UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DIEGO TUMELERO

Exploração de Paralelismo no Roteamento Global de Circuitos VLSI

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ricardo Augusto da Luz Reis

Porto Alegre
2015

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Tumelero, Diego
Exploração de Paralelismo no Roteamento Global de Circuitos VLSI [manuscrito] / Diego Tumelero. – 2015.
72 f.:il.
Orientador: Prof. Dr. Ricardo Augusto da Luz Reis.
Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2015.
1.Roteamento Global. 2.Paralelismo 3.CAD. 4.VLSI. I. Reis, Prof. Dr. Ricardo Augusto da Luz. II. Título.

"Trust I seek and I find in you
Every day for us something new
Open mind for a different view
And nothing else matters"
– JAMES HETFIELD E LARS ULRICH

AGRADECIMENTOS

Aos meus pais, Roberto e Marilene que sempre me apoiaram e em especial a minha companheira Flávia por estar ao meu lado me apoiando mesmo nos momentos difíceis.

RESUMO

Com o crescente aumento das funcionalidades dos circuitos integrados, existe um aumento consequente da complexidade do projeto dos mesmos. O fluxo de projeto de circuitos integrados inclui em um de seus passos o roteamento, que consiste em criar fios que interconectam as células do circuito. Devido à complexidade, o roteamento é dividido em global e detalhado. O roteamento global de circuitos VLSI é uma das tarefas mais complexas do fluxo de síntese física, sendo classificado como um problema NP-completo. Neste trabalho, além de realizar um levantamento de trabalhos que utilizam as principais técnicas de paralelismo com o objetivo de acelerar o processamento do roteamento global, foram realizadas análises nos arquivos de *benchmark* do ISPD 2007/08. Com base nestas análises foi proposto um método que agrupa as redes para então verificar a existência de dependência de dados em cada grupo. Esta verificação de dependência de dados, que chamamos neste trabalho de colisor, tem por objetivo, criar fluxos de redes independentes umas das outras para o processamento em paralelo, ou seja, ajudar a implementação do roteamento independente de redes. Os resultados demonstram que esta separação em grupos, aliada com a comparação concorrente dos grupos, podem reduzir em 67x o tempo de execução do colisor de redes se comparada com a versão sequencial e sem a utilização de grupos. Também foi obtido um ganho de 10x ao comparar a versão com agrupamentos sequencial com a versão paralela.

Palavras-chave: Roteamento Global. Paralelismo. EDA. VLSI. Microeletrônica.

Parallel Computing Exploitation Applied for VLSI Global Routing

ABSTRACT

With the increasing of the functionality of integrated circuits, there is a consequent increase in the complexity of the design. The IC design flow includes the routing in one of its steps, which is to create wires that interconnect the circuit cells. Because of the complexity, routing is divided into global and detailed. The global routing of VLSI circuits is one of the most complex tasks in the flow of physical synthesis and it's classified as an NP-complete problem. In this work, a parallel computing techniques survey was applied to the VLSI global routing in order to accelerate the global routing processing analyzes. This analyzes was performed on the ISPD 2007/08 benchmark files. We proposed a method that groups the networks and then check for data dependence in each group based on these analyzes. This data dependency checking, we call this checking of collider, aims to create flow nets independent of each other for processing in parallel, or help implement the independent routing networks. The results demonstrate that this separation into groups, together with the competitor comparison of groups, can reduce 67x in the collider networks runtime compared with the sequential release and without the use of groups. It was also obtained a gain of 10x when comparing the version with sequential clusters with the parallel version.

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

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BFS	*Breadth-first Search*
BLMR	*Bounded-length Maze Routing*
CPU	*Central Processing Unit*
CSP	*Constrained Shorstest Path*
CTS	*Clock Tree Synthesis*
DFS	*Depth-first Search*
EDA	*Eletronic Design Automation*
FPGA	*Field-programmable Gate Array*
GCC	*GNU Compiler Collection*
GPU	*Graphics Processing Unit*
GR	*Global Router*
ILP	*Integer Linear Programming*
IP	*Integer Programming*
ISPD	*ACM International Symposium on Phisical Design*
MIMD	*Multiple Instruction, Multiple Data stream*
MISD	*Multiple Instruction, Single Data stream*
SIMD	*Single Instruction, Multiple Data stream*
SISD	*Single Instruction, Single Data stream*
MPI	*Message Passing Interface*
NLC	*Net Level Concurrency*
NVCC	*NVidia Cuda Compiler*
UFRGS	Universidade Federal do Rio Grande do Sul
VLSI	*Very Large Scale Integration*

SUMÁRIO

1 INTRODUÇÃO

O processo de projeto de circuitos integrados inclui em um de seus passos a síntese física de circuitos integrados que tem em uma de suas etapas o roteamento, que consiste em criar fios que interconectam as portas lógicas do circuito.

A etapa de roteamento é dividida em roteamento global e roteamento detalhado: O roteamento global cria rotas para as redes de forma preliminar, sem considerar todas as nuances das regras de projeto, ou seja, trabalha de forma simplificada o processo de roteamento. Desta forma, o roteador detalhado tem a tarefa de aplicar as rotas definitivas a cada fio, seguindo os detalhes de fabricação da tecnologia utilizada.

Apesar de ser uma das primeiras áreas de *Eletronic Automation Design* que foi automatizada, o roteamento de circuitos VLSI (*Very Large Scale Integration*) permanece como uma área com significativa atividade de pesquisa e desenvolvimento (ISPD, 2008). Em parte, isto é devido que a etapa de roteamento é considerada um problema NP-completo (SZYMANSKI, 1985). Este passo é o que mais demanda tempo de processamento e, por muitas vezes, não é explorada a computação paralela que pode ser encontrada nos métodos utilizados para a execução do roteamento global de circuitos integrados.

1.1 Objetivos

O objetivo deste trabalho simular e medir a possibilidade de paralelizar de forma eficiente a etapa de Roteamento Global para futuras implementações que utilizem o potencial de arquiteturas massivamente paralelas, como as unidades de processamento gráfico (GPUs).

Para isto, foi criada uma metodologia que é dividida em quatro etapas: Estudar um roteador global sequencial, analisar estatisticamente os arquivos de *benchmark*, escrever um colisor de redes, que detecta as dependências de dados e analisar o *overhead* deste processo para medir a performance do conceito proposto.

1.2 Organização do Documento

No capítulo 2 é realizada uma revisão acerca dos conceitos gerais, incluindo roteamento global, paralelismo, complexidade computacional e pesquisa em grafos. No capítulo 3 é abordado o estado da arte de roteadores globais que utilizam paralelismo. A

metodologia é abordada no capítulo 4. No capítulo 5 são apresentados os resultados. Por fim, no capítulo 6 são apresentadas as conclusões.

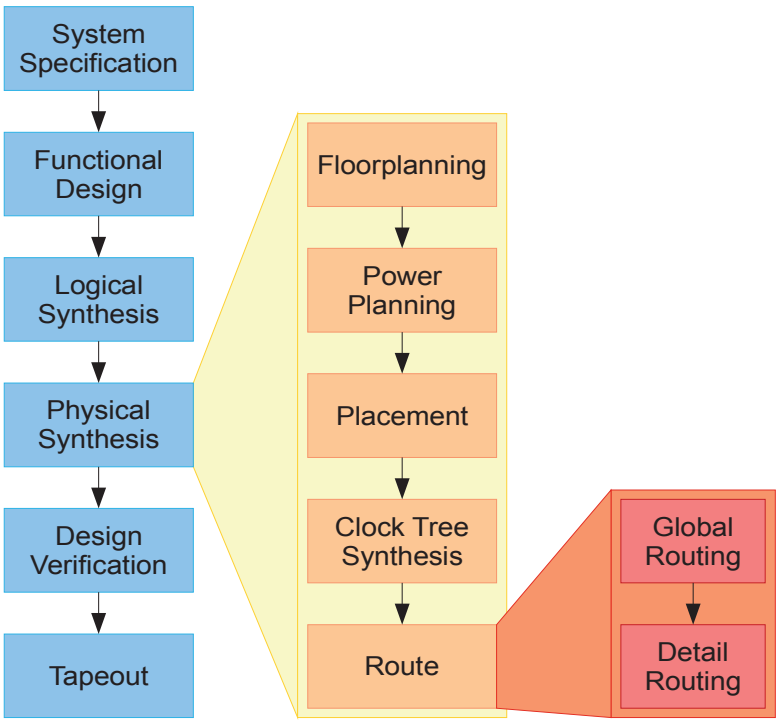
2 CONTEXTO DA PESQUISA

Neste capítulo são abordados conceitos necessários para o melhor entendimento do trabalho. Roteamento global, paralelismo e estudo de grafos são os principais temas explanados nesta seção.

2.1 Roteamento Global

O roteamento global faz parte da síntese física de circuitos integrados e é realizado antes do roteamento detalhado, que define por quais camadas de metal os fios passarão e onde exatamente cada fio passará no chip. E, tipicamente, após a síntese da árvore de relógio (ou CTS) na síntese física. O processo de roteamento global tem como entrada uma lista de conexões a serem realizadas com locais fixos de blocos e pinos expressa pela etapa de posicionamento. A Figura 2.1 mostra um fluxo de projeto de circuitos VLSI.

Figura 2.1 – Fluxo de projeto de circuitos VLSI.



Baseada em (Sherwani 1993).

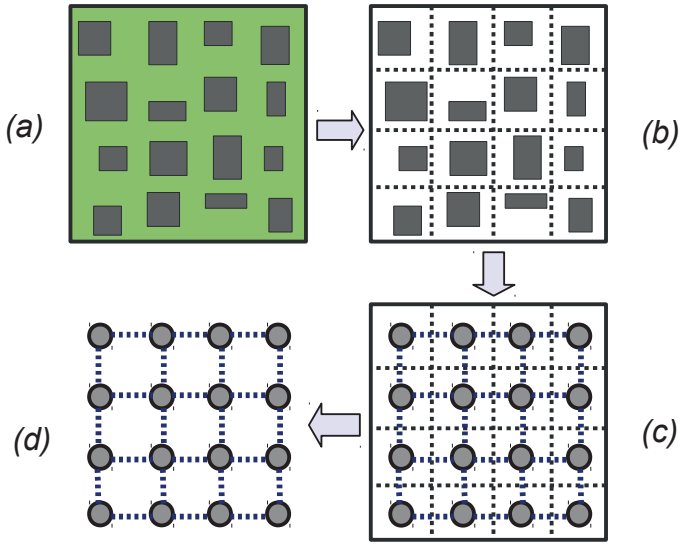
Após o posicionamento, o circuito é particionado em regiões denominadas *tiles* e decide o caminho entre estas partições para todas as conexões. O objetivo do roteamento

global é determinar regiões para onde as conexões especificadas na *netlist* serão realizadas respeitando as restrições impostas. Estas conexões interligam células padrão que contém elementos básicos da lógica. O roteamento global não considera de forma total as regras de projeto de cada tecnologia. E, após finalizado, o resultado pode ainda conter algum congestionamento. Entretanto, o roteamento global tem por um dos seus objetivos eliminar o congestionamento de forma a tornar mais fácil o roteamento detalhado. Depois desta tarefa o roteador detalhado definirá por quais camadas de metal os fios passarão e por onde exatamente cada fio passará no chip (JOHANN, 2001) (REIMANN, 2013).

A caracterização matemática do roteamento global é baseada na teoria dos grafos, onde o circuito é representado por uma grade G que especifica dois grupos de componentes, um conjunto de vértices V e de arestas (*egdes*) E. Cada $v_i \in V$ indica uma região (célula ou *tile*). Cada $e_{ij} \in E$ corresponde a fronteira entre regiões adjacentes (MOFFITT; ROY; MARKOV, 2008). Estas bordas têm um limite de recursos disponíveis m_{ij} que pode ser utilizado para quantificar o congestionamento das conexões. Existe, ao menos, um conjunto de redes N, *sendo* cada $n_i \in N$ é composto de um conjunto P_i de pinos. Cada um destes pinos corresponde a um vértice v_i.

A solução é encontrada quando todas as conexões são roteadas, respeitando os limites de cada aresta com o menor tempo de processamento possível e satisfazendo restrições como o comprimento de fio, número de vias, fabricabilidade, temperatura, entre outros. A Figura 2.2 mostra uma visão geral do roteamento global.

Figura 2.2 – Metodologia do Roteamento Global. (a) resultado do posicionamento. (b) particionamento do circuito. (c) representação das bordas. (d) grafo do roteamento global.



A próxima seção apresenta uma revisão envolvendo roteadores globais acadêmicos presentes na literatura.

2.1.1 Principais Roteadores Globais Sequenciais Presentes na Acadêmia

Notáveis avanços foram, recentemente, conquistados nos algoritmos de roteamento e ferramentas EDA. Alguns roteadores ganharam notoriedade na esfera acadêmica devido à performance demonstrada em competições recentes do ISPD 2007 e 2008 (NAM, 2007) (NAM; SZE; YILDIZ, 2008).

O *International Symposium on Physical Design* (ISPD) é uma competição de desenvolvimento de ferramentas EDA, onde nas edições de 2007 e 2008 o assunto abordado foi o roteamento global. Para a realização destas duas edições da competição, foram criados arquivos de benchmark baseados em circuitos fabricados pela indústria, ou seja, arquivos produzidos por empresas do ramo especialmente para a competição, com o intuito de serem o mais próximo possível da realidade da indústria de semicondutores. Dentre os roteadores competidores, pode-se citar:

- **FastRoute** (PAN; CHU, 2006), **FastRoute 2.0** (PAN; CHU, 2007) e **FastRoute 4.0** (XU; ZHANG; CHU, 2009). FastRoute utiliza um mapa de congestionamentos para modelar uma grade de Hanan (HANAM, 1966) durante a geração de árvore Steiner seguido pelo *edge shifting* e *pattern routing*. Em (PAN; CHU, 2007), o roteador é atualizado com o método de roteamento monotônico e por um *maze routing*, *multi-source* de roteamento. A versão mais recente aborda o problema de otimização do número de vias ao longo de todo o fluxo de roteamento global.
- **BoxRouter** (PAN, 2006). A ideia principal do BoxRouter é expandir progressivamente um perímetro a partir da região mais congestionada do chip, aplicando programação linear inteira (ILP) na formulação e considerando os padrões em forma de L para refazer conexões entre perímetros sucessivos.
- **BoxRouter 2.0** (PAN, 2007) é uma evolução que utiliza uma versão dinâmica do método de pesquisa A* (A estrela) e incorpora, adaptável à topologia *rip-up* para passar os fios das regiões congestionadas, sem alterar a topologia da rede.
- **Fairly Good Router (FGR)** (ROY; MARKOV, 2008) baseia-se no roteador *PathFinder* (MCMURCHIE; EBELING, 1995), originalmente desenvolvido para

FPGAs. Ele usa uma função específica para penalizar o congestionamento e realiza uma rápida atribuição de camadas seguido por um *3D clean-up*.

- **MaizeRouter** (MOFFITT, 2008). Este roteador utiliza duas operações elementares baseadas em borda (*extreme edge shifting* e *edge retraction*) e manipula sequencialmente os segmentos individuais das redes. Sua abordagem é baseada na decomposição interdependente das redes, em que as soluções de roteamento são implicitamente mantidas por conjuntos de intervalos em vez de topologias definidas.
- **NTHU-Route 2.0** (GAO; WANG, 2008) é baseado em *rip-up* e *re-route*. Ele usa uma função de custo baseado em um histórico para distribuir o *overflow* e emprega um método de identificação para especificar a ordem de processamento para regiões *rip-up* congestionadas. A redução do comprimento de fio é conquistada através de um método *maze routing* adaptativo, *multi-sink*, *multi-source* de roteamento.

A Tabela 2.1 apresenta uma relação entre os roteadores apresentados e suas características. Uma das mais perceptíveis é a utilização quase que da totalidade dos roteadores da ferramenta FLUTE para a realização da decomposição de redes com múltiplos pinos em redes de 2 pinos. Além disto, todos os GRs apresentados fazem uso de algum algoritmo de roteamento de labirinto, sendo que apenas 3 utilizam A*.

Tabela 2.1 – Principais roteadores acadêmicos.

	Pattern routing	Monotonic routing	Maze routing	A* search	FLUTE dependence	Topology reconstruction	Incremental	Edge "sliding"	Resource sharing	ILP or MCF	Congestion manipulation	History-based	Layer Assignment	Open Source
FastRoute	■		■		■	■			■		■			
FastRoute 2.0	■	■	■		■	■		■	■		■			
FastRoute 4.0	■	■	■		■	■		■	■		■		■	
BoxRouter	■		■		■					■				
BoxRouter 2.0	■		■	■	■	■				■	■			
FGR			■	■		■	■		■		■	■	■	■
MaizeRouter	■		■	■	■	■	■	■	■		■	■	■	■
NTHU-Route 2.0	■	■	■		■	■					■	■	■	

Na próxima seção será abordada uma revisão sobre computação paralela, onde será visto também a taxonomia de Flynn e hardwares massivamente paralelos que podem ser utilizados para processar o roteamento global.

2.2 Paralelismo

Computação paralela é uma forma de computação em que a execução de cálculos ocorre de forma simultânea (ALMASI, 1989).

Uma das características sobre o paralelismo está presente na confusão sobre o entendimento entre escalabilidade e alta performance, pois ambos tratam sobre coisas distintas. Um exemplo é que um algoritmo pode ser escalável, porém com uma complexidade de $O(n^2)$ em vez de utilizar uma abordagem otimizada que poderia ser de complexidade $O(n)$ (MADDEN, 2012).

Isto quer dizer que os algoritmos existentes na computação foram, em grande parte, concebidos em um tempo onde com computadores tinham um funcionamento sequencial e não havia um conhecimento de que estes métodos um dia precisariam ser convertidos em abordagens paralelas. Isto é um problema no sentido de que estes métodos foram otimizados ao longo de várias décadas para serem executados apenas em máquinas sequenciais, e frente a isto métodos paralelos ainda não tem todo este tempo de maturidade. Isto tem impacto pratico onde existe uma tendência de que abordagens sequenciais tenham um nível de otimização muito grande para implementações sequenciais, mas que ao mesmo tempo, são algoritmos que não tem grande potencial de escalarem em arquiteturas paralelas.

Por outro lado, os métodos que são mais facilmente escaláveis tendem a serem os algoritmos que não são otimizados e, de certa forma, podem ser classificados como algoritmos de força bruta em grande maioria. Isto pode gerar implementações aparentemente contraditórias, onde métodos que tem desempenho de tempo de execução muito ruins em arquiteturas sequenciais podem ser aqueles que mais tem o maior potencial de escalar em sistemas de computação paralela.

Outra característica trata de que a parcela que é paralelizável é a mais importante, o que é errado segundo a lei de Amdahl (AMDAHL, 1967) em que o ganho de performance é limitado à porção serial do algoritmo. Ou seja, a porção serial de um método é o que acaba por limitar em última análise o seu desempenho, não a porção paralela.

Outra característica a se considerar durante o desenvolvimento de software que utilize recursos de computação paralela é existem algoritmos que simplesmente são sequenciais e não conseguem escalar em sistemas deste tipo, independente do tempo de programação despendido em uma tentativa de conversão. Estes métodos foram desenvolvidos segundo uma filosofia sequencial. Enquanto isto, existem fortes indícios de que diversos algoritmos voltados para a computação paralela ainda não foram desenvolvidos.

É de conhecimento (BORKAR, 2007) que o através da Regra de Pollack existe uma diferença entre a taxa de crescimento da capacidade de processamento e a complexidade (ou área) de um circuito. Desta forma, se a capacidade de processamento de um processador cresce em um ritmo linear, a complexidade aumenta quadraticamente. Isto pode ser explicado melhor pela fórmula $Performance \sim \sqrt{Área}$. Esta formulação de Fred Pollack é baseada empiricamente ao longo de seu trabalho na intel é um dos mais fortes argumentos para a existência dos processadores de múltiplos núcleos de processamento.

