso de roteamento ISPD2008, e atualmente uma de suas versões foi disponibilizada como um roteador global de código fonte aberto. Ele possui como princípios, facilitar entendimento do fluxo de roteamento global de circuitos VLSI, e disponibilizar o código para aplicação de melhorias e estudos de algoritmos. Aponta-se como as principais contribuições deste trabalho a análise e implementação de melhorias no roteador NTHU. Através da análise deste roteador, foi evidenciada a possibilidade da realização de melhorias aplicadas através dos parâmetros utilizados para o roteamento nos algoritmos implementados pelo roteador atualmente, e também na aplicação de paralelismo em partes do roteador sem modificações no comportamento atual dos algoritmos.

Como contribuições envolvendo parâmetros adotados pelos algoritmos, após a análise dos estágios principais e de refinamento do fluxo geral do roteador, e, compreendida a capacidade do estágio de refinamento de lidar com congestionamentos não resolvidos pelo estágio principal, foi escolhido um valor limiar para a aceitação de um valor de congestionamento ainda no estágio principal, sendo este refinado e resolvido no estágio de refinamento. Com a aplicação deste limiar, foi possível obter ganhos no tempo de execução total para chegar em resultados válidos para todos os *benchmarks* testados.

Como contribuições envolvendo paralelismo, foram introduzidos blocos de paralelismo em partes do código utilizando a biblioteca OpenMP. Dentro do fluxo geral do roteador, foram modificadas partes do código dentro do estágio principal, e também do estágio de refinamento. No estágio de refinamento foi possível gerar melhorias satisfatórias através da aplicação de paralelismo em operações realizadas dentro deste estágio.

Dada a grande dependência de dados que os algoritmos principais do roteador possuem, a aplicação de paralelismo não trouxe ganhos significativos no estágio principal devido a dificuldade na exploração da execução *multi-thread* em grande parte dos processos neste estágio.

A partir dos resultados finais, conclui-se que é possível gerar melhorias significativas nos resultados, mesmo com a realização de modificações que não envolvam grandes mudanças na estrutura do roteador ou em algoritmos implementados. Em resumo, após a análise do roteador e aplicação das melhorias propostas, foi possível gerar melhorias nos resultados de tempo de execução total, tempo de execução no estágio de refinamento, e também no comprimento total de fio em alguns casos.

Acredita-se que este trabalho pode servir como base para trabalhos futuros, servindo como um atalho para futuras análises e realização de melhorias no roteador estudado. A caracterização realizada no roteador serviu principalmente como ponto inicial para entender o fluxo de execução e as partes mais custosas em termos de desempenho. Uma caracterização mais detalhada aplicada com o objetivo de levantar novas melhorias poderia ser um segundo ponto de partida em novos trabalhos. Além disso, como possíveis melhorias envolvendo novas implementações no roteador em trabalhos futuros, um possível ponto inicial seria a refatoração do roteador para se torna-lo de fácil extensão como um *framework*, de forma a facilitar a implementação de novos algoritmos voltados para o estudo de roteamento global.

REFERÊNCIAS

CHANG, Y.-J.; LEE, Y.-T.; WANG, T.-C. NTHU-Route 2.0: a fast and stable global router. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2008., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.338–343.

GAO, J.-R.; WU, P.-C.; WANG, T.-C. A new global router for modern designs. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 2008., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.232–237.

HAN, Y.; ANCAJAS, D. M.; CHAKRABORTY, K.; ROY, S. Exploring high-throughput computing paradigm for global routing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.22, n.1, p.155–167, 2014.

HU, J.; ROY, J. A.; MARKOV, I. L. Completing high-quality global routes. In: PHYSICAL DESIGN, 19., 2010. **Proceedings...** [S.l.: s.n.], 2010. p.35–41.

ISPD2008. **ISPD 2008 Global Routing Contest**. Disponível em: <http://archive.sigda.org/ispd2008/contests/ispd08rc.html/>, Acesso em: 21 de nov. 2017.

ISPD2018. **ISPD 2018 Detailed Routing Contest**. Disponível em: <http://www.ispd.cc/contests/18/index.htm/>, Acesso em: 13 de ago. 2018.

LIU, G.; GUO, W.; LI, R.; NIU, Y.; CHEN, G. XGRouter: high-quality global router in X-architecture with particle swarm optimization. **Frontiers of Computer Science**, [S.l.], v.9, n.4, p.576–594, 2015.

LIU, W.-H.; KAO, W.-C.; LI, Y.-L.; CHAO, K.-Y. NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing. **IEEE Transactions on computer-aided design of integrated circuits and systems**, [S.l.], v.32, n.5, p.709–722, 2013.

MIKAMI, K. A computer program for optimal routing of printed circuit conductors. **IFIP cong.'68, 1968**, [S.l.], 1968.

MOFFITT, M. D.; ROY, J. A.; MARKOV, I. L. The coming of age of (academic) global routing. In: PHYSICAL DESIGN, 2008., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.148–155.

PAN, M.; CHU, C. FastRoute: A step to integrate global routing into placement. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2006., 2006. **Proceedings...** [S.l.: s.n.], 2006. p.464–471.

REY, J. **What does physical verification need today?** « **Design with Calibre.** Disponível em: <https://blogs.mentor.com/calibre/blog/2017/05/31/what-does-physical-verification-need-today/>, Acesso em: 14 de nov. 2017.

SHERWANI, N. A. **Algorithms for VLSI physical design automation.** [S.l.]: Springer Science & Business Media, 2012.