Este argumento também funciona quando relacionamos a capacidade de processamento de placas gráficas (GPUs) a capacidade de processamento de CPUs. Onde as GPU conseguem processar montantes de dados na ordem de alguns *teraflops* com suas centenas de pequenos processadores, as CPUs conseguem entregar poucas centenas de *gigaflops.*

Associado a isto, a Lei de Moore nos aponta o caminho do paralelismo para solucionar os problemas complexos da computação.

Dentre os casos de estudo na área de EDA tem-se duas abordagens: A portabilidade e a re-arquitetura de algorítmos. No primeiro caso são apresentados exemplos como aceleração de SPICE, simulação de falhas e análise estatística de tempo estático baseado em Monte Carlo (CROIX; KHATRI, 2009). No segundo caso, o problema de satisfazibilidade booleana, geração de tabela de falhas e *power grid analysis* (CROIX; KHATRI, 2009).

2.2.1 Taxonomia de Flynn

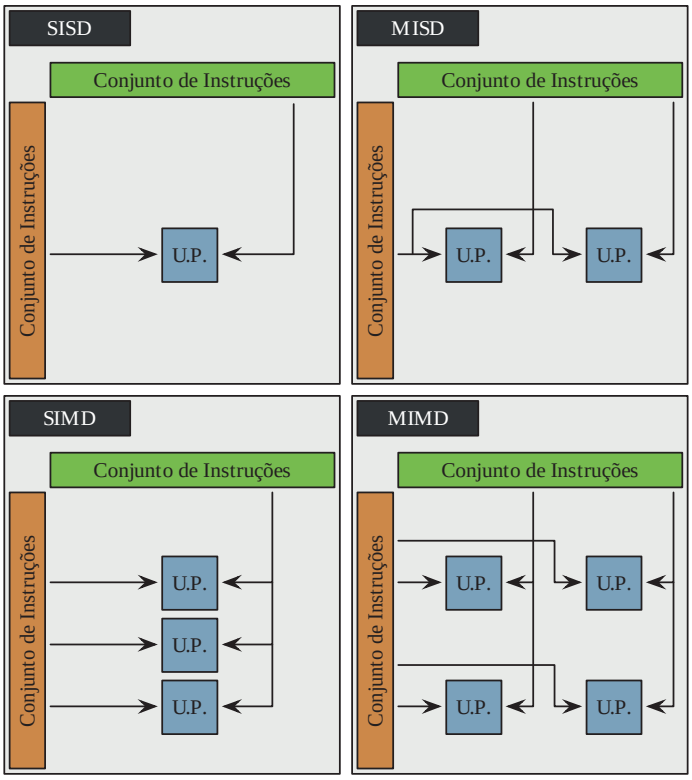
A taxonomia de Flynn (Flynn, 1972) pode ser dividida em 4 categorias conforme a Figura 2.3.

SISD (*Single Instruction, Single Data stream*) refere-se ao processamento sequencial, onde uma instrução é aplicada a um conjunto de dados por vez. Em MISD (*Multiple Instruction, Multiple Data stream*) diversos conjuntos de instruções são executadas

simultaneamente em um conjunto de dados, uma abordagem utilizada em alguns modelos de tolerância a falhas.

Os modelos SIMD (*Single Instruction, Multiple Data stream*) e MIMD (Multiple *Instruction, Multiple Data stream*) são os que ~~nos~~ interessam a esta pesquisa pois são as formas de computação paralelas existentes. Assim sendo, respectivamente, a execução de uma instrução em diversos conjuntos de dados simultaneamente, por exemplo um vetor, além da execução paralela de diversas instruções em diversos conjuntos de dados ao mesmo tempo. Desta forma caracterizando a computação paralela em 2 grupos.

Figura 2.3 – Taxonomia de Flynn.



Estes modelos inspiraram várias implementações que utilizam paralelismo e arquiteturas massivamente paralelas, tema da próxima subseção.

2.2.2 Hardware Massivamente Paralelo

Nesta seção é apresentado um breve panorama do hardware massivamente paralelo, em uma abordagem focada em dispositivos únicos, como GPUs.

Os avanços tecnológicos na área das GPUs trazem para o seu domínio novos problemas, diferentes dos tradicionais, necessitando toda uma atenção a comunicação entre as unidades de processamento e os problemas decorrentes da execução concorrente. Desta forma, com centenas de unidades de processamento, as GPUs tem potencial, se bem utilizadas, de ter maior capacidade de processamento que os processadores de propósito geral atuais. Existem tipicamente 2 tipos de GPUs, SIMD e MIMD (AKL, 1997). AMD (AMD) e NVIDIA (NVIDIA) seguem o primeiro tipo e o coprocessador intel Xeon Phi (INTEL) o segundo.

As duas tecnologias de programação principais para GPUs são o CUDA (NVIDIA) (HWU, 2012) e o OpenCL (KHRONOS GROUP) (GASTER, 2012). A primeira é propriedade da NVIDIA e consiste em uma extensão para C/C++, sendo que a parcela de GPU é compilada pelo NVCC (compilador do CUDA) e depois vinculada com a parcela C/C++ compilada pelo GCC, gerando um arquivo executável híbrido, em que é primariamente executado pela CPU e posteriormente envia para a GPU as instruções CUDA.

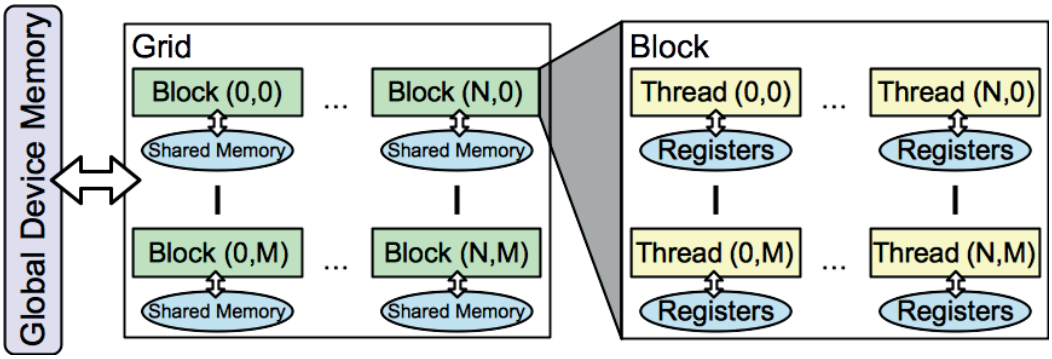
A tecnologia OpenCL que é desenvolvida pelo Khronos Group aceita uma gama de dispositivos maiores, inclusive FPGAs. E, também, é uma extensão do C/C++, com a diferenciação de gerar um *bytecode* que é compilado em tempo de execução. A Apple é participante ativa do grupo de desenvolvimento do OpenCL e a tecnologia é suportada nativamente pelo MacOS X, enquanto que em outras plataformas é necessário que os controladores sejam instalados (GASTER, 2012). A natureza do projeto OpenCL permite que a sua implementação ocorra em dispositivos de diversas empresas, entre elas Altera, AMD, Apple, ARM Holdings, Creative, IBM, Intel, Nokia, Samsung dentre outras empresas.

O coprocessador Xeon Phi diferentemente das GPUs é programado como uma máquina com uma CPU de múltiplos núcleos de processamento, permitindo utilizar tecnologias como OpenMP e MPI de forma transparente (INTEL). Um exemplo de utilização deste equipamento é o supercomputador chinês Tianhe2, com um desempenho de aproximadamente 34 petaflops (TOP 500, 2014).

Outro fator a ser considerado é de que a CPU é projetada para ter acesso randômico à memória, utilizando para tais diversos níveis de cache. Já a GPU mantém um sistema de *pipeline* com uma quantidade de cache de tamanho menor. Outra situação é a ramificação de instruções condicionais, de forma a evitar o esvaziamento do *pipeline*, pois nesta arquitetura, nas instruções condicionais ambas as opções da lógica do desvio são executadas devido ao arrasto do pipeline e quando a condição do desvio é resolvida, uma das linhas de execução é morta. Na Figura 2.4 é apresentada de maneira geral a hierarquia de memória de uma GPU, a

qual existe uma divisão em níveis, formando uma hierarquia de memória semelhante ao que ocorre com a cache das CPUs, com a diferença de haver centenas de unidades de processamento a mais.

Figura 2.4 – Hierarquia de memória em uma GPU.



Existe uma liberdade para a modelagem do problema a ser executado em CUDA ou OpenCL. Pois na visão do programador existe um número ilimitado de processadores de forma que o problema pode ser alocado em diversas dimensões (CROIX; KHATRI, 2009).

Ao executar uma aplicação na GPU, organizam-se os comandos recebidos em uma fila e os despacha de forma transparente, ou seja, não é possível alterar a forma em que os comandos são executados, exceto pela adição de barreiras de sincronismo.

O grande desempenho potencial de arquiteturas de hardware massivamente paralelas só pode ser aproveitado se os algoritmos forem especificamente projetados. Por isso, na próxima seção será abordado o estudo de grafos, seus métodos de pesquisa e complexidade computacional.

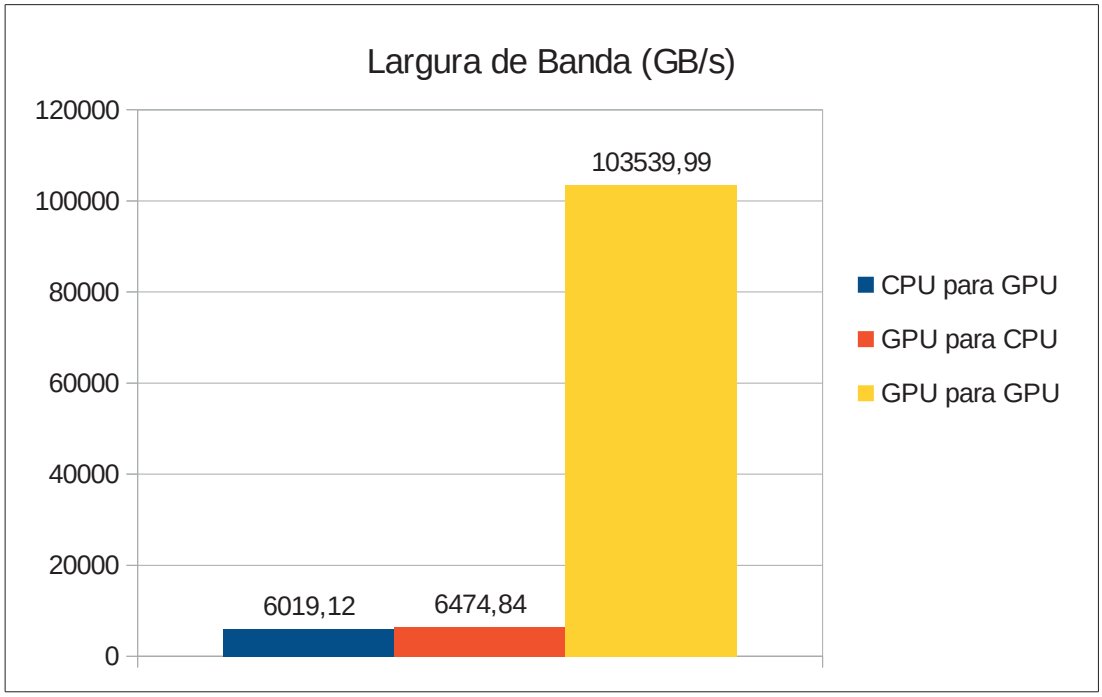
Uma grande limitação deste modelo é o gargalo de comunicação que existe devido a separação das memórias da GPU e da CPU. Este gargalo pode facilmente anular o desempenho extra da GPU (HAN, 2014).

Para demonstrar esta situação, montou-se um experimento ao qual foram realizados testes de transferência de dados de forma a mostrar de forma numérica este fator limitante. Este gargalo ocorre no barramento PCI Express, no qual a velocidade do barramento na versão 3.0 é de aproximadamente 16 GB/s, em contraste com a velocidade de comunicação interna da GPU que no modelo NVIDIA GeForce GTX 650 Ti é da ordem de aproximadamente 100 GB/s como apresentado na Figura 2.5.

Os resultados da Figura 2.5 foram obtidos em um computador com processador Intel Core i7 3930k (6 núcleos) e uma GPU NVIDIA GeForce GTX650Ti Boost com 2GB de

memória GDDR5 em uma placa-mãe com barramento PCI Express x16 3.0 (aproximadamente 16 GB/s teóricos) com 64 GB de memória RAM DDR3. O software de teste é um exemplo do pacote de instalação do ambiente de desenvolvimento CUDA.

Figura 2.5 – Relação entre largura de banda interna da GPU e da comunicação entre a GPU e a memória RAM. É apresentada a diferença entre três formas de comunicação, de forma a evidenciar o gargalo de comunicação entre a CPU e a GPU.



2.3 Complexidade Computacional e Pesquisa em Grafos

Complexidade computacional é um ramo da ciência da computação que estuda como os algoritmos e métodos utilizam os recursos computacionais quando são escalados, ou seja, como um algoritmo se comporta a quantidade de dados de entrada (alteração do tamanho da entrada). O estudo desta alteração do tamanho das entradas de um método tem impactos no tempo de execução e consumo de memória principalmente. Predizer o impacto no tempo de execução é uma das atribuições do estudo de complexidade computacional e isto é realizado por meio de funções matemáticas baseadas no tamanho da entrada do algoritmo (GEREZ, 1999).

Existem dois tipos de complexidade computacional: Complexidade temporal e espacial. A primeira quantifica o tempo necessário para completar a computação. A segunda quantifica o montante de memória necessário para a execução do algoritmo.

Dependendo da magnitude do tamanho da entrada, diversos critérios podem ser usados para qualificar um algoritmo: Ordem polinomial vs. ordem exponencial, linear vs. quadrático e sublinear.

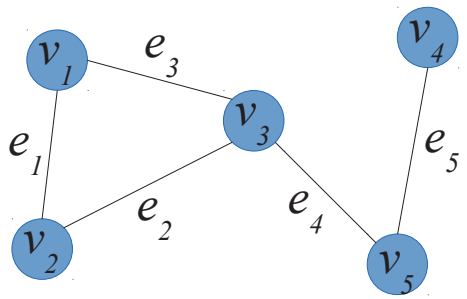
Como exemplo prático, executar o roteamento global de um circuito com X redes leva um tempo Y para ser processado pelo roteador. Enquanto que para um roteador processar um circuito com X*2 redes, o tempo de processamento é muito maior do que Y*2. A dimensão deste tempo pode ser estimada com base nos métodos de pesquisa de grafos utilizados para encontrar o caminho para as redes.

2.3.1 Pesquisa em Grafos

Grafo é uma estrutura matemática que descreve um conjunto de objetos e as conexões entre eles (GEREZ, 1999).

Os grafos são caracterizados por dois conjuntos: o conjunto dos vértices V e o conjunto das arestas E que denotam um grafo $G(V,E)$. Um circuito pode ser representado por G onde cada V indica uma região (célula ou *tile*) e cada $e_{ij} \in E$ corresponde a fronteira entre regiões adjacentes (Moffitt 2008a). Estas bordas têm um limite de recursos disponíveis que pode ser utilizado para quantizar o congestionamento das conexões. Existe ao menos um conjunto de $nets\,N$, e cada $n_i \in N$ é composto de um conjunto P_i de pinos. Cada um destes pinos corresponde a um vértice v_i.

Figura 2.6 – Exemplo de um grafo.



Os dois vértices que formam uma aresta podem ser chamados de *endpoints*. A aresta pode ser identificada por 2 *endpoints* $u, v \in V$ seguindo a notação (u, v) como no exemplo do grafo da Figura 2.6, por exemplo, $e3 = (v1, v3)$.

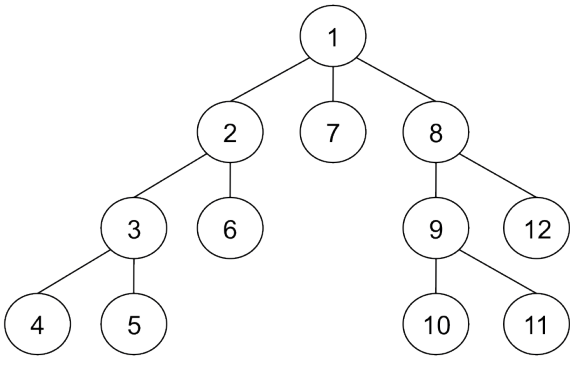
O uso de grafos pode facilitar a formulação matemática, a análise e solução do roteamento global. Estas pesquisas podem ser avaliadas segundo a complexidade computacional do algoritmo de pesquisa de grafos. Logo, eles podem ser usados para simplificar o roteamento transformando-o em uma pesquisa de redes.

2.3.2 Métodos de Pesquisa

Os métodos de pesquisa em grafos mostram a diversidade de formas de se definir a ligação entre 2 pontos em um grafo. Demonstram também abordagens "força bruta" e abordagens "heurísticas", onde a principal diferença entre elas é a complexidade computacional reduzida nas abordagens heurísticas. A seguir estão listados os principais métodos de busca em grafos:

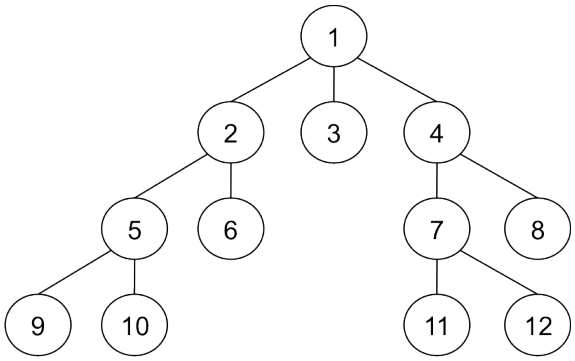
DFS (*Depth-first Search*): A busca em profundidade é um algoritmo usado para realizar uma busca em um grafo, ele explora, a partir da raiz, todos os nodos de um ramo até as suas folhas, para então retroceder até a raiz, Figura 2.7, ou seja, o objetivo é visitar todos os vértices apenas uma vez, aonde por sua vez os mesmos são marcados como visitados. A complexidade temporal deste método é de $O(|E|)$ (sendo E o número de arestas), ou seja, o mesmo pode percorrer todo o grafo na pior hipótese. Já a complexidade espacial é de $O(|V|)$ (sendo V o número de vértices) o que na prática significa que a profundidade da busca é limitada para não esgotar os recursos computacionais (GEREZ, 1999).

Figura 2.7 – Ordem em que os nodos são percorridos no método DFS.



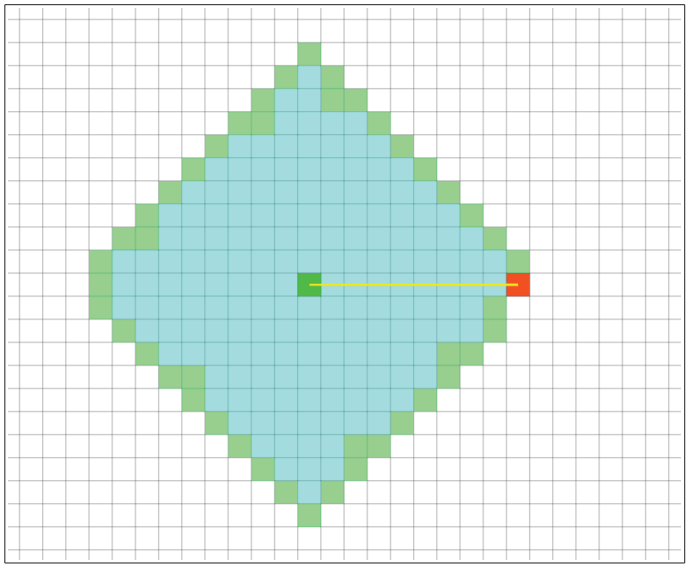
BFS (*Breadth-first Search*): Diferentemente da busca em profundidade, a busca em largura realiza uma pesquisa sistemática (Figura 2.8) no grafo, aonde a complexidade temporal é $O(n+|E|)$, isto ocorre devido a pesquisa abordar os ramos do grafo de forma igualitária, visitando todos os nodos vizinhos do mesmo nível antes de seguir para os nodos filhos (GEREZ, 1999). Já a complexidade espacial é $O(|V|)$ onde $|V|$ é a cardinalidade do conjunto de vértices.

Figura 2.8 – Ordem em que os nodos são percorridos no método BFS.



Dijkstra: Este método pode resolver o problema do caminho mais curto para grafos de arestas de peso não negativo. Para grafos de peso negativo o método mais utilizado é o Bellman-Ford (GEREZ, 1999). O algoritmo é semelhante ao método BFS, porém o mesmo é do tipo guloso, ou seja, utiliza a decisão que parece ótima no momento. Ele verifica todas as arestas conectadas ao nodo inicial (S) e escolhe a de menor custo para ser o novo nodo inicial (S) e continua recursivamente até que o destino seja encontrado (mas ignora as arestas que ligam vértices já pertencentes a S), Figura 2.9. A complexidade deste método é $O(n^2+|E|)$, que pode ser simplificada para $O(n^2)$ (GEREZ, 1999).

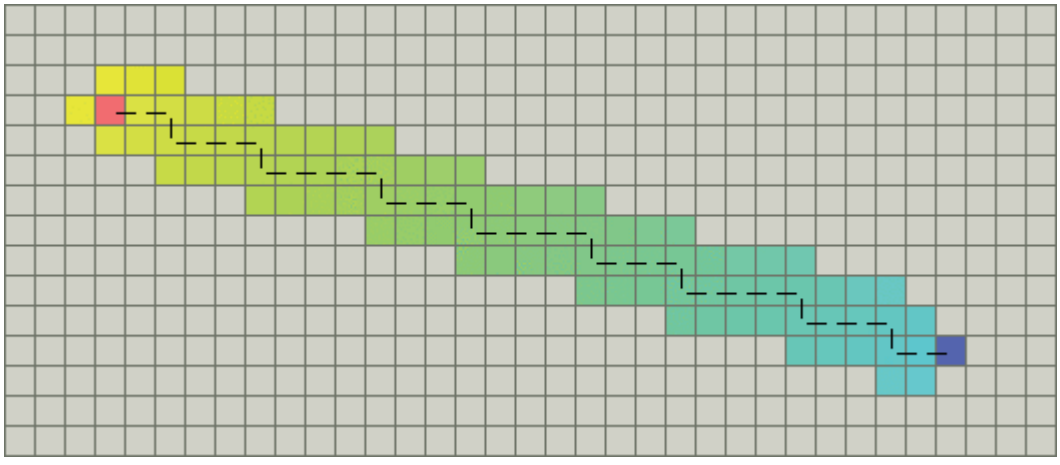
Figura 2.9 – Exemplo de uma busca utilizando o algoritmo de Dijkstra.



Baseada em (XU, 2013).

A* (A Estrela): O algoritmo A* é uma evolução do método de Dijkstra e do *Greedy Best-First-Search*, onde as melhores características dos algoritmos de Dijkstra e *Greedy BFS* são utilizadas, pois encontra o menor caminho se o mesmo existir e pode utilizar heurística como no *greedy*. O método é baseado na seguinte expressão $f(n)=g(n)+h(n)$, sendo $g(n)$ é o custo exato do ponto de partida até um nodo *n* qualquer e $h(n)$ o custo estimado de um nodo *n* qualquer para o destino. Esta estimativa pode ser uma heurística, caminho *manhattan* ou outros métodos que estimem a distância $h(n)$, Figura 2.10 (PATEL, 2013).

Figura 2.10 – Exemplo de busca utilizando o método A*.



Baseada em (PATEL, 2013).

O pseudocódigo do algoritmo A* pode ser visto na Figura 2.11.

Figura 2.11 – Pseudocódigo do algoritmo A*.

```
function A*(start,goal)
    closedset := the empty set
    openset := {start}
    came_from := the empty map

    g_score[start] := 0
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            tentative_g_score := g_score[current] + dist_between(current,neighbor)
            if neighbor in closedset and tentative_g_score >= g_score[neighbor]
                continue

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] +
heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openset
                    add neighbor to openset

    return failure

function reconstruct_path(came_from, current_node)
    if current_node in came_from
        p := reconstruct_path(came_from, came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

LCS* (*Lower bound Cooperative Search*): O algoritmo LCS* (JOHANN, 2000) é baseado em uma estrutura de busca bidirecional e pode ser dividido em duas características, cooperação e visibilidade:

A cooperação é conquistada ao se observar se a função *g()* de uma busca em um grafo, corresponde a função *h()* de outra busca e vice-versa, ou seja, uma estimativa dinâmica de custo. Desta forma, é possível reduzir o montante de informação armazenada e melhorar a estimativa de uma busca baseada em outras pesquisas. Em buscas bidirecionais a cooperação requer apenas que os valores sejam atualizados.

A visibilidade é a novidade do LCS* onde as bordas das pesquisas onde existem sobreposição entre a busca da origem e o caminho percorrido do destino (busca bidirecional)

podem ter suas estimativas dinâmicas ocultadas temporariamente quando as buscas entre ambas as frentes se mostrarem ótimas.

Tabela 2.2 – Comparativo entre os principais métodos de pesquisa.

	Implementação	*Velocidade*	*Complexidade*
DFS	Simples	Lento	$O(\|E\|)$
BFS	Simples	Lento	$O(n+\|E\|)$
Dijkstra	Simples	Lento	$O(n^2+\|E\|)$
A* (A Star)	Complexa	Rápido	$O(\log h*(x))$

Neste capítulo foram apresentados os principais métodos de pesquisa em grafos dentre os que tem garantia de encontrar um caminho caso exista um. Conforme a Tabela 2.2, dentre os métodos apresentados, somente o A estrela tem um desempenho relativamente bom com um resultado próximo do ótimo.

Neste capítulo foram apresentados conceitos gerais necessários para o melhor entendimento dos próximos capítulos. No capítulo seguinte será abordado o estado da arte dos roteadores globais que utilizam computação paralela.

3 ESTADO DA ARTE EM PARALELISMO PARA ROTEAMENTO GLOBAL

Neste capítulo serão abordados roteadores globais que fazem uso de computação paralela com o objetivo de acelerar o tempo de processamento desta tarefa.

A partir do estado da arte em roteamento global usando abordagens paralelas, distinguem-se dois métodos principais. Roteamento independente de cada rede (Han, 2013) e roteamento baseado em particionamento do circuito (Wu 2010):

- Roteamento independente de cada rede (Han, 2013): Consiste em considerar cada rede independentemente como uma linha de execução. Este método permite realizar o roteamento global usando um número grande de linhas de execução, ou seja, permite utilizar melhor os recursos computacionais como GPUs. Para rotear de forma independente são escolhidas as conexões distantes o suficiente, para que ao longo das possíveis combinações de um caminho *Manhattan* (distância entre 2 pontos em um sistema cartesiano, onde não existem diagonais, apenas rotas verticais e horizontais) estas não se sobreponham. Como nem todas as redes são distantes, temos blocos sequenciais que não permitem paralelização massiva.
- Roteamento baseado em particionamento do circuito (Wu 2010): Consiste em dividir o circuito em regiões atribuindo uma linha de execução para cada região. Este método gera um número menor de linhas de execução que o anterior. Para isto, temos que definir como delimitar as regiões. Após o roteamento parcial de cada região, resta o esforço computacional de interligar redes que pertençam a mais de uma região.

Ambos os métodos têm parcelas intrinsecamente sequenciais de processamento, seja ao rotear redes vizinhas, seja ao conectar as fronteiras de partição. Desta forma, os algoritmos para tratar estas partes sequenciais representam uma área na pesquisa de algoritmos paralelos que podem ajudar o roteamento global.

3.1 Roteadores que utilizam a Computação Paralela

A Tabela 3.1 apresenta uma lista dos principais roteadores acadêmicos que fazem uso de técnicas de computação paralela com o objetivo de acelerar a sua execução.

Os roteadores estão classificados pela metodologia de paralelização. As subseções seguintes serão abordam alguns destes trabalhos em maior profundidade.

Tabela 3.1 – Roteadores Globais que utilizam computação paralela.

Referência	*Nome do Roteador*	*Método de Paralelismo*
WU, 2010	PGRIP	Partition Level
QI, 2014	VFGR	Net And Partition Levels
HAN, 2014	-	Net Level
HAN, 2013	-	Net Level
LIU, 2013	NCTU-GR-2.0	Net Level
SHINTANI, 2013	-	Net Level
SHOJAEI, 2013	CGR	Pareto Algebra Instances

3.1.1 PGRIP

O PGRIP (WU, 2010) é um roteador global que utiliza programação inteira e paralelismo e é baseado no *Global Router IP* (GRIP) (WU, 2009). A proposta dos autores é de um roteador global que concorrentemente processe subproblemas correspondendo a regiões retangulares do circuito. E para isto o algoritmo utiliza uma formulação de programação inteira para realizar o roteamento global de cada subproblema.

Este trabalho está diretamente relacionado ao roteador global baseado em programação inteira GRIP. O processo de roteamento consiste em decompor o circuito em regiões retangulares e um posterior assinalamento das redes, de forma a gerar subproblemas de tamanho reduzido. E, então, é aplicado um algoritmo baseado em ILP para resolver de forma sistemática os subproblemas.

Desta forma, o GRIP consegue resolver o roteamento global com alta qualidade, mas tendo como efeito colateral um longo tempo de processamento.

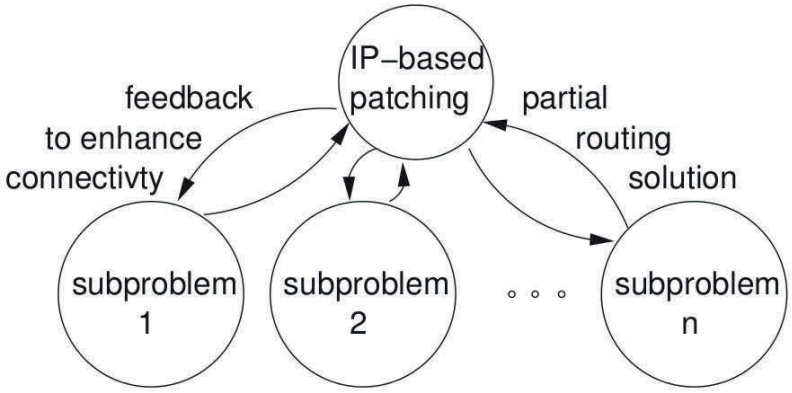
Uma forma encontrada para acelerar o processamento do roteador ocorre por meio da utilização de paralelismo, dividindo em *threads* diferentes a busca *branch and bound* de cada subproblema. Desta forma, as soluções encontradas podem ser eficientemente agregadas em uma solução final.

O método dos autores consiste em abordar este problema com uma fase de retrabalho que utiliza IP para conectar redes parcialmente solucionadas. Este retrabalho provê uma análise de congestionamento para cada subproblema de roteamento. Para, posteriormente, conectar estas redes.

Alguns dos desafios desta abordagem envolvem a eficiente decomposição do roteamento global em subproblemas, numa espécie de particionamento, conforme apresentado na Figura 3.1. Também existe a necessidade em manter a conectividade destes subproblemas, mesmo que para isto haja algum *overflow*.

Para realizar de forma eficiente a computação paralela, os autores criaram uma etapa de sincronização de forma a permitir que grandes porções de trabalho sejam realizadas de maneira descentralizada. Também, o método apresentado alimenta os nodos do *branch and bound* com as possíveis rotas que uma rede pode tomar, no final escolhendo uma que seja adequada o suficiente.

Figura 3.1 – Fluxo de geração dos subproblemas.



Fonte: Wu (2010).

Mais especificamente, os subproblemas são submetidos a uma etapa inicial com o objetivo de gerar um pequeno número de rotas candidatas. Após isto, cada subproblema envia para uma ou mais CPUs mestre, informações sobre a utilização de bordas de interconexão e outros parâmetros. Para isto, cada unidade de processamento mestre intermédia pares de subproblemas. Em seguida, são criados pinos virtuais que guiarão o roteamento dentro de cada partição do circuito e após o roteamento de cada subproblema, é realizada uma etapa legalização, de forma a minimizar o congestionamento do resultado.

Os resultados do PGRIP são apresentados na Tabela 3.2, aos quais foram utilizados os arquivos de *benchmark* dos ISPDs de 2007 e 2008. Para a obtenção dos resultados foi utilizada uma grade computacional com diversas máquinas com 2GB de memória cada.

Tabela 3.2 – Resultados para os *benchmarks* dos ISPD 2007/2008.

	PGRIP				GRIP			FGR		FastRoute		NTHU-Route	
	TOF	WL	WCPU	TCPU	TOF	WL(%)	TCPU	TOF	WL(%)	TOF	WL(%)	TOF	WL(%)
adaptec1	0	82.3	76	2101	0	-1.56	2247	0	7.00	0	9.60	0	7.38
adaptec2	0	83.4	76	2704	0	-1.24	2677	0	7.20	0	8.90	0	8.21
adaptec3	0	186.5	77	6319	0	-0.58	5168	0	6.61	0	8.87	0	7.15
adaptec4	0	173.2	79	5221	0	-0.52	5258	0	3.44	0	7.36	0	6.88
adaptec5	0	241.5	77	3175	0	-1.07	7133	0	7.13	0	10.79	0	7.20
newblue1	0	84.9	76	2306	0	-1.14	3076	526	9.97	0	7.46	0	6.71
newblue2	0	123.3	77	4192	0	-1.55	5228	0	4.73	0	9.11	0	8.43
newblue3	41K	156.3	82	14590	53K	-1.03	6768	30K	10.02	32K	14.17	31K	6.38
Média						-1.09			6.58		8.87		7.42
newblue4	132	124.9	77	2944	152	-0.44	3974	262	3.65	144	6.78	138	4.29
newblue5	0	223.8	80	4953	0	-0.44	6598	0	3.95	0	5.47	0	3.38
newblue6	0	172.0	78	2219	0	-0.88	5096	0	4.61	0	5.83	0	2.78
newblue7	54	338.4	86	4788	74	-0.83	5377	1458	3.37	62	5.17	68	4.22
bigblue1	0	54.0	76	956	0	-0.54	2770	0	5.81	0	6.72	0	3.49
bigblue2	0	86.5	77	3411	0	-0.64	3793	0	5.38	0	9.50	0	4.50
bigblue3	0	126.5	78	2690	0	-0.24	3448	0	4.20	0	3.24	0	3.22
bigblue4	176	221.1	82	3096	186	-0.22	4400	414	4.54	152	8.50	162	4.30
Média						-0.53			4.44		6.40		3.77

Fonte: Wu (2010).

3.1.2 VFGR

Neste trabalho os autores propõem um roteador global 3D utilizando computação paralela e um modelo de congestionamento local apurado (QI, 2014).

Em observação de circuitos projetados por ferramentas comerciais e acadêmicas, percebe-se uma correlação muito grande entre as características dos recursos utilizados com o tamanho de uma rede. Geralmente redes curtas tendem a consumirem recursos de uma pequena região utilizando camadas mais baixas de metal, diferentemente de redes longas que tendem a utilizar um grande escopo e camadas de metal mais elevadas. Considerando este aspecto, o roteador global 3D proposto pelos autores implementa diversos níveis de hierarquia com diversos tamanhos de células de roteamento. As redes são distribuídas em cada nível de hierarquia de acordo com o tamanho de seu *bounding box*.

O roteamento é realizado em ordem crescente de níveis, começando pelo nível mais baixo de metal até o mais elevado. Desta forma, as redes que não conseguem ser roteadas em um nível são entregues a níveis mais elevados. O paralelismo é explorado em cada nível, através das duas principais metodologias, particionamento e roteamento independente de redes. Desta forma, estas duas técnicas de paralelismo implementadas tem como foco arquiteturas computacionais do tipo MIMD.

Uma característica do processo é que os níveis mais elevados não conseguem uma melhoria de desempenho muito expressiva, pois as células de roteamento são maiores e, consequentemente, o espaço de pesquisa ser consideravelmente menor. Assim as chances de paralelismo por particionamento, também, são menores. Para maximizar o uso dos recursos computacionais, é utilizado, também, a abordagem de roteamento independente de redes para rotear as redes que não tem dependência de dados entre si de forma independente, ou seja, cada rede é processada isoladamente em uma *thread* desde que nesta fila de processamento não haja dependência de dados entre os elementos.

Os resultados do VFGR são apresentados na Tabela 3.3, onde pode-se perceber que o tempo de execução é consideravelmente mais lento que o roteador global NCTU, que será abordado na próxima subseção. Nos testes utilizou-se uma máquina Intel com 8 núcleos de processamento e 24GB de memória.

Tabela 3.3 – Comparativo de execução entre roteadores globais.

	VFGR					NCTU-GR 2.0				BFG-R			
	OF	WL	via	E-CPU	CPU	OF	WL	via	CPU	OF	WL	via	CPU
superblue2	0	9.32	5.25	6.93	45.07	0	9.21	5.29	16.40	0	9.61	5.35	97.48
superblue3	0	5.5	4.82	5.48	33.98	0	5.42	4.81	11.05	0	5.99	5.17	49.83
superblue6	0	5.36	4.93	3.23	19.40	0	5.3	4.94	6.55	0	5.54	5.21	19.60
superblue7	0	6.48	7.29	3.87	22.42	0	6.46	7.31	5.90	0	7.33	7.92	25.32
superblue9	0	3.95	3.96	2.17	12.35	0	3.91	4.01	4.23	0	4.40	4.23	13.65
superblue11	0	5.22	4.2	2.15	12.03	0	5.21	4.37	2.82	0	5.49	4.37	14.68
superblue12	0	5.55	6.71	2.20	12.53	0	5.49	6.68	4.08	0	6.05	6.79	11.67
superblue14	0	3.54	3.32	3.50	17.85	0	3.51	3.30	3.22	0	3.69	3.61	18.48
superblue19	0	2.34	2.22	1.02	5.50	0	2.35	2.29	1.13	0	2.37	2.40	6.12
Média	0	1.00	1.00	0.17	1.00	0	0.99	1.01	0.29	0	1.07	1.06	1.24

Fonte: Qi (2014).

3.1.3 NCTU-GR 2.0

Neste trabalho (LIU, 2013), os autores propõem dois algoritmos de roteamento de labirinto baseados em *bounded-length* que conseguem ser mais rápidos que os métodos convencionais. Optimal-BLMR (Optimal Bounded-Length Maze Router) e Heuristic-BLMR. Além disso, utilizam roteamento monotônico para redes de menor comprimento de fio. Este trabalho, também, propõe uma implementação paralela que diferente da abordagem de particionamento pois utiliza concorrência de tarefas.

Para superar as limitações do modelo de particionamento do roteamento global, os autores propõem reduzir o problema de GR em maximização e minimização de

compartilhamento de recursos. Os autores, por sua vez, propõem um método que considere o comprimento de fio.

Desta forma, o BLMR consiste em limitar o espaço de pesquisa em busca de um processamento mais rápido e com maior possibilidade de paralelismo. A formulação do problema BLMR é dada por 4-tuplas (*s*, *t*, *G*, *L*), *s* e *t* denotam a origem e destino, *G* representa a grade do grafo e *L* (que não pode ser menor que a distância Manhattan entre os pinos de origem e destino) é a restrição do *bounded-length*.

BLMR trabalha de forma muito semelhante ao método *constrained shortest path* (CSP), com a diferença de que o primeiro trabalha com grades bidimensionais e com no máximo 4 vizinhos.

A técnica *Optimal-BLMR* adota duas diferentes políticas para realizar o roteamento. Potencial de comprimento de fio e não existe o descarte imediato de possíveis rotas parciais, de forma a procurar o mínimo global.

Já a técnica BLMR heurística não garante uma solução ótima, porém consegue ter um tempo de processamento muito mais rápido que a versão ótima do BLMR, em média 270 vezes mais rápido. É a versão heurística do BLMR que é utilizada efetivamente pelo roteador NCTU-GR 2.0.

Uma das diferenças, se comparado com a versão ótima, é que a versão heurística preserva apenas um caminho entre o nodo inicial e o nodo atual, de forma que o desafio é como escolher este caminho. Uma das métricas para esta decisão está relacionada ao tempo de propagação de cada possibilidade de rota, as rotas que tem uma sobra de tempo maior tendem a serem escolhidas, em detrimento de rotas com maior atraso.

A paralelização do NCTU-GR 2.0 ocorre da seguinte maneira. Cada rede de dois pinos é definida como uma tarefa em uma fila em que cada unidade de processamento vai construindo cada rede. De forma a evitar uma redução do grau de paralelismo, existe uma etapa de sincronização das *threads* apenas ao final de cada iteração.

Para minimizar condições de corrida entre os recursos de roteamento entre as *threads*, situação não existente no método de particionamento, é realizada uma análise de congestionamento ao final de cada iteração servindo de base para quais redes podem ser processadas.

Assim sendo, a heurística aplicada visa prever possíveis colisões de dados na execução. Assim, as *threads* notificam umas as outras, se determinadas arestas da grade de roteamento estiverem utilizadas por redes já implementadas em uma iteração anterior. Com isso, cada *thread* estima o risco de utilização destas arestas. Para os outros casos de condição

de corrida, é utilizado um modelo de bloqueios de execução, inferindo em uma pequena redução de desempenho para realizar o processamento das redes.

Os resultados são apresentados nas Tabelas 3.4 e 3.5 e, para a obtenção dos mesmos, foi utilizado um servidor Intel Xeon com 8 núcleos de processamento rodando a frequência de 3GHz e com 32GiB de memória principal. Os resultados foram divididos entre os *benchmarks* roteáveis e os de roteamento difícil. Os *benchmarks* de roteamento difícil são aqueles que a solução de roteamento sem congestionamento é muito difícil ou até impossível de ser encontrada. Desta forma o NCTU-GR 2.0 tem um desempenho muito próximo em tempo de execução, tendo uma redução de WL considerável se comparado ao NTHU 2.0.

Tabela 3.4 – Comparativo de desempenho entre diversos roteadores globais com a implementação apresentada sobre os *benchmarks* sem *overflow* dos ISPD 2007 e 2008.

	NTHU-Route2.0		FastRoute 4.1		NCTU-GR 2.0	
	WL	CPU	WL	CPU	WL	CPU
adaptec1	53.49	4.86	53.73	3.31	53.50	3.90
adaptec2	52.31	1.42	52.17	0.95	51.69	1.45
adaptec3	131.11	6.16	130.82	3.69	130.35	4.88
adaptec4	121.73	2.08	121.24	1.25	120.67	2.28
adaptec5	155.55	11.95	155.81	6.70	154.70	9.07
newblue1	46.53	4.07	46.33	12.01	45.99	3.63
newblue2	75.85	1.17	75.12	0.85	74.88	0.90
newblue5	231.73	10.88	230.94	9.82	230.31	15.03
newblue6	177.01	10.34	177.87	8.78	176.87	9.67
bigblue1	56.35	6.93	56.64	4.22	56.56	6.35
bigblue2	90.59	6.47	91.18	12.12	89.40	11.18
bigblue3	130.76	3.91	130.04	2.06	129.66	4.38
Ratio	1.01	1.90	1.01	1.77	1.01	1.92

Fonte: Liu (2013).

Tabela 3.5 – Comparativo de desempenho entre diversos roteadores globais com a implementação apresentada sobre os *benchmarks* de difícil roteamento dos ISPD 2007 e 2008.

	FastRoute 4.1				NCTU-GR				NTHU-Route 2.0			
	MO	TO	WL	CPU	MO	TO	WL	CPU	MO	TO	WL	CPU
newblue3	736	31276	108.40	15.99	198	31808	104.28	131.43	204	31454	106.49	64.97
newblue4	2	136	130.46	65.23	2	134	126.79	40.92	4	138	130.46	52.01
newblue7	4	54	353.38	868.74	2	114	338.63	71.52	2	62	353.35	50.28
bigblue4	2	130	230.24	93.25	2	164	223.99	65.37	2	162	231.04	52.63
Ratio	1.95	0.88	1.03	3.89	1.01	1.19	0.99	1.30	1.26	0.96	1.02	1.23

Fonte: Liu (2013).

3.1.4 *Exploring High-Throughput Computing Paradigm for Global Routing*

O roteador global apresentado neste artigo tem um sistema híbrido de execução, dividindo as tarefas entre CPU e GPU. Além disto, utiliza a grande capacidade de *throughput* disponível nas GPUs (HAN, 2013).

Uma das características do roteamento global é que existe um grande compartilhamento de recursos ao longo do processo, o que dificulta a sua extração de paralelismo. No entanto, devido ao grande volume de dados envolvidos no processo existe uma parcela considerável de paralelismo que depende da reformulação de métodos atuais e intrinsecamente sequenciais.

Para tanto, neste trabalho os autores implementaram uma metodologia capaz de extrair uma granularidade de paralelismo suficientemente fina para a sua implementação em arquiteturas massivamente paralelas, como GPUs, além de utilizarem também arquiteturas de múltiplos núcleos. Para isto, o roteador implementado emprega uma técnica nova de paralelismo, onde as redes sem dependência de dados entre si são computadas paralelamente.

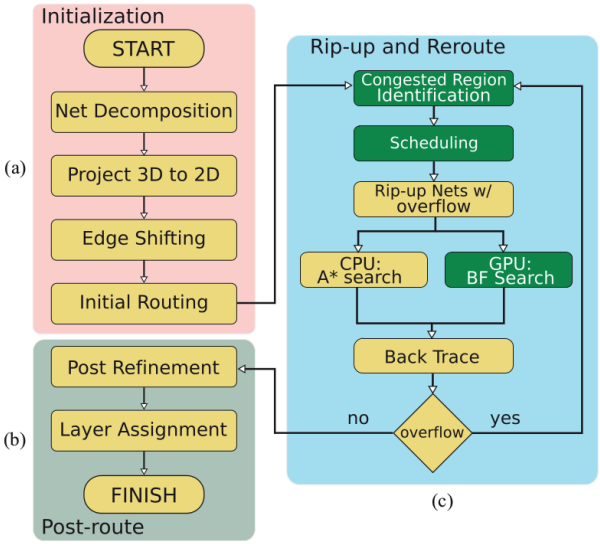
Esta técnica, chamada de *Net Level Concurrency* (NLC) consiste em criar uma região de *bounding box* em torno das redes e criar uma fila de processamento com as redes sem dependência de dados entre si.

O roteador global executado na GPU utiliza o método de busca BFS e o utilizado na CPU utiliza o algoritmo de roteamento por labirinto A*. Esta abordagem mista é necessária para despachar os problemas que necessitam de alta latência e pouca largura de banda para a CPU, enquanto que os problemas com baixa latência e necessidade de muita largura de banda são encaminhados para a GPU. Os autores acreditam que esta é a chave para manipular o roteamento global em hardware massivamente paralelo.

Os autores desenvolveram, também, um algoritmo escalonador que tem por objetivo explorar o NLC no roteamento global. Este escalonador tem por objetivo analisar as dependências de dados das redes e direcionar o fluxo de processamento para a CPU ou a GPU conforme o comprimento de fio das redes, pois redes menores tendem a serem processadas na GPU, enquanto que as maiores são processadas na CPU devido à carga de computação sequencial necessária.

A Figura 3.2 mostra o fluxo de execução do roteador implementado pelos autores do trabalho.

Figura 3.2 – Fluxo de execução do roteador global.



Fonte: Han (2013).

Os resultados mostram que ao executar a ferramenta com uma GPU nVIDIA com arquitetura Fermi, 99% das redes puderam ser roteadas pela GPU e apenas 1% pela CPU. A execução do roteador ocorre em média 3.15 vezes mais rapidamente ao habilitar o uso de múltiplos núcleos (CPU quad core), comparado ao NTHU Router 2.0 e o desempenho é 4.01 vezes superior ao NTHU 2.0 quando habilitado o uso de GPU. Os resultados são apresentados na Tabela 3.6. Os resultados foram obtidos em uma máquina Intel com 4 núcleos de processamento rodando a 2,4GHz e com memória principal de 8GB.

Tabela 3.6 – Comparativo de execução e melhoria de desempenho.

	Parallel GR (4 CPU)		Parallel GR (4 CPU+GPU)		NTHU 2.0		Speedup	
	WL_a	$Runtime_b$	WL_a	$Runtime_b$	WL_a	$Runtime_b$	4 CPU	4 CPU+GPU
adaptec1	5.43E6	2.90	5.44E6	2.07	5.34E6	9.95	3.43	4.81
adaptec2	5.29E6	0.70	5.30E6	0.63	5.23E6	2.1	3.00	3.33
adaptec3	1.31E7	3.53	1.31E7	3.05	1.31E7	10.86	3.08	3.56
adaptec4	1.24E7	0.95	1.24E7	0.65	1.22E7	2.5	2.63	3.85
adaptec5	1.55E7	9.98	1.55E7	8.12	1.55E7	21.9	2.19	2.70
newblue1	4.70E6	2.87	4.70E6	1.88	4.65E6	6.2	2.16	3.30
newblue2	7.79E6	0.66	7.81E6	0.65	7.57E6	1.1	1.67	1.69
newblue5	2.38E7	5.38	2.38E7	4.42	2.32E7	19.1	3.55	4.32
newblue6	1.80E7	4.12	1.80E7	3.78	1.77E7	17.5	4.25	4.63
bigblue1	5.63E6	3.10	5.63E6	2.86	5.59E6	13.1	4.22	4.58
bigblue2	9.10E6	2.31	9.05E6	2.09	9.06E6	8.4	3.64	4.02
bigblue3	1.30E7	1.09	1.30E7	1.01	1.31E7	4.4	4.04	4.37
Média	1.01_c	–	1.01_c	–	1	–	3.15	4.01

Fonte: Han (2013).

Os 4 roteadores globais apresentados neste capítulo fazem uso de técnicas de computação paralela para tornar o processamento do roteamento mais rápido. Destaca-se a técnica de *net level*, que é abordada na maioria dos trabalhos atuais que utilizam paralelismo.

4 METODOLOGIA PARA A PARALELIZAÇÃO DO ROTEAMENTO GLOBAL

Como explicado anteriormente, o objetivo dessa dissertação é estudar a possibilidade de paralelizar de forma eficiente em arquiteturas massivamente paralelas a etapa de Roteamento Global. Este capítulo aborda a metodologia empregada no estudo.

A metodologia compreende 4 passos: (1). A análise de um roteador sequencial já existente no grupo de pesquisa na UFRGS; (2). A análise de dados provenientes do ISPD 2008 (NAM, 2008) em termos de comprimento de redes; (3). O desenvolvimento de um colisor que permite detectar a dependência de dados ao interior de cada grupo e (4). A estimativa do custo temporal (*Overhead*) para a detecção da dependência de dados no colisor.

4.1 Roteador Global presente no Grupo de Pesquisa

A primeira etapa do trabalho estudou o roteador já existente no grupo de pesquisa (REIMANN 2013). Este roteador é capaz de tratar o roteamento global de circuitos VLSI, utilizando diversas técnicas existentes na literatura. Este Roteador Global tem o objetivo de servir de padrão para o teste de nossas novas técnicas de paralelismo.

Dentre as principais características, o Roteador Global não utiliza nenhuma técnica de avaliação de congestionamento, como nos trabalhos já existentes na literatura e nos trabalhos participantes das competições dos ISPDs 2007 e 2008. Existem 2 modalidades de execução do Roteador Global:

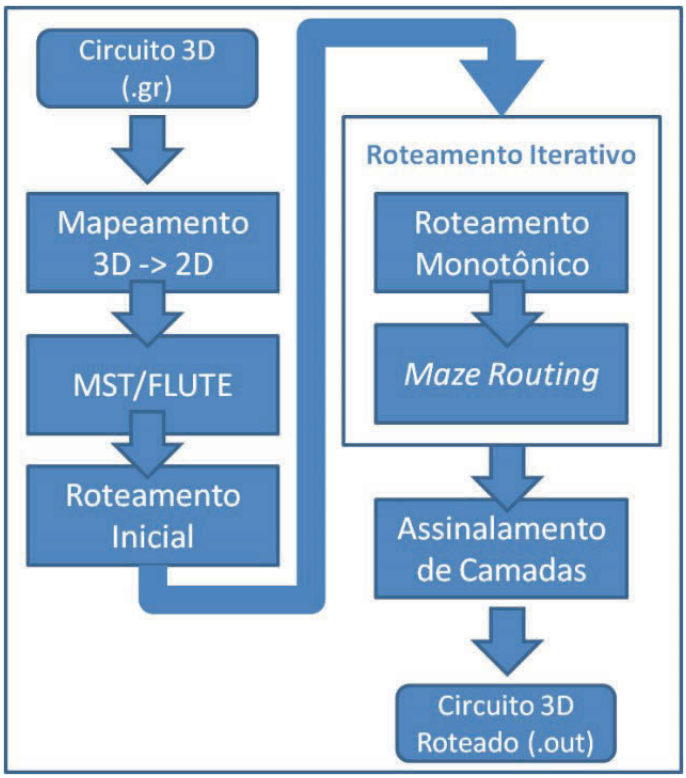
- GR WL: Menor comprimento de fio possível;
- GR RT: Execução rápida, ignorando algumas verificações.

As estruturas de dados da ferramenta englobam a lista de todas as redes e fios e a grade de roteamento. Esta última, representada por uma matriz bidimensional, onde cada aresta tem uma capacidade específica (as bordas têm capacidade zero).

Ao realizar o roteamento inicial, conforme o fluxo de execução apresentado na Figura 4.1, todos os fios com 2 unidades de comprimentos são roteados utilizando a técnica de *patern routing*, que consiste na prática em gerar caminhos em “L” e em “I”. Esta abordagem inicial é muito mais ágil se comparada ao *maze routing*. A versão RT realiza este tipo de roteamento para todos os fios, independente de seu comprimento. Enquanto que a versão WL

consegue entregar de 20% a 60% de fios roteados, sem haver congestionamento. Já na versão RT a taxa de entrega de fios roteados varia de 75% a 98% nesta fase do roteamento.

Figura 4.1 – Fluxo de execução do roteador global do grupo de pesquisa em EDA da UFRGS.



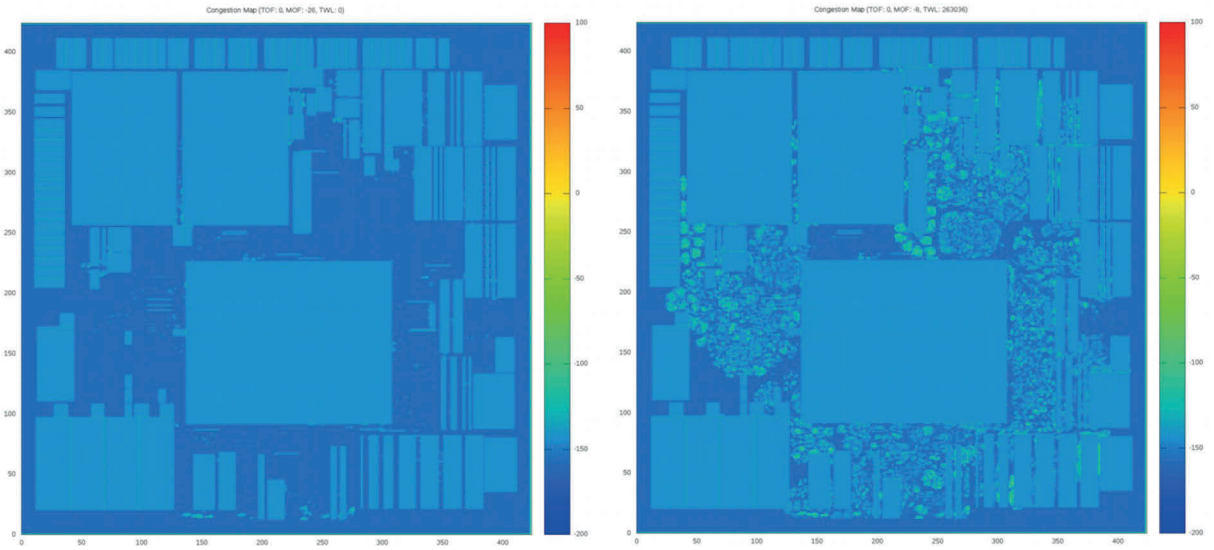
Fonte: REIMANN (2013, p 84).

Os mapas de congestionamento das versões WL e RT para o roteamento inicial do circuito adaptec2 podem ser vistos nas Figuras 4.2 e 4.3, neles é possível ver a diferença entre as duas abordagens nessa etapa do processo de roteamento.

Após o roteamento inicial, é dado início à fase iterativa do roteamento, utilizando roteamento monotônico e *maze routing*. No roteamento monotônico, não é permitido criar violações, ou seja, não é permitida a criação de *overflow*. Existem diferentes configurações para a capacidade implementadas nas versões RT e WL, de forma a tornar mais rápida a versão RT e a viabilizar a execução do roteador devido ao menor grau de liberdade encontrado nas rotas do roteamento monotônico.

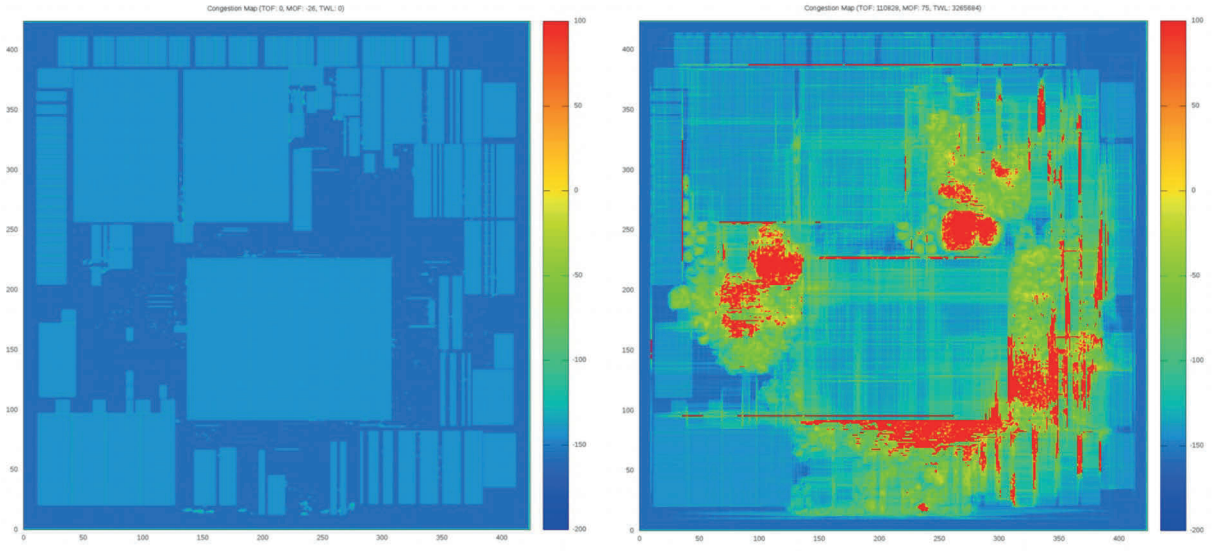
A utilização do *maze routing* ocorre de duas formas, limitada (como no roteamento monotônico) e irrestrita, considerando apenas os custos de cada aresta, desconsiderando a capacidade. A utilização da primeira ocorre apenas quando o *overflow* esteja dentro de limites preestabelecidos, não sendo necessariamente executado.

Figura 4.2 – Mapa de congestionamento inicial na versão WL.



Fonte: REIMANN (2013, p 86).

Figura 4.3 – Mapa de congestionamento inicial na versão RT.



Fonte: REIMANN (2013, p 86).

Ao início de cada iteração, é necessária a identificação de que fios serão desfeitos e re-roteados. A sua identificação é realizada com base no *overflow* da iteração anterior. Como nas primeiras iterações existe um grande número de fios e redes que precisam ser re-roteados, ao passo em que cada fio é desfeito, o mesmo é analisado de forma a saber se ainda é necessário desfazê-lo.

Para realizar o roteamento iterativo utilizando o *maze routing* a ordenação dos fios é fator importante. Esta ordenação ocorre, principalmente, com base no congestionamento médio das arestas.

Por fim, os critérios de término da fase iterativa do roteador global são respectivamente:

- Encontrar uma solução sem violações;
- Passadas 20 iterações consecutivas sem haver redução do *overflow*;
- O tempo de execução se aproximar de 24 horas.

O capítulo seguinte apresenta os pontos possíveis para se alterar o código para paralelizá-lo com um novo fluxograma para o roteamento global paralelo. Antes disto, será abordado na próxima seção o próximo passo da metodologia: A análise de dados provenientes dos ISPDs 2007 e 2008 (NAM, 2007)(NAM, 2008) em termos de comprimento de redes.

4.2 Análise de Dados

Com o intuito de estimar a validade de nossa hipótese, analisou-se os arquivos de *benchmarks* do ISPD 2008. “Estes *benchmarks* são todos derivados de circuitos reais projetados pela indústria e apresentam desafios do design físico moderno como escalabilidade, variedade de *floorplans*, *movable macro handling* e negociação de congestionamento (*congestion mitigation*)” (NAM, 2007).

Na Tabela 4.1 são mostrados os circuitos de *benchmark* utilizados nos ISPDs de 2007 e 2008.

Construiu-se uma peça de software que realiza a leitura de cada *benchmark*, enumerando as redes. Para cada rede, criou-se uma região chamada *bounding box*, que contém todos os pinos associados a ela.

A região de *bouding box* é um retângulo que engloba todos os pinos referentes a uma rede. Dentro desta região é onde provavelmente está a solução do roteamento para a rede.

Tabela 4.1 – Circuitos de *benchmark* utilizados nos ISPDs de 2007 e 2008.

Circuíto	Posicionador Utilizado	Densidade	Tamanho do Circuito em *Tiles*	Tamanho dos *Tiles*	Camadas de Metal	Nro. De Nets	HPWL
adaptec1	Capo	70	324 x 324	35	6 (3/3)	176715	3000320
adaptec2	mPL6	60	424 x 424	35	6 (3/3)	207972	2882254
adaptec3	Dragon	70	774 x 779	30	6 (3/3)	368494	8619596
adaptec4	APlace	60	774 x 779	30	6 (3/3)	401060	8175006
adaptec5	mFAR	50	465 x 468	50	6 (3/3)	548073	8896706
bigblue1	Capo	60	227 x 227	50	6 (3/3)	196885	2986719
bigblue2	mPL6	60	468 x 471	40	6 (3/3)	428968	4049521
bigblue3	APlace	70	555 x 557	50	8 (4/4)	665629	7170444
bigblue4	FastPlace	70	403 x 405	80	8 (4/4)	1133535	10489255
newblue1	NTUplace	80	399 x 399	30	6 (3/3)	270713	2079947
newblue2	FastPlace	90	557 x 463	50	6 (3/3)	373790	4191219
newblue3	KraftWerk	80	973 x 1256	40	6 (3/3)	442005	6998467
newblue4	mPL6	50	455 x 458	40	6 (3/3)	531292	7357235
newblue5	NTUplace	50	637 x 640	40	6 (3/3)	891920	12357104
newblue6	mFAR	80	463 x 464	60	6 (3/3)	835267	8823094
newblue7	KraftWerk	80	488 x 490	80	8 (4/4)	1647410	16284051

Fonte: ISPD 2008.

Então, juntou-se as redes em grupos com a mesma ordem de tamanho, ou seja, redes pequenas são agrupadas com outras redes pequenas. Mais precisamente, estes grupos estão divididos em ordem crescente de comprimento de fio em uma escala de potências de 2^{2n}, de forma a separar e permitir antes o processamento de redes curtas.

O ordenamento das redes tende a seguir uma ordem crescente, onde as redes curtas tendem a serem processadas primeiro e também tendem a ocuparem camadas de metal mais baixas.

Este é o primeiro passo para o nosso pré roteamento global. Desta análise pode-se determinar a porcentagem de cada grupo e, com isso, avaliar a existência de uma quantidade considerável de redes de menor comprimento de fio. Cada grupo será processado em um colisor.

4.3 Colisor Proposto

O próximo passo consiste em testar a colisão interna em cada grupo de redes. Estas colisões em nível de roteamento global são sinônimos de dependência de dados para o processamento paralelo de grafos. O objetivo deste colisor é de testar o maior número de redes no menor tempo possível considerando um único grupo de redes. Este teste é uma análise combinatória entre as redes, com o objetivo de detectar dependências de dados.

Três diferentes implementações do colisor foram executadas, com o objetivo de descobrir o *overhead* do processo.

a) Sem grupos, analisando todas as redes sem separação em grupos.
b) Agrupadas de forma sequencial, dividindo as redes em grupos de forma a reduzir o espaço de pesquisa.
c) Agrupadas utilizando processamento paralelo, dividindo as redes em grupos e utilizando a tecnologia OpenMP para realizar o processamento em paralelo.

Com base nos resultados desse colisor, que serão abordados em maior profundidade na sessão 5.3, pode-se definir que a quantidade de passos de processamento pode ser reduzida consideravelmente ao utilizar esta técnica de aceleração do processamento do roteamento global, o que pode ajudar a validar a nossa hipótese.

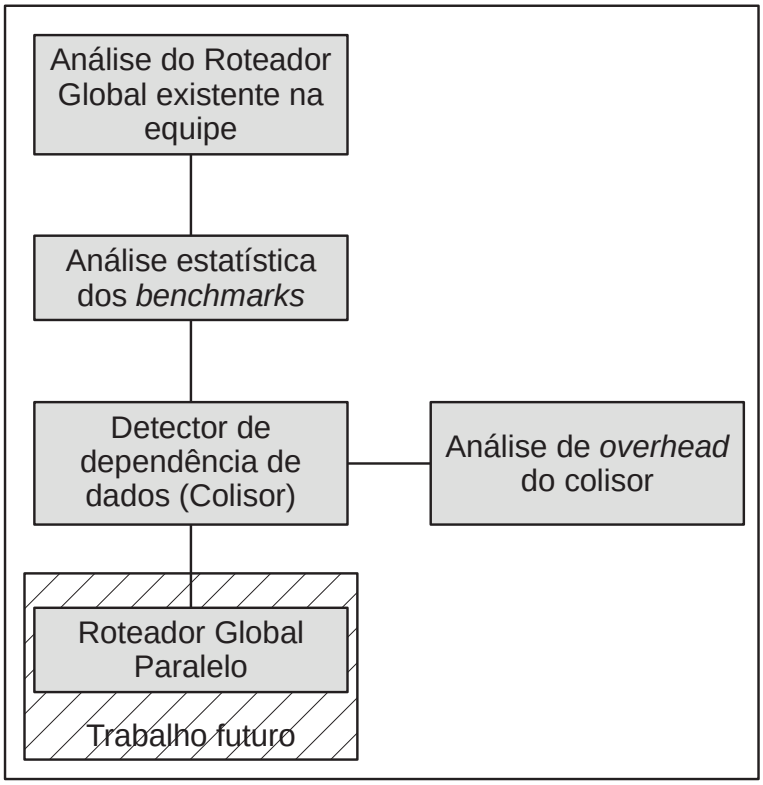
4.4 Análise do *Overhead* do Colisor de Redes Proposto

Por fim, a forma utilizada para averiguar a viabilidade da implementação final de um roteador paralelo é a medição do tempo de execução do colisor. Esta medição de tempo, deve ser menor que o *speedup* da implementação final para que este método seja considerado eficiente.

Para isso, mediu-se o tempo de execução de todo o processo para estudar a viabilidade deste agrupamento. Os testes foram realizados em um computador Sandy Bridge-E, com 12 núcleos de processamento (6 núcleos físicos), com 64 GB de memória RAM.

O método desta dissertação pode ser resumido por meio da Figura 4.4. Onde primeiro foi realizada uma análise do código do RG presente no grupo de pesquisa, com o objetivo de verificar a viabilidade de uma implementação paralela no mesmo. Na sequência foi realizado um estudo sobre os arquivos de benchmark, para averiguar o potencial de paralelismo através do número de redes curtas. Feito isto, foi escrito o colisor de redes e por fim verificado o seu tempo de execução:

Figura 4.4 – Fluxo apresentado neste trabalho.



No próximo capítulo são abordados os resultados e a discussão do trabalho, onde são apresentados os rumos para o futuro desta pesquisa em paralelismo aplicada a etapa de roteamento global de circuitos integrados.

5 RESULTADOS E DISCUSSÕES

Este capítulo apresenta a análise dos resultados obtidos nas implementações realizadas nesta dissertação. Dividiu-se o capítulo nas seções que abordam resultados sobre a análise do código fonte do Roteador Global presente no grupo de pesquisa, análise de dados dos arquivos de *benchmark* dos ISPDs de 2007 e 2008, implementação do colisor de redes, *overhead* da implementação do colisor no roteador global e uma subseção que discute estes resultados ao final do capítulo.

5.1 Análise do Roteador Global Presente no Grupo de Pesquisa

A análise do código fonte do roteador global existente no grupo de pesquisa, mostra um código bastante estruturado e com potencial real de incorporação da técnica de paralelismo apresentada neste trabalho. Pois o roteador realiza o ordenamento de redes, bastando alterar o código com o objetivo de alterar a ordem de execução das redes. Foi também realizada a compilação deste roteador para Linux, de forma a avaliar diretamente o tempo de execução em outros computadores, comprovando a característica de ser uma peça de software sequencial.

5.2 Análise de Dados dos Arquivos de Benchmarks dos ISPDs 2007/2008

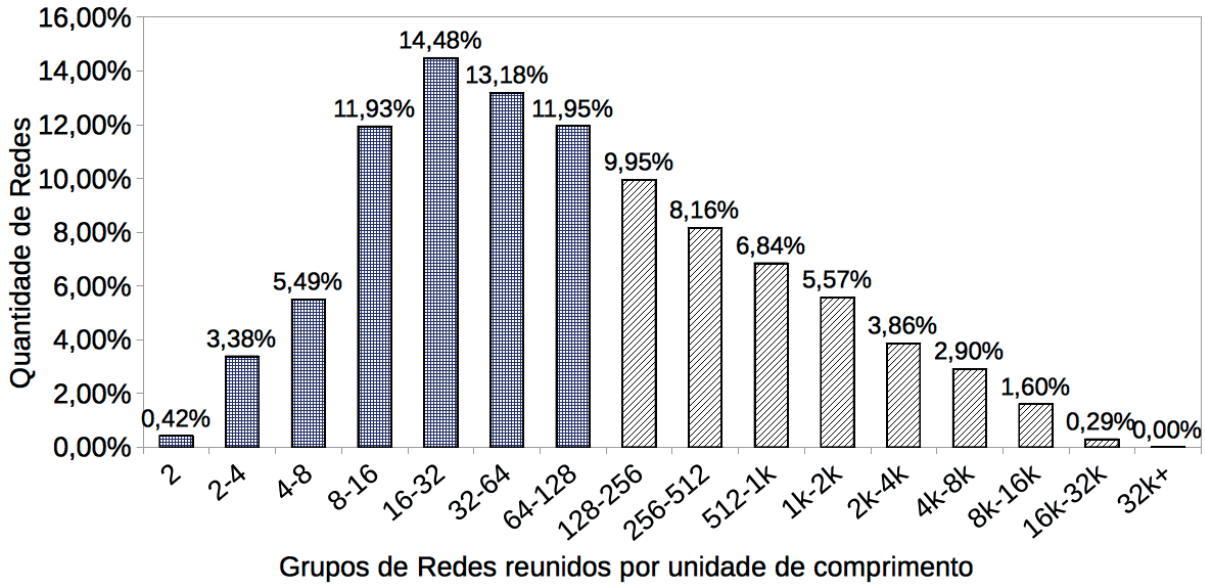
A Figura 5.1 mostra os resultados da análise de dados. O grupo predominante possui um semiperímetro entre 16-32. De fato, 61% das redes tem tamanho menor que 128. Isto sugere um grande potencial para nossa intenção de agrupar as redes. Também demonstra empiricamente, que existe uma quantidade grande de redes curtas, grupo de redes onde é mais fácil encontrar redes sem conflito de dados entre si.

Para a obtenção de resultados o software que engloba a análise dos *benchmarks* e o colisor de redes, o seguinte ambiente computacional foi utilizado:

CPU Intel core i7 Sandy Bridge 3930k com 6 núcleos de processamento, com 2 *threads* por núcleo (totalizando 12 *threads* simultâneas). Esta máquina tem 64 Gb de memória principal instalada.

Para a execução do aplicativo foram realizadas 10 execuções para os circuitos maiores e 20 execuções para os circuitos menores. O resultado é determinístico, desta forma, o desvio padrão dos testes mostra a variação do tempo de execução devido a fatores do sistema operacional (Linux Ubuntu Server 12.04) e da configuração dos níveis de memória cache durante a execução.

Figura 5.1 – Distribuição de redes em cada grupo. Os grupos são distribuídos pelo tamanho do semiperímetro das bounding box. Fios menores que 128 unidades de comprimento representam 61% de todas as redes (Em azul, a esquerda no gráfico).



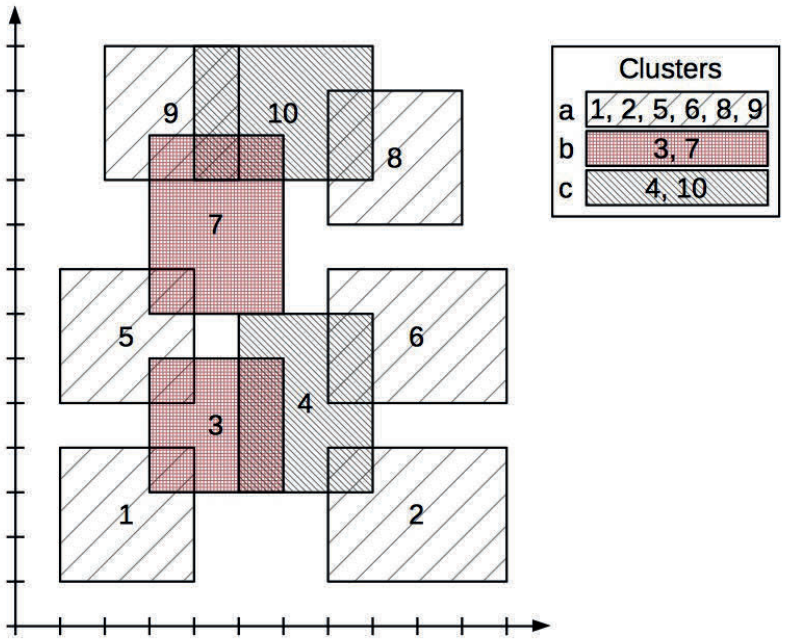
Esta pesquisa considera essencial esta análise mais detalhada dos arquivos de *benchmark*, pois em outros trabalhos apresentados, tais como (HAN, 2013), apresenta-se apenas um histograma sem valores detalhados a respeito da distribuição das redes em relação ao comprimento de fio e a sua real representatividade.

5.3 Colisor Proposto

A Figura 5.2 mostra um exemplo de como o colisor proposto trabalha em um circuito simples de 10 redes dentro de um mesmo grupo. O maior grupo de redes sem colisão é composto das redes 1, 2, 5, 6, 8 e 9. Estas 6 redes podem ser divididas em 6 linhas de execução. O restante das redes pode ser agrupada em duas outras linhas de execução, uma

com as redes 3 e 4, e outro grupo com as redes 7 e 10. Neste exemplo, o roteamento de 10 redes dentro de um mesmo grupo pode ser realizado em três etapas de processamento paralelo. A detecção de dependência de dados dentro de um grupo de tamanho determinado pode reduzir consideravelmente o tempo de processamento das redes.

Figura 5.2 – Exemplo de identificação de dependência de dados entre *bounding box* de redes.



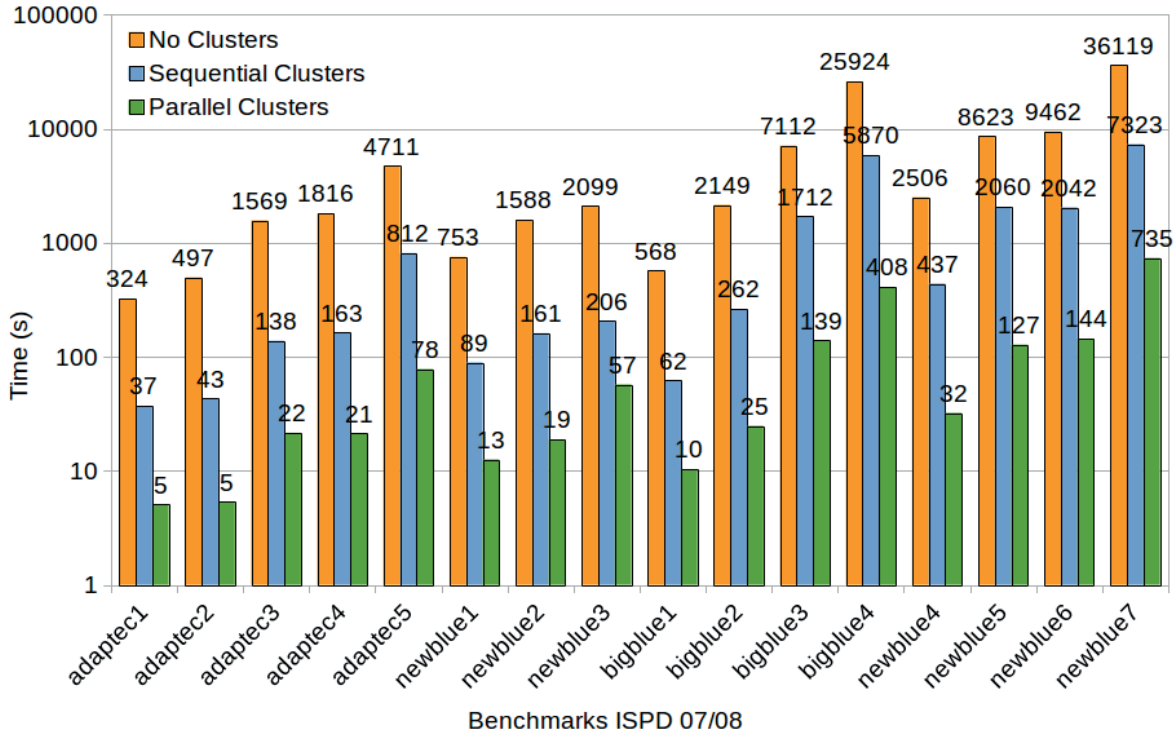
Grande parte das redes independentes aparecem nos primeiros ciclos de verificação. Nas interações seguintes, o tamanho do grupo tende a reduzir linearmente e o tempo de processamento diminui geometricamente. Isto mostra que esta metodologia, como já aplicada por (HAN, 2013) e (HAN, 2014) tem um real potencial, porém ainda não devidamente explorado pelos trabalhos atuais, devido ao tempo necessário no processo de detecção de dependência de dados. Este tempo de processamento, é abordado com mais detalhes na próxima seção.

5.4 Análise do *Overhead* do Colisor de Redes Proposto

A Figura 5.3 mostra o tempo de execução de cada uma das implementações do detector de colisões: Sem grupos, Agrupadas de forma sequencial, Agrupadas utilizando

processamento paralelo. De acordo com os tempos de execução obtidos pela implementação paralela, é possível analisar todas as redes, utilizando melhor os recursos computacionais de arquiteturas massivamente paralelas.

Figura 5.3 – *Overhead* em valores absolutos para três implementações do detector de colisões: A. Sem grupos, B. Grupos sequenciais, e C. Grupos com paralelismo. Podemos verificar que existe um ganho significativo ao reduzirmos o espaço de pesquisa e ao utilizarmos o paralelismo.



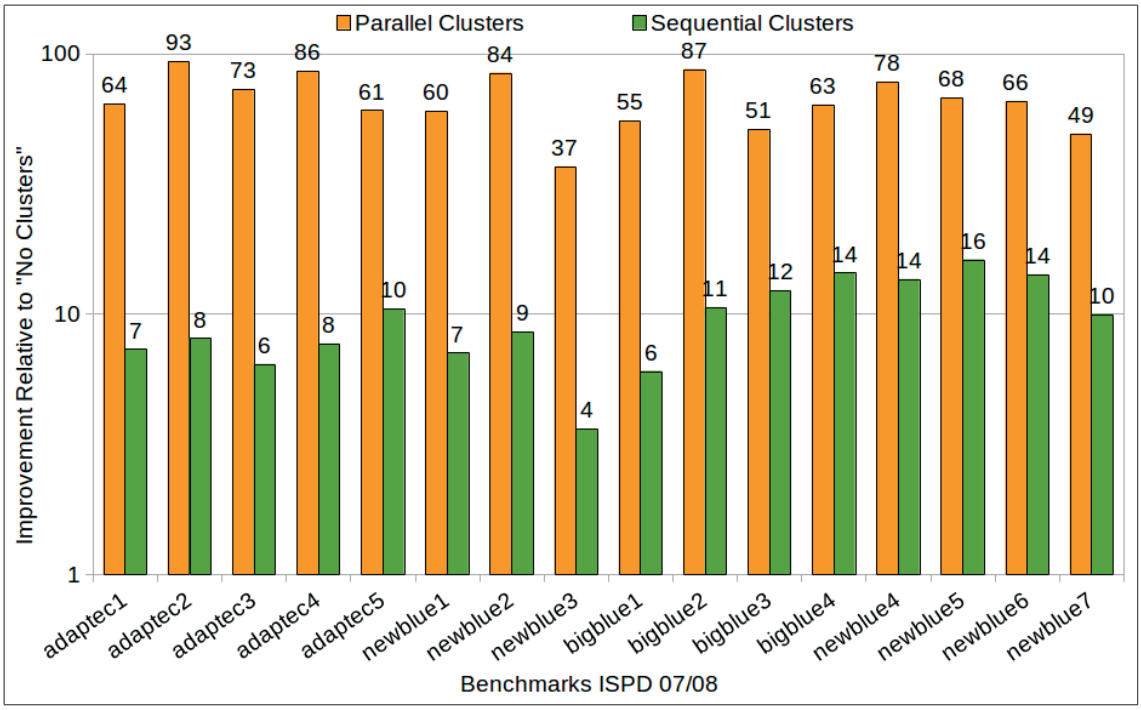
O ambiente computacional utilizado é o mesmo da subseção 5.3, onde foram realizadas 10 execuções para cada circuito de *benchmark*.

A Figura 5.4 mostra a relação entre os tempos de execuções das 3 implementações. O agrupamento das redes reduz o tempo de processamento em 10 vezes (em média). Paralelizar o processo de agrupamento reduz o *overhead* em média 67 vezes. Por exemplo, o pré-roteamento do *benchmark* adaptec2 ocorre 100 vezes mais rápido que a implementação inicial.

Isto mostra uma importante redução no tempo de processamento, através de uma divisão do espaço de pesquisa. Desta forma, o pior tempo possível transita na faixa de 12 minutos de execução, informação que tende a viabilizar a pesquisa de detecção de

dependência de dados em todas as redes e não em uma pequena parcela como em (HAN, 2013).

Figura 5.4 – Valores de *speed up* entre as três implementações, Tendo como referência a implementação sem grupos. Podemos verificar que existe uma melhoria de 10 vezes analisando a abordagem sequencial com grupos, enquanto que nossa abordagem onde paralelizamos os grupos têm melhoria média de 67 vezes.



A abordagem apresentada neste trabalho possui um *speed up* médio quase linear, quando o ganho de performance é quase igual ao acréscimo de unidades de processamento. Por exemplo, no *benchmark* adaptec2 podemos perceber um acréscimo de 11,6 vezes para um acréscimo de 12 vezes no processamento. Desta forma, a utilização de paralelismo de forma eficiente, aliada à estratégia de dividir em grupos as redes, tendo como base a estimativa de comprimento de fio, tem se mostrado bastante eficiente.

5.5 Discussão dos Resultados

Neste capítulo foram apresentados os resultados a respeito das análises realizadas no roteador global existente no grupo de pesquisa. Também foi apresentado dados detalhados a respeito dos arquivos de *benchmark*, de forma a mostrar a real quantidade de redes de menor

porte com potencial de paralelismo. Paralelismo que foi incorporado ao analisador de dependência de dados e serviu para apresentar informações detalhadas a respeito do tempo de processamento necessário para a descoberta de paralelismo, ou seja, o *overhead* da proposta de paralelismo.

Neste trabalho não foi possível implementar completamente o colisor de redes devido à dificuldade de programação do mesmo. Pretende-se finalizar esta etapa de programação como um trabalho futuro.

A união destes resultados mostram que a técnica de paralelismo apresentada é uma boa solução e que tem potencial para reduzir o tempo de processamento do roteamento global de forma expressiva.

6 CONCLUSÕES E TRABALHOS FUTUROS

Percebe-se que a grande maioria dos trabalhos atuais em roteamento global de circuitos integrados apontam para o uso de paralelismo e mais ainda, a sua exploração através da técnica de abordagem de roteamento independente de redes, variações da mesma ou inclusive implementações híbridas.

Isto se deve à crescente complexidade dos circuitos atrelada ao aumento não linear da necessidade de recursos computacionais. Isto impulsiona para uma utilização de abordagens mais eficientes que reduzam o tempo de processamento do roteamento global. Sendo que, uma destas formas é a utilização de computação paralela, seja através de máquinas paralelas com memória compartilhada, seja através de arquiteturas massivamente paralelas como as GPUs.

Desta forma, este trabalho, além de mostrar o panorama do processo computacional, em torno do roteamento global abordou com análises dos arquivos de *benchmarks* dos ISPDs de 2007 e 2008 suas características e os resultados mostram uma predominância de redes menores nos circuitos. Isto sugere um potencial de paralelização da tarefa de roteamento global através da estratégia de roteamento independente de redes.

Neste trabalho, foi proposto o agrupamento de redes de acordo com o seu comprimento de fio. Foram abordadas as vantagens deste pré-roteamento, avaliando o *overhead* existente e comparando o tempo necessário para detectar as dependências de dados (colisões) em três casos: sem agrupamentos de redes, com agrupamentos e processamento sequencial e com agrupamentos e processamento paralelo. Nossa abordagem de agrupamento de redes, combinada com a utilização de computação paralela de memória compartilhada, reduziu em aproximadamente 67 vezes o tempo da etapa de detecção de colisões quando comparada com a implementação sequencial e sem agrupamentos de dados.

Desta forma, pode-se concluir que o método apresentado tem potencial de reduzir o tempo para a execução do roteamento global considerando a sua implementação em hardwares altamente paralelos como, por exemplo, as GPUs.

Trabalhos futuros envolvem a implementação total da técnica de processamento das redes em um roteador com o objetivo de revelar o *overhead* total do processo comparado com a melhoria de desempenho, comparada com a versão sequencial, de forma a mostrar a dimensão do grau de paralelismo existente e da escalabilidade do sistema. Além disso, uma implementação baseada em grafos para realizar a detecção de dependência de dados também

pode ser considerada, devido à possibilidade real de melhorar significativamente a performance.

Ainda assim, outras perguntas ainda estão em aberto, como qual o real ganho de desempenho que o roteamento global pode ter, caso implementado em arquiteturas massivamente paralelas ou em hardware programável. Também pretende-se elucidar se em implementações fora da CPU qual a importância do barramento de comunicação, quais abordagens de arquitetura se sobressaem de forma mais eficiente perante a técnica de paralelismo do roteamento global.

REFERÊNCIAS

AKL, Selim G.. **Parallel computation: models and methods**. Upper Saddle River: Prentice Hall, 1997. 608 p.

ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., 1989.

AMD**, Advanced Micro Devices**, Disponível em: http://www.amd.com. Acesso em: 29 maio 2013.

AMDAHL, G. M. **Validity of the single processor approach to achieving large scale computing capabilities**Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring). **Anais**...New York, New York, USA: ACM Press, 1967.

BORKAR, S.; Thousand core chips: a technology perspective. In PROCEEDINGS OF THE 44TH ANNUAL DESIGN AUTOMATION. 2007...**Proceedings**. New York, ACM Press, 2007. 4 p.

CROIX, J. F.; KHATRI, S. P. **Introduction to GPU programming for EDA**Proceedings of the 2009 International Conference on Computer-Aided Design - ICCAD '09. **Anais**...New York, New York, USA: ACM Press, 2009.

GAO, J.; WANG, T. **A new global router for modern designs**2008 Asia and South Pacific Design Automation Conference. **Anais**...IEEE, jan. 2008.

GASTER, Benedict et al. **Heterogeneous computing with OpenCL**. Amsterdam: Morgan Kaufmann, 2012. 277 p.

GEREZ, Sabih H. **Algorithms for vlsi design automation**. Chichester: John Wiley, 1999. 326 p.

FLYNN, M. J. Some Computer Organizations and Their Effectiveness. **IEEE Transactions on Computers**, v. C-21, n. 9, p. 948–960, set. 1972.

HAN, Y. et al. Exploring High-Throughput Computing Paradigm for Global Routing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems,** v. 22, n. 1, p. 155–167, jan. 2014.

HAN, Y.; CHAKRABORTY, K.; ROY, S.; A global router on GPU architecture. In 2013 IEEE 31ST INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD). 2013...**Proceedings**. Asheville NC, IEEE. 2013. 6 p.

HANAN, M. On Steiner's Problem with Rectilinear Distance. **SIAM Journal on Applied Mathematics**, v. 14, n. 2, p. 255-265, mar. 1966.

HU, Jin; ROY, Jarrod a.; MARKOV, Igor L. Completing high-quality global routes. In 19TH INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN - ISPD '10. 2010...**Proceedings**. New York, ACM Press. 2010. 7 p.

HWU, W. **GPU computing gems**. Amsterdam: Morgan Kaufmann, 2012. 541 p.

INTEL, **Intel Xeon Phi Product Family**, Disponível em: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html. Acesso em: 29 maio 2013.

JOHANN, M.; REIS, R. Net by net routing with a new path search algorithm. In 13TH SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN. 2000...**Anais**. Manaus, IEEE, 2000, 5 p.

JOHANN, M. **Novos algoritmos para roteamento de circuitos VLSI**. 2001. 158p. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

KHRONOS GROUP, **OpenCL: The Open Standard for Parallel Programming of Heterogeneous Systems**, Disponível em: http://www.khronos.org/opencl. Acesso em: 29 maio 2013.

LIU, W. et al. 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.

MADDEN, P. H. Dispelling the Myths of Parallel Computing. **IEEE Design & Test of Computers**, n. February, p. 1–1, 2012.

MCMURCHIE, L.; EBELING, C. PathFinder. In PROCEEDINGS OF THE 1995 ACM THIRD INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE GATE ARRAYS - FPGA '95. 1995...**Proceedings**. New York, ACM Press, 1995. 6 p.

MOFFITT, M. D. MaizeRouter: Engineering an Effective Global Router. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 11, p. 2017-2026, nov. 2008.

MOFFITT, M. D.; ROY, J. A.; MARKOV, I. L. The Coming of Age of (Academic) Global Routing. In ISPD'08 PROCEEDINGS OF THE 2008 INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN. 2008...**Proceedings**. New York, ACM Press. 2008. 7 p.

NAM, G.-J. et al. ISPD placement contest updates and ISPD 2007 global routing contest. In 2007 INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN – ISPD '07. 2007...**Proceedings**. New York, ACM Press, 2007. 1 p.

NAM, G.; SZE, C.; YILDIZ, M. The ISPD global routing benchmark suite. In 2008 INTERNATIONAL SYMPOSIUM ON PHYSICAL DESIGN - ISPD '08. 2008...**Proceedings**. New York, ACM Press, 2008. 4 p.

NVIDIA, **CUDA Parallel Computing Platform**, Disponível em: http://www.nvidia.com/object/cuda_home_new.html. Acesso em: 29 maio 2013b.

NVIDIA, Disponível em: http://www.nvidia.com. Acesso em: 29 maio 2013a.

PAN, D. Z. BoxRouter 2.0: architecture and implementation of a hybrid and robust global router. In 2007 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN. 2007...**Proceedings**. San Jose, CA, IEEE, 2007. 6 p.

PAN, D. Z. BoxRouter: a new global router based on box expansion and progressive ILP. In 2006 43RD ACM/IEEE DESIGN AUTOMATION CONFERENCE. 2006...**Proceedings**. [S.l.]. IEEE, 2006. 6 p.

PAN, M.; CHU, C. FastRoute 2.0: A High-quality and Efficient Global Router. **2007 Asia and South Pacific Design Automation Conference**, p. 250–255, jan. 2007.

PAN, M.; CHU, C. **FastRoute: A Step to Integrate Global Routing into Placement.** 2006 IEEE/ACM International Conference on Computer Aided Design. **Anais**...IEEE, nov. 2006

PATEL, A. **Amit's A* Pages**, Disponível em: http://theory.stanford.edu/~amitp/GameProgramming/index.html. Acesso em: 29 maio 2013.

QI, Z. et al. **VFGR: A very fast parallel global router with accurate congestion modeling** 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC). Anais...IEEE, jan. 2014

REIMANN, T. **Roteamento Global de Circuitos VLSI**. 2012. 111p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

ROY, J. A.; MARKOV, I. L. High-Performance Routing at the Nanometer Scale. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 6, p. 1066-1077, jun. 2008.

SHERWANI, N. A. **Algorithms for VLSI physical design automation**. [S.l.]: Kluwer Academic Publishers, 1993.

SHINTANI, Y. et al. **A Multithreaded Parallel Global Routing Method with Overlapped Routing Regions** 2013 Euromicro Conference on Digital System Design. Anais...IEEE, set. 2013

SHOJAEI, H.; DAVOODI, A.; BASTEN, T. Collaborative Multiobjective Global Routing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 21, n. 7, p. 1308–1321, jul. 2013.

SZYMANSKI, T. Dogleg Channel Routing is NP-Complete. **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems**, [S.l.], v.4, n.1, p.31 – 41, 1985.

TOP500 Supercomputer Sites, **Top500 list – November 2014**, Disponível em: http://www.top500.org/list/2014/11. Acesso em: 12 dezembro 2014.

WU, T.-H.; DAVOODI, A.; LINDEROTH, J. T. **GRIP: scalable 3D global routing using integer programming.** Proceedings of the 46th Annual Design Automation Conference - DAC ’09. Anais...New York, New York, USA: ACM Press, 2009.

WU, T.-H.; DAVOODI, A.; LINDEROTH, J. T. **A parallel integer programming approach to global routing.** Proceedings of the 47th Design Automation Conference on - DAC ’10. Anais...New York, New York, USA: ACM Press, 2010.

XU, Xueqiao, **PathFinding.js A comprehensive path-finding library in javascript. 2012.** Disponível em: https://github.com/qiao/PathFinding.js. Acesso em: 29 maio 2013.

XU, Y.; ZHANG, Y.; CHU, C. FastRoute 4.0: Global router with efficient via minimization. **2009 Asia and South Pacific Design Automation Conference**, p. 576–581, jan. 2009.

ANEXO A – ARTIGOS PUBLICADOS

Foram publicados três artigos relacionados ao tema da dissertação, apresentados neste anexo. Inicialmente é apresentado o artigo intitulado *Overhead for Independent Net Approach for Global Routing*, foi apresentado e publicado nos anais do VI Latin American Symposium on Circuits & Systems – LASCAS 2015, em fevereiro de 2015, ocorrido em Montevideu – Uruguai.

O segundo artigo, intitulado *Paralelismo no Roteamento Global de Circuitos VLSI: Estado da Arte*, publicado nos anais do XXI Iberchip Workshop – IWS'2015, em fevereiro de 2015, ocorrido em Montevideu – Uruguai.

Por fim, o artigo relacionado a dissertação, intitulado *Global Routing and Parallelism* apresentado no 28º Simpósio Sul de Microeletrônica, em abril de 2013, ocorrido em Porto Alegre.

Overhead for Independent Net Approach for Global Routing

Diego Tumelero, Guilherme Bontorin, Ricardo Reis

Instituto de Informática, PGMICRO/PPGC - Universidade Federal do Rio Grande do Sul

Porto Alegre, Brasil

{dtumelero, gbontorin, reis}@inf.ufrgs.br

***Abstract*—Global Routing is one of the major Electronic Design Automation steps and it is classified as an NP-hard problem. We verified that 61% of the nets in ISPD 2008's benchmarks are shorter than 128 length units. We propose a method to cluster these nets using an independent net approach to perform global routing in massively parallel systems. We demonstrate that separating nets in clusters according to its length and processing it in parallel can reduce by 67 the processing time for collision detection, if compared with a sequential non-clustered analysis.**

***Keywords*—Global Routing; Parallelism; Net Level.**

I. Introduction

Global Routing (GR) is part of the design of integrated circuits and consists of interconnecting the elements of these circuits. This is one of the most time demanding steps of Electronic Design Automation (EDA), mainly because it is a combinatorial optimization problem classified as NP-hard [1]. Despite being one of the first automated areas in Electronic Design Automation, VLSI (Very Large Scale Integration) circuit routing remains a significant research and development domain [2-3].

An emerging research topic is the parallelization of GR, because state-of-the-art circuits have added a massive amount of transistors, e.g. the IBM Power8 series can be as large as 4.2 billions of transistors [4]. Therefore, it is necessary to optimize the process of GR on two manners: (a) in terms of the algorithm, using more efficient heuristics, and (b) in terms of CPU processing, making better use of available hardware resources to accomplish the task.

In this paper we focus on (b). We discuss the overhead of independently processing each net in GR and aim to streamline the task without interfering directly with routing algorithms in order to make the process more modular. In literature, [5] presents a similar approach, but we are the first to show the predominant size of the nets and to detect the data dependency in a reasonable time.

The rest of the paper is organized as follows. Section II gives a brief statement of related work. Section III presents the details of the independent net level approach. Section IV explains the methodology, followed by the results in section V. Section VI concludes the paper.

II. Global Routing and Parallelism

A. Global Routing

The GR step is typically performed before the detailed routing and after clock tree synthesis (CTS) in the physical synthesis flow of digital integrated circuits. The aim of global routing is to establish areas where the connections specified in the netlist file will be made, respecting the constraints, such as congestion, IP cores, antenna gate, heating, time, and others. Figure 1 shows a typical design flow.

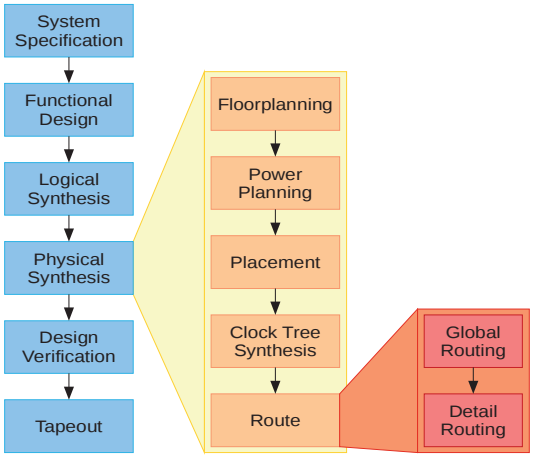


Figure 1 – Typical IC project flow. - Based on [6].

The mathematical characterization of the global routing is based on graph theory, where a circuit is represented by a grid G that specifies two groups of components: a set of vertices V and edges E . Each $v_i \in V$ sets a region (tile). Each $e_{ij} \in E$ corresponds to a boundary between adjacent areas [7]. These frontiers have a limit of available resources m_{ij} that may be used to quantify the connections congestion.

There is at least one group of nets N, and each $n_i \in N$ is formed by a set P_i of pins. Each one of these pins corresponds to a vertex v_i .

The solution is found when all the connections are routed,

978-1-4799-8332-2/15/$31.00 ©2015 IEEE

respecting the limits of each edge with the lowest processing time as possible, still satisfying constraints such as wire length (WL), number of vias, manufacturability, heating, among others.

Notable advances have been recently achieved in routing algorithms and EDA tools. Some routers have gained notoriety in the academic sphere because of the performance demonstrated in recent contests [2-3] (Table I).

TABLE I – MAJOR ACADEMIC GR – BASED ON [8]

	FastRoute	FastRoute 2.0	FastRoute 4.0	BoxRouter	BoxRouter 2.0	FGR	MaizeRouter	NTHU-Route 2.0
Pattern routing	■	■	■	■	■		■	■
Monotonic routing		■	■					■
Maze routing	■	■	■	■	■	■	■	■
A* search					■	■	■	
FLUTE dependence	■	■	■	■	■		■	■
Topology reconstruction	■	■	■		■	■	■	■
Incremental						■	■	
Edge “sliding”		■	■				■	
Resource sharing	■	■	■			■	■	
ILP or MCF				■	■			
Congestion manipulation	■	■	■		■	■	■	■
History-based						■	■	■
Layer Assignment			■			■	■	■
Open source				■	■	■	■	

B. Parallelism in GR

Parallel computing is a way of computing where execution of computations occurs simultaneously [9].

From the state of art about global routing using parallel approaches, two principal methods can be distinguished: (a) partitioning circuit and (b) independent routing nets.

a) The partitioning of the circuit [10] is the form encountered to split the routing task, and the problem is literally divided into parts which are processed in parallel. The challenge lies in finding out where to split the circuit, and after that, understanding how to resolve potential conflicts between the responses of the router for each partition.

b) The independent routing of nets [5][11-13] is an approach where nets sufficiently distant from each other are processed in parallel. For this purpose, there are mechanisms responsible to detect and work properly with the data dependencies between nets, as will be explained better in section III.

Both methods intrinsically have sequential processing portions of work. Also, there is a portion of rework, resulting from data dependencies, for having nets crossing partitions or a net extrapolating it's boundary-box. Table II reviews the main references in parallelism and independent routing of nets.

TABLE 2 – PARALLEL APPROACH FOR GLOBAL ROUTERS.

Reference	Router Name	Parallel Method
[10]	PGRIP	Partition Level
[14]	VFGR	Net And Partition Levels
[5]	-	Net Level
[11]	-	Net Level
[12]	NCTU-GR 2.0	Net Level
[13]	-	Net Level
[15]	CGR	Pareto Algebra Instances

In this paper, we focus on the Independent Net Routing. Next section presents the details of our pre-routing technique for parallelism in global routing.

III. INDEPENDENT NET APPROACH

A clear advantage of independent routing of nets is that this method provides a better utilization of computational resources. Indeed, as it generates a much larger number of threads, it can better use multi-core architectures.

To avoid problems of race condition, when processing nets with data dependency that dispute areas too close one to another in the circuit, some measures are taken:

a) To perform a pre-processing step to identify data dependency for processing only nets distant to one another.

b) To cluster the netlist, classifying the nets according to WL in groups, in order to separate short and long nets without performing a total sorting.

c) Processing first the lower WL nets instead of major ones [16] in each processing step. This is important, because short nets tend to use lower metal layers, and, consequently, release the higher layers for global nets [14].

Although other works [5][7][16] propose to use this pre-ordering and clustering, we are the first to estimate numerically the advantage of such a method and to quantify the computing time overhead to implement it.

IV. METHODOLOGY

Our methodology has three steps: A. Data Analysis, B. Development of the collider, and C. the Overhead itself. In this section, we present the methods to measure these three elements, and, in the next one, we show the results and discuss its implications.

A. Data Analysis

In order to estimate the coverage of our hypothesis, we analyze the benchmarks from ISPD 2008 [3]. “These benchmarks are all derived from real industrial circuits and present modern physical design challenges such as scalability, variety of floorplans, movable macro handling, and congestion mitigation.” [3]

We have built a piece of software that reads each benchmark. And enumerates the nets in it. For each net, we create a rectangular boundary box, containing all the pins associated to it. Then, we cluster the nets in groups with the same range of length. More precisely, each group contains wires whose boundary box's semiperimeter is in between two consecutives powers of 2 (2^n) lengths.

This is the first step of our pre-global-routing. In this data analysis, we run all ISPD 2008's benchmarks to see the percentage of each cluster.

B. Collider

We write a "collider" of nets to detect which ones have data dependency, with the challenge of testing the highest number of nets in the shortest time, considering a single cluster of nets.

The algorithm includes three different implementations of the collider that were performed, aiming to discover the overhead of the procedure:

a) No cluster, analyzing all the nets without separation into clusters;
b) Cluster with sequential processing, dividing the nets into clusters to reduce the search space;
c) Cluster with parallel processing, dividing the nets into clusters and using OpenMP technology to process them in parallel.

In the next section, we quantify the time overhead for each one.

C. Overhead

Finally, the main contribution of our paper is to measure the runtime of the whole process to study the viability of this clustering.

The tests were performed on a Sandy Bridge-E 12-core (6 physical cores) processor with 64GB RAM.

V. Results

A. Data Analysis

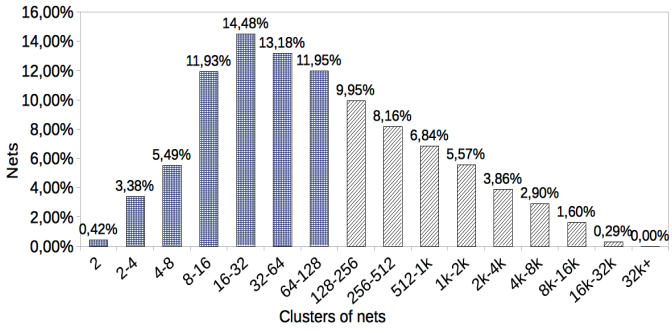


Figure 2 – Proportion of nets on each cluster from all benchmarks of ISPD 2008 [2]. The clusters are divided by semiperimeter of boundary boxes size. Nets shorter than 128 length units represents 61% of all nets (left and blue bars).

Figure 2 shows the results of our data analysis. The most predominant group has a semiperimeter between 16-32. The fact that, 61% of the nets are shorter than 128 units of length. This suggests a great potential for our intention of clustering the pre-routing.

B. Collider

Figure 3 shows an example of how our collider detector works for a simple circuit with 10 boundary boxes inside a same cluster. The biggest group of nets without collision is composed of the nets 1, 2, 5, 6, 8 and 9. These 6 nets can be divided into 6 simultaneous threads. The remaining nets can be grouped in two other threads, one with the nets 3 and 4, and another group with nets 7 and 10. In this example, the routing of 10 nets inside a same-semiperimeter cluster can be done in three groups of parallel processing.

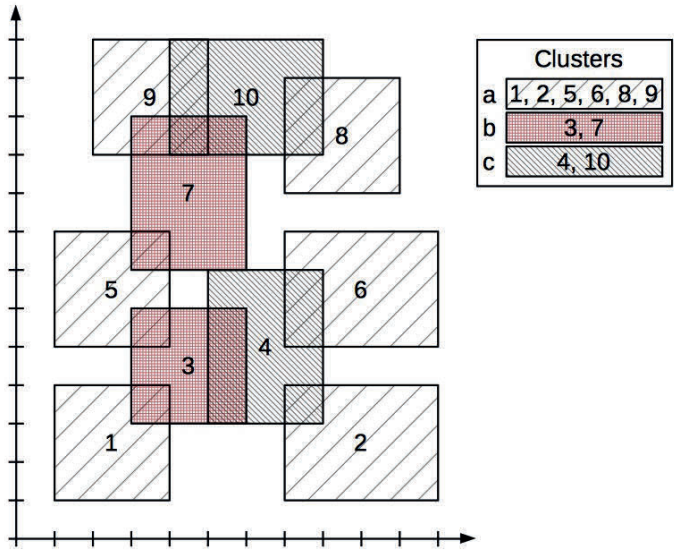


Figure 3 – Example of identification of data dependency between boundary boxes of nets.

Most of the dependency-free nets appears on the first cycle of verification. In the following iterations, the group's size tends to decrease linearly and the processing time decreases geometrically.

C. Overhead

Figure 4 shows the execution time for each one of the implementations of the collider detector: No clustering, Cluster with sequential processing, and Cluster with parallel processing. Just from the processing of data dependency (or collision detection) data suggests that clustering the data through wire length (or boundary box's semiperimeter) can improve the further routing, and, more than this, the using of the processor capacity on massively parallel (multicore, multiprocessor, multi thread) systems.

Figure 5 shows the relative gain in time performance among the 3 implementations. The fact of clustering the data reduces the processing time by about 10 times. Parallelizing the process of the clusters reduces the overhead and has an

978-1-4799-8332-2/15/$31.00 ©2015 IEEE

average speedup of 67. For the adaptec2 prerouting is 100 times faster.

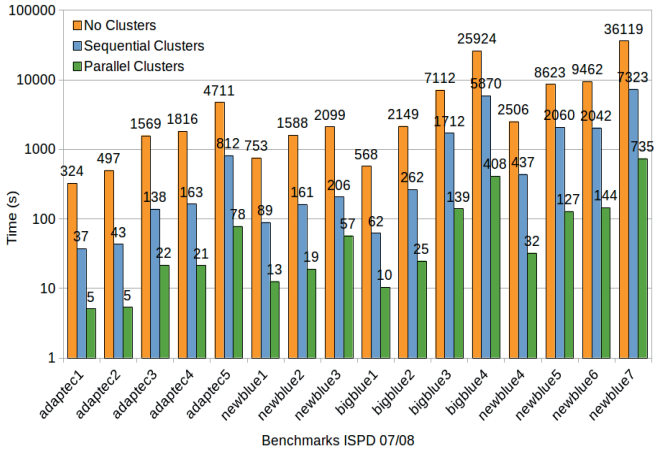


Figure 4 – Time overhead in absolute values for data dependency detection. There are three implementations of the net clustering: A. No clusters, B. Sequential Clusters, and C. Parallel Clusters. We can see that there is a significant gain when we parallelize and reduce the search space.

Our approach has an almost linear speedup for most benchmarks, the gain in performance is closer to the increase on the number of processors.

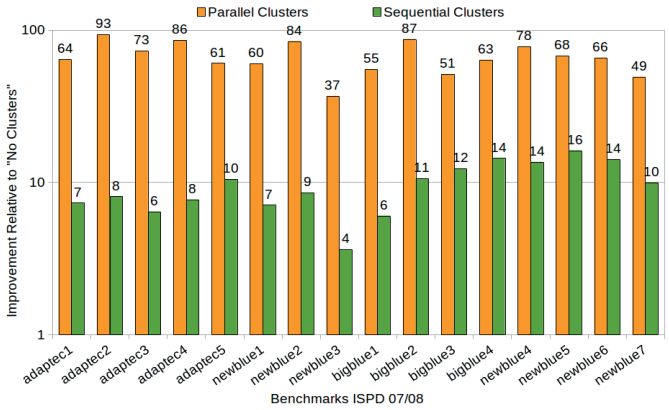


Figure 5 – Relative values of improvement on time overhead from Figure 4, taking as reference the No Clusters implementation. We see that we have a reduction in 10 times comparing to Sequential Clusters and, even better, our approach with Parallel Clusters is 67 times faster.

VI. Conclusion

Our data analysis from ISPD 2008's benchmark circuits shows the predominance of short nets in the circuits. It suggests a potential to parallelize the process of global routing.

In this paper, we propose to cluster the nets according to their lengths. We investigate the benefits of a pre-routing method by measuring its total overhead and comparing the time for collision detection in three cases: no clustering, clustering with sequential processing and clustering with parallel processing. Our clustering, combined with the parallel execution, reduces in about 67 times the collision detection, when compared to sequential processing of no clustered data.

Future work involves the incorporation of our prerouting in several academic routers for further comparison.

Acknowledgment

This work is founded by Brazilian agencies Capes (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

References

[1] T.G. Szymanski, "Dogleg Channel Routing is NP-Complete," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 4, no. 1, pp. 31-41, 1985.

[2] G.-J. Nam, M. Yildiz, D. Z. Pan, and P. H. Madden, "ISPD placement contest updates and ISPD 2007 global routing contest," in *Proceedings of the 2007 international symposium on Physical design - ISPD '07*, 2007, p. 167.

[3] G. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proceedings of the 2008 international symposium on Physical design - ISPD '08*, 2008, p. 156.

[4] E. J. Fluhr, J. Friedrich, D. Dreps, V. Zyuban, G. Still, C. Gonzalez, A. Hall, D. Hogenmiller, F. Malgioglio, R. Nett, J. Paredes, J. Pille, D. Plass, R. Puri, P. Restle, D. Shan, K. Stawiasz, Z. T. Deniz, D. Wendel, and M. Ziegler, "5.1 POWER8TM: A 12-core server-class processor in 22nm SOI with 7.6Tb/s off-chip bandwidth," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 96–97.

[5] Y. Han, D. M. Ancajas, K. Chakraborty, and S. Roy, "Exploring high throughput computing paradigm for global routing," *2011 IEEE/ACM Int. Conf. Comput. Des.*, pp. 298–305, Nov. 2011.

[6] J. Banker, A. Shanbhag, and N. Sherwani, "Physical design tradeoffs for ASIC technologies," in *Sixth Annual IEEE International ASIC Conference and Exhibit*, 1993, pp. 70–78.

[7] M. D. Moffitt, J. A. Roy, and I. L. Markov, "The Coming of Age of (Academic) Global Routing [Invited Paper]," in *ISPD'08 Proceedings of the 2008 international symposium on Physical design Pages 148-155*, 2008.

[8] Reimann, Tiago. Roteamento Global de Circuitos VLSI. Master Thesis – UFRGS 2012.

[9] G. S. Almasi and A. Gottlieb. 1989. Highly Parallel Computing. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA, USA.

[10] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A parallel integer programming approach to global routing," in *Proceedings of the 47th Design Automation Conference on - DAC '10*, 2010, p. 6.

[11] Y. Han, K. Chakraborty, and S. Roy, "A global router on GPU architecture," in 2013 IEEE 31st International Conference on Computer Design (ICCD), 2013, pp. 78–84.

[12] W. Liu, W. Kao, Y. Li, and K. Chao, "NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing With Bounded-Length Maze Routing," IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 32, no. 5, pp. 709–722, May 2013.

[13] Y. Shintani, M. Inagi, S. Nagayama, and S. Wakabayashi, "A Multithreaded Parallel Global Routing Method with Overlapped Routing Regions," in 2013 Euromicro Conference on Digital System Design, 2013, pp. 591–597.

[14] Z. Qi, Y. Cai, Q. Zhou, Z. Li, and M. Chen, "VFGR: A very fast parallel global router with accurate congestion modeling," in 2014 *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 525–530.

[15] H. Shojaei, A. Davoodi, and T. Basten, "Collaborative Multiobjective Global Routing," IEEE Trans. Very Large Scale Integr. Syst., vol. 21, no. 7, pp. 1308–1321, Jul. 2013.

[16] Johann, Marcelo de Oliveira. Novos algoritmos para roteamento de circuitos VLSI. PhD Thesis – UFRGS. 2001

Paralelismo no Roteamento Global de Circuitos VLSI: Estado da Arte

Diego Tumelero, Vitor V. Bandeira, Guilherme Bontorin e Ricardo A. L. Reis
PGMicro/PPGC – Instituto de Informática – Universidade Federal do Rio Grande do Sul
{dtumelero, vvbandeira, gbontorin, reis}@inf.ufrgs.br

***Resumo*—O roteamento global de circuitos VLSI (*Very Large System Integration*) é uma das tarefas mais complexas do fluxo de síntese física. Esta tarefa é demasiadamente demorada nas ferramentas atuais, isto se deve a complexidade do problema e as implementações que não exploram a capacidade total de computação das máquinas atuais. Uma das formas de agregar grande poder de computação a um baixo custo é a utilização de GPUs (*Graphical Processing Units*) para o processamento geral. Neste trabalho é apresentada uma revisão sobre o estado da arte nesta área e as propostas para a paralelização desta etapa do projeto de circuitos integrados.**

Palavras-chaves* — *Roteamento Global; Paralelismo; GPU (Graphical Processing Unit); VLSI (Very Large System Integration).

I. Introdução

O processo de projeto de circuitos integrados inclui em um de seus passos a tarefa de roteamento global, ou seja, interconectar os elementos do circuito. Apesar de ser uma das primeiras áreas de *Eletronic Automation Design* (EDA) que foi automatizada, o roteamento de circuitos VLSI (*Very Large Scale Integration*) permanece como uma área com significativa atividade de pesquisa e desenvolvimento [1]. Em parte, isto se deve que a tarefa de roteamento é considerada um problema NP-completo [2]. Este passo é o que mais demanda tempo de processamento e, por muitas vezes, o poder da computação em paralelo não é utilizado.

No capítulo II é feita uma revisão a respeito do roteamento global. O capítulo III repassa alguns conceitos da teoria dos grafos e descreve brevemente alguns mecanismos de pesquisa. No capítulo IV fala-se a respeito do paralelismo e das particularidades das GPUs (*Graphical Processing Units*). Capítulo V apresenta os principais roteadores acadêmicos e as alternativas de paralelismo para o Roteamento Global. Por fim, o capítulo VI concluí o trabalho e apresenta as perspectivas de desenvolvimento nesta área de pesquisa.

II. Roteamento Global

O roteamento global faz parte da síntese física e é realizado antes do roteamento detalhado — que define por quais camadas de metal os fios passarão e onde exatamente cada fio passará no *chip* — e, tipicamente, após a síntese da árvore de relógio (ou CTS) na síntese física. O processo tem como entrada o resultado final do posicionamento, uma lista de conexões a serem realizadas com locais fixos de blocos e pinos. A figura 1 mostra o fluxo de projeto.

Após o posicionamento, o circuito é particionado em regiões denominadas *tiles* e decide o caminho entre estas partições para todas as conexões. O objetivo do roteamento global

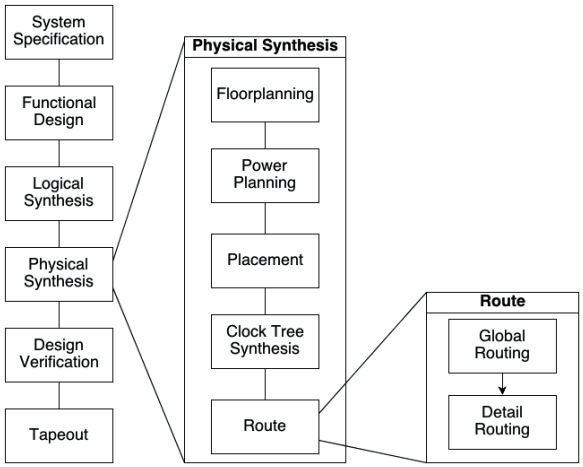


Figura 1. Fluxo de projeto típico de circuitos integrados [3]. O roteamento global é uma etapa da síntese física, entre o CTS e o roteamento detalhado.

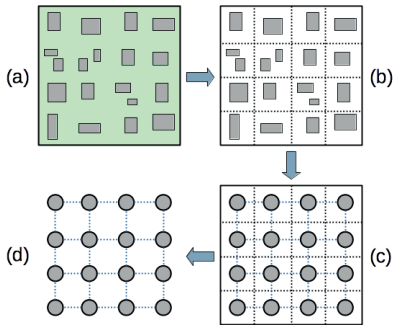


Figura 2. Etapas do roteamento global: (a) resultado do posicionamento. (b) particionamento do circuito. (c) representação das bordas. (d) grafo do roteamento global.

é determinar regiões para onde as conexões especificadas na *netlist* serão feitas respeitando as restrições impostas. Estas conexões interligam células padrão que contém elementos de lógica. O roteamento global não considera as regras de projeto de cada tecnologia, e após finalizado o mesmo pode ainda admitir algum congestionamento.

A solução é encontrada quando todas as conexões são roteadas, respeitando os limites de cada aresta com o menor tempo de processamento possível e satisfazendo restrições como o comprimento de fio, menor número de vias, fabricabilidade, temperatura, entre outros. A figura 2 mostra uma visão geral do roteamento global. Por fim, a caracterização matemática do roteamento global é baseada na teoria dos grafos que é analisada no próximo capítulo.

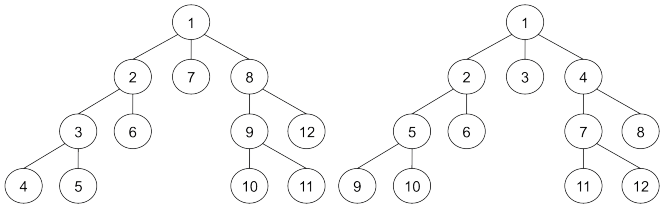


Figura 3. Comparação dos métodos de pesquisas: (a) DFS e (b) BFS.

III. PESQUISA EM GRAFOS

Os grafos são caracterizados por dois conjuntos, o conjunto dos vértices V e o conjunto das arestas E que denotam um grafo $G(V, E)$. Um circuito pode ser representado por *G* onde cada $v_{\text{i}} \in V$ indica uma região (célula ou *tile*) e cada $e_{\text{ij}} \in E$ corresponde a fronteira entre regiões adjacentes [4]. Estas bordas tem um limite de recursos disponíveis m_{ij} que pode ser utilizado para quantizar o congestionamento das conexões. Existe ao menos um conjunto de *nets N* , e cada $n_i \in N$ é composto de um conjunto P_i de pinos. Cada um destes pinos corresponde a um vértice v_i . Dois vértices que formam uma aresta podem ser chamados de *endpoints*. A aresta pode ser identificada por dois *endpoints* $u, v \in V$ seguindo a notação (u, v), por exemplo, $e_3 = (v_1, v_3)$.

O uso de grafos pode facilitar a formulação matemática, a análise e solução do problema. Logo, eles podem ser usados para simplificar o roteamento transformando-o em uma pesquisa de *nets*. A seguir estão listados os principais métodos de busca:

1) DFS (Depth-first Search): A busca em profundidade explora a partir da raiz, todos os nodos de um ramo até as suas folhas, para então retroceder até a raiz, figura 3a. Ou seja, o objetivo é visitar todos os vértices apenas uma vez, aonde por sua vês os mesmos são marcados como visitados. A complexidade temporal deste método é de $O(|E|)$ (sendo E o número de arestas), ou seja, o mesmo pode percorrer todo o grafo na pior hipótese. Já a complexidade espacial é de $O(|V|)$ (sendo V o número de vértices) o que na prática significa que a profundidade da busca é limitada para não esgotar os recursos computacionais [5].

2) BFS (Breadth-first Search): Diferentemente da busca em profundidade, a busca em largura realiza uma pesquisa sistemática (figura 3b) no grafo, onde a complexidade temporal é $O(n + |E|)$, isto ocorre devido a pesquisa aprofundar os ramos do grafo de forma igualitária, visitando todos os nodos vizinhos do mesmo nível antes de seguir para os nodos filhos [5]. Já a complexidade espacial é $O(|V|)$ onde $|V|$ é a cardinalidade do conjunto de vértices.

3) Dijkstra: O método de Dijkstra resolve o problema do caminho mais curto para grafos de arestas de peso não negativo. O algoritmo é do tipo guloso, ou seja, utiliza a decisão que parece ótima no momento. Ele verifica todas as arestas conectadas ao nodo inicial (S) e escolhe a de menor custo para ser o novo nodo inicial (S) e continua recursivamente até que o destino seja encontrado (mas ignora as arestas que ligam vértices já pertencentes a S). A complexidade deste método é $O(n^2 + |E|)$, que pode ser simplificada para $O(n^2)$ [5].

Tabela I. COMPARATIVOS ENTRE OS PRINCIPAIS MÉTODOS DE PESQUISA.

	Implementação	Velocidade	Complexidade
DFS	Simples	Lento	$O(\|E\|)$
BFS	Simples	Lento	$O(n + \|E\|)$
Dijkstra	Simples	Lento	$O(n^2 + \|E\|)$
$A*$(A Star)	Complexa	Rápido	$O(log\ h * (x))$

4) A (A Star):* O algoritmo A* (*A Star*) é uma evolução do método de Dijkstra e do *Best-First-Search*, onde as melhores características de ambos, encontra o menor caminho se o mesmo existir e pode utilizar heurística como no guloso. O método é baseado na seguinte equação:

$$f_{(n)} = g_{(n)} + h_{(n)} \tag{1}$$

Sendo $g_{(n)}$ o custo exato do ponto de partida até um nodo n qualquer e $h_{(n)}$ o custo estimado de um nodo n qualquer para o destino. Esta estimativa pode ser uma heurística, caminho *manhattan* ou outros métodos que estimem a distancia $h_{(n)}$ [6].

Os métodos apresentados neste capítulo são os principais dentre os que tem garantia de encontrar um caminho caso exista um caminho. Conforme a Tabela I, o *A Star* tem um desempenho bom com resultados próximo ao ótimo.

Uma vez que o reteamento é traduzido em grafos e foi determinado o mecanismo de pesquisa, podemos abordar a implementação em máquinas *multicore* através de paralelismo.

IV. PARALELISMO

Computação paralela é uma forma de computação onde a execução de operações ocorre simultaneamente [7].

Uma característica da computação paralela é o conceito de *speedup*. Uma forma de comparar o desempenho de um algoritmo paralelo com sua implementação sequencial. Segundo a formulação de Amdahl [8], o ganho de desempenho de uma implementação, ou de speedup S, é dado pela equação:

$$S_{(P)} = \frac{T_{(1)}}{T_{(P)}} \tag{2}$$

Onde $T_{(1)}$ é o tempo de execução com um processador, P o número de processadores e $T_{(P)}$ é tempo de execução com P processadores. Para atingir um bom *speedup* é necessário que a porção sequencial seja a menor possível.

Gustafson [9], por sua vez afirmou que a limitação de *speedup* pela regra de Amdahl tem como exceção quando o conjunto de dados é grande, fixando no tempo a parcela sequencial em vez do tamanho, ou seja, resolver o maior problema possível em uma máquina de grande porte em um tempo semelhante ao de uma maquina menor com um problema menor.

Pollack [10], constatou empiricamente que a performance de um processador aumenta linearmente enquanto que sua complexidade aumenta quadraticamente conforme a equação:

$$performance = \sqrt{complexidade} \tag{3}$$

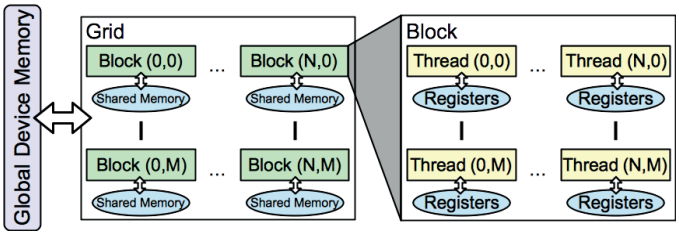


Figura 4. Hierarquia de memória em uma GPU.

Isto significa que processadores com múltiplos núcleos de processamento tem vantagem comparado a processadores single core de complexidade equivalente, fato que explica a grande capacidade de desempenho das GPUs.

Uma característica das GPUs é sua hierarquia de memória e das unidades de processamento, isto reduz a quantidade de memória utilizada, de forma que existem mais transistores realizando o processamento do que utilizados em memória cache. A figura 4 mostra a organização típica das GPUs. Este modelo infere em um sistema de *pipeline* muito mais profundo que o das CPUs (*Central Processing Units*), onde instruções condicionais são tratadas como *threads* separadas para evitar ao máximo o seu esvaziamento. Ou seja, a execução de um desvio condicional gera a criação de duas linhas de execução, quando é sabido o resultado uma destas linhas é morta.

Segundo a taxonomia de Flynn [11], pode ser dividida em quatro categorias: SISD (*Single Instruction, Single Data Stream*) onde uma instrução é aplicada a um fluxo de dados em cada momento de processamento. MISD (*Multiple Instruction, Single Data Stream*) onde diversas instruções atuam sobre o mesmo fluxo de dados. SIMD (*Single Instruction, Multiple Data Stream*) uma instrução é aplicada em diversos fluxos de dados simultaneamente. MIMD (*Multiple Instruction, Multiple Data Stream*) onde diversas instruções podem ser aplicadas em diversos fluxos de dados simultaneamente.

Existem tipicamente dois tipos de GPUs [12] com características de programação de propósito geral (GPGPU). As baseadas em SIMD como as placas da nVidia e da AMD e as baseadas em MIMD como a aceleradora Xeon Phi da Intel.

Dentre os casos de estudo na área de EDA, temos duas abordagens, a portabilidade e a re-arquitetura. No primeiro, caso temos exemplos como aceleração de SPICE, simulação de falhas e análise estatística de tempo estático baseado em Monte Carlo. No segundo caso, o problema de satisfazibilidade booleana, geração de tabela de falhas e *power grid analysis* [13].

No próximo capítulo veremos a aplicação do paralelismo no roteamento global.

V. Paralelismo no Roteamento Global

A. Principais Roteadores Acadêmicos

Notáveis avanços foram recentemente conquistados nos algoritmos de roteamento e ferramentas EDA. Alguns roteadores ganharam notoriedade na esfera acadêmica devido a performance demonstrada em competições recentes [1][14].

Tabela II. Características dos principais roteadores acadêmicos

	[15]	[16]	[17]	[19]	[20]	[21]	[22]	[23]
Pattern routing	•	•	•	•	•		•	•
Monotonic routing		•	•					•
Maze routing	•	•	•	•	•	•	•	•
A* search					•	•	•	
FLUTE dependence	•	•	•	•	•		•	•
Topology reconstruction	•	•	•		•	•	•	•
Incremental						•	•	
Edge "sliding"		•	•				•	
Resource sharing	•	•	•			•	•	
ILP or MCF				•	•			
Congestion manipulation	•	•	•		•	•	•	•
History-based						•	•	•
Layer Assignment			•			•	•	•
Open source				•	•	•	•	

1) FastRoute [15], FastRoute 2.0 [16] e FastRoute 4.0 [17]: utilizam um mapa de congestionamentos para modelar uma grade de Hanan [18] durante a geração de árvore Steiner, seguido pelo *edge shifting e pattern routing*. Em [16], o roteador é atualizado com o método de roteamento monotônico e por um *maze routing, multi-source* de roteamento. A versão mais recente aborda o problema de otimização do número de vias ao longo de todo o fluxo de roteamento global.

2) BoxRouter [19]: expande progressivamente um perímetro a partir da região mais congestionada do *chip*, aplicando programação linear inteira (ILP) na formulação e considerando os padrões em forma de L para refazer conexões entre perímetros sucessivos.

3) BoxRouter 2.0 [20]: é uma evolução que utiliza uma versão dinâmica do método de pesquisa A* e incorpora, adaptável à topologia *rip-up* para passar os fios das regiões congestionadas, sem alterar a topologia da rede.

4) Fairly Good Router (FGR) [21] : baseia-se no roteador *PathFinder* [6], originalmente desenvolvido para FPGAs. Ele usa uma função específica para penalizar o congestionamento e realiza uma rápida atribuição de camadas seguido por um 3D *clean-up*.

5) MaizeRouter [22]: utiliza duas operações elementares baseadas em borda (*extreme edge shifting* e *edge retraction*) e manipula sequencialmente os segmentos individuais das redes. Sua abordagem se baseia na decomposição interdependente das redes, em que as soluções de roteamento são implicitamente mantidas por conjuntos de intervalos em vez de topologias definidas.

6) NTHU-Route 2.0 [23]: é baseado em *rip-up* e *re-route*. Ele usa uma função de custo baseado em um histórico para distribuir o *overflow* e emprega um método de identificação para especificar a ordem para regiões *rip-up* congestionadas. A redução do comprimento de fio é conquistada através de um método *maze routing* adaptativo, *multi-sink, multi-source* de roteamento.

Na Tabela II temos uma relação entre os roteadores trazendo mais algumas características.

B. Paralelizando o Roteamento Global

A partir do estado da arte em roteamento global usando abordagens paralelas, podem ser distinguidos dois métodos:

1) Roteamento baseado em particionamento do circuito [24]: Consiste em dividir o circuito em regiões atribuindo uma *thread* para cada região. Este método gera um número menor de *threads* que o próximo. Destarte, temos que definir como delimitar as regiões. Após o roteamento parcial de cada região, resta o esforço computacional de interligar nets que pertençam a mais de uma região.

2) Roteamento independente de cada net [25]: Consiste em considerar cada *net* independentemente como uma *thread*. Este método permite realizar o roteamento global usando um número grande de *threads*, ou seja, permite utilizar melhor os recursos computacionais como GPUs. Para rotear de forma independente são escolhidas as conexões distantes o suficiente, para que ao longo das possíveis combinações de um caminho *manhattan* estas não se sobreponham. Como nem todas as redes são distantes, temos blocos sequenciais que não permitem paralelização massiva.

Ambos os métodos têm parcelas intrinsecamente sequenciais de processamento, seja ao conectar as fronteiras de partição, seja ao rotear *nets* vizinhas. Os métodos para tratar estas partes sequênciais resta um capítulo em aberto na pesquisa de algoritmos de paralelismo para o roteamento global.

VI. Conclusão

O roteamento global representa um dos maiores desafios computacionais na área de ferramentas EDA devido a sua complexidade computacional e o seu tempo de execução. Das formas de acelerar a tarefa de roteamento, o paralelismo através da GPU é uma alternativa que merece ser estudada melhor, devido a sua arquitetura e capacidade de entregar uma grande quantidade de processamento a um baixo custo. Dentre os métodos apresentados, o A* (*A Star*) apresenta o melhor desempenho e menor complexidade computacional; todavia, sua implementação em GPU é um desafio em aberto.

Agradecimentos

Este trabalho é financiado pelo Governo Federal do Brasil e do Estado do Rio Grande do Sul pelas seguintes agências: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS).

Referências

[1] G.-J. Nam, C. Sze, and M. Yildiz, "The ISPD Global Routing Benchmark Suite," in *Proceedings of the 2008 International Symposium on Physical Design*, ser. ISPD '08. New York, NY, USA: ACM, 2008, pp. 156–159.

[2] T. Szymanski, "Dogleg Channel Routing is NP-Complete," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 4, no. 1, pp. 31–41, January 1985.

[3] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*, 2nd ed. Norwell, MA, USA: Kluwer Academic Publishers, 1995.

[4] M. D. Moffitt, J. A. Roy, and I. L. Markov, "The Coming of Age of (Academic) Global Routing," in *Proceedings of the 2008 International Symposium on Physical Design*, ser. ISPD '08. New York, NY, USA: ACM, 2008, pp. 148–155.

[5] S. H. Gerez, *Algorithms for VLSI Design Automation*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[6] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs," in *Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays*, ser. FPGA '95. New York, NY, USA: ACM, 1995, pp. 111–117.

[7] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1989.

[8] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485.

[9] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

[10] F. J. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (Keynote Address)(Abstract Only)," in *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO 32. Washington, DC, USA: IEEE Computer Society, 1999, pp. 2–.

[11] M. Flynn, "Some Computer Organizations and Their Effectiveness," *Computers, IEEE Transactions on*, vol. C-21, no. 9, pp. 948–960, Sept 1972.

[12] D. SzyId, "Parallel Computation: Models And Methods," *IEEE Concurrency*, vol. 6, no. 4, pp. 79–80, Oct 1998.

[13] J. Croix and S. Khatri, "Introduction to GPU programming for EDA," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, Nov 2009, pp. 276–280.

[14] G.-J. Nam, M. Yildiz, D. Z. Pan, and P. H. Madden, "ISPD Placement Contest Updates and ISPD 2007 Global Routing Contest," in *Proceedings of the 2007 International Symposium on Physical Design*, ser. ISPD '07. New York, NY, USA: ACM, 2007, pp. 167–167.

[15] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, Nov 2006, pp. 464–471.

[16] ——, "FastRoute 2.0: A High-quality and Efficient Global Router," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 250–255.

[17] Y. Xu, Y. Zhang, and C. Chu, "FastRoute 4.0: Global router with efficient via minimization," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, Jan 2009, pp. 576–581.

[18] M. Hanan, "On Steiner's Problem with Rectilinear Distance," *SIAM Journal on Applied Mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[19] M. Cho and D. Pan, "BoxRouter: a new global router based on box expansion and progressive ILP," in *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006, pp. 373–378.

[20] M. Cho, K. Lu, K. Yuan, and D. Pan, "BoxRouter 2.0: architecture and implementation of a hybrid and robust global router," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov 2007, pp. 503–508.

[21] J. A. Roy and I. L. Markov, "High-performance Routing at the Nanometer Scale," in *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, ser. ICCAD '07. Piscataway, NJ, USA: IEEE Press, 2007, pp. 496–502.

[22] M. Moffitt, "MaizeRouter: Engineering an effective global router," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, March 2008, pp. 226–231.

[23] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, March 2008, pp. 232–237.

[24] Y. Han, D. Ancajas, K. Chakraborty, and S. Roy, "Exploring high throughput computing paradigm for global routing," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, Nov 2011, pp. 298–305.

[25] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A Parallel Integer Programming Approach to Global Routing," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 194–199.

Global Routing and Parallelism

Roger Caputo[†], Diego Tumelero[‡], Marcelo Johann[†] and Ricardo Reis[†]

[†] *PGMicro, Instituto de Informática*

[‡] *PPGC, Instituto de Informática*

Universidade Federal do Rio Grande do Sul, Brazil

{rcllanos, dtumelero, johann, reis}@inf.ufrgs.br

***Abstract*— This paper addresses the topic of global routing (GR) in VLSI (Very-Large-Scale Integration) and the use of parallelism as an approach to improve its performance. We review the history of GR and give a look at recent work that has contributed to the state-of-the-art in the field. Academic routers with better results in the past decade are briefly compared and also is shown the work developed by the Microelectronics Group (GME) of the Universidade Federal do Rio Grande do Sul (UFRGS). The use of graphics processing units (GPUs) and a combination of routing algorithms are promising approaches to reduce execution time and ameliorate GR's resolution for open challenges.**

***Keywords*— Global routing, parallelism, GPU, VLSI, EDA**

I. INTRODUCTION

In the design of integrated circuits, the global routing plays a key role and is one of the main challenges that Electronic Design Automation (EDA) tools must face. Routing is a very complex combinatorial problem. It has become a more elaborated process inside EDA due to the increasing number of transistors per die and the advent of a myriad of rules and constraints that each technology advance and device shrinking bring with. Despite all the complexities related to Very-Large-Scale Integration (VLSI) designs, make a circuit routable is arguably the most important task of physical synthesis, even more important than timing closure [1]. A design that does not accomplish the time metrics but is routable instead may require much less effort to be finished than another one that closes on timing (using for example Steiner estimates) but is *unroutable*.

A lot of research, leaded by different groups, has been done in recent years, generating a comprehensive understanding of the basic principles of global routing problem and presenting different approaches for timing, wirelength, congestion, runtime and/or 3D routing. After 2007, the International Symposium on Physical Design (ISPD) contests [2], [3] encouraged the improvement of academic routers, propitiating a competition space to share results and a great opportunity to know the latest industry benchmarks. However, some have questioned the scope of the problem specification and the via capacity consideration for multi-layer routing promoted by these contests, exposing that the research community has converged its effort to the wrong problem [4]. Certainly, the academic formulation has known limitations and particular characteristics that differentiate it from the industrial proposal.

In this paper, is addressed the state-of-the-art of global routing. In the sections III and IV