SIDDIQI, U. F.; SAIT, S. M. A game theory based post-processing method to enhance VLSI global routers. **IEEE Access**, [S.l.], v.5, p.1328–1339, 2017.

TUMELERO, D.; BONTORIN, G.; REIS, R. Overhead for independent net approach for Global Routing. In: CIRCUITS & SYSTEMS (LASCAS), 2015 IEEE 6TH LATIN AMERICAN SYMPOSIUM ON, 2015. **Anais...** [S.l.: s.n.], 2015. p.1–4.

WANG, L.-T.; CHANG, Y.-W.; CHENG, K.-T. T. **Electronic design automation: synthesis, verification, and test.** [S.l.]: Morgan Kaufmann, 2009.

XU, Y.; ZHANG, Y.; CHU, C. FastRoute 4.0: global router with efficient via minimization. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 2009., 2009. **Proceedings...** [S.l.: s.n.], 2009. p.576–581.

ANEXO A EXEMPLOS DE ALTERAÇÕES REALIZADAS COM O OPENMP

```
45 void Multisource_multisink_mazeroute::trace_back_to_find_path_2d(MMM_element *end_point) {
46     int shared_loop_counter, shared_stopping_condition, thread_number, copy_shared_loop, thread_stop;
47     omp_set_num_threads(4);
48     shared_loop_counter = -1;
49     shared_stopping_condition = 0;
50     MMM_element *cur_pos = end_point;
51     #pragma omp parallel private(thread_number, copy_shared_loop, thread_stop)
52     {
53         thread_number = omp_get_thread_num();
54         while (!shared_stopping_condition) {
55             #pragma omp critical
56             {
57                 shared_loop_counter++;
58                 copy_shared_loop = shared_loop_counter;
59             }
60
61             SPDLOG_TRACE(log_sp, "cur_pos->coor {}", cur_pos->coor.toString());
62             element->path.push_back(cur_pos->coor);
63             if (cur_pos == cur_pos->parent) {
64                 thread_stop = 1;
65             }
66             cur_pos = cur_pos->parent;
67
68             if (thread_stop) {
69                 shared_stopping_condition = 1;
70                 #pragma omp flush(shared_stopping_condition)
71             }
72         }
73     }
74 }
```

Figura 32: Exemplo de alteração realizada em um *loop while* para ser executado em paralelo utilizando a biblioteca OpenMP.

```
10 void Route_2pinnets::allocate_gridcell() {  
11     #pragma omp parallel num_threads(4)  
12     {  
13         #pragma omp for  
14         for (u_int32_t x = 0; x < gridcell.size(); ++x) {  
15             #pragma omp parallel for  
16             for (u_int32_t y = 0; y < gridcell[0].size(); ++y) {  
17                 gridcell[x][y].set(x, y);  
18             }  
19         }  
20     }  
21     SPDLOG_TRACE(log_sp, "initialize gridcell successfully");  
22 }  
23
```

Figura 33: Exemplo de alteração realizada em um *loop for* para ser executado em paralelo utilizando a biblioteca OpenMP.

ANEXO B EXEMPLO DE ARQUIVOS LEF/DEF

```
VERSION 5.7 ;
NAMECASESENSITIVE ON ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

UNITS
    DATABASE MICRONS 2000 ;
END UNITS

MANUFACTURINGGRID 0.0025 ;

SITE core
    SIZE 0.19 BY 1.71 ;
    CLASS CORE ;
    SYMMETRY Y ;
END core

LAYER metal1
    TYPE ROUTING ;
    DIRECTION HORIZONTAL ;
    PITCH 0.200 ;
    OFFSET 0.000 ;
    WIDTH 0.100 ;
END metal1

END LIBRARY
```

Figura 34: Exemplo de um arquivo LEF com a descrição de camadas (*layers*), e espaços de posicionamento (*sites*) (ICCAD2015, 2019).

```

VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN simple ;
UNITS DISTANCE MICRONS 2000 ;

DIEAREA ( 0 0 ) ( 13680 13680 ) ;

ROW core_SITE_ROW_2 core 0 6840 N D0 36 BY 1 STEP 380 0 ;
COMPONENTS 2 ;
  - u1 NAND2_X1
    + PLACED ( 3420 6840 ) N ;
  - u2 NOR2_X1
    + PLACED ( 3420 3420 ) N ;
END COMPONENTS
PINS 2 ;
  - out + NET out
    + DIRECTION OUTPUT
    + FIXED ( 13680 6000 ) N
    + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
  - inp1 + NET inp1
    + DIRECTION INPUT
    + FIXED ( 3420 13680 ) N
    + LAYER metal3 ( 0 0 ) ( 100 100 ) ;
END PINS
END DESIGN

```

Figura 35: Exemplo de um arquivo DEF com a descrição de linhas (*rows*), componentes (*components*), e pinos (*pins*) (ICCAD2015, 2019).

ANEXO C EXEMPLO DE UM ARQUIVO DE ENTRADA PARA ROTEAMENTO GLOBAL

```
grid # # # (x grids, y grids, number of layers)
vertical capacity # # # # # (vertical capacity by default on each layer)
horizontal capacity # # # # #
minimum width # # # # #
minimum spacing # # # # #
via spacing # # # # #
lower_left_x lower_left_y tile_width tile_height

num net #
netname id_# number_of_pins minimum_width
x y layer
x y layer
...
[repeat for the appropriate number of nets]
```

Figura 36: Estrutura da entrada de arquivo para roteamento global na extensão utilizada no ISPD 2007 e 2008. No arquivo são descritas a grade de roteamento, as redes a serem roteadas, e outras constantes envolvidas. (ISPD2008, 2017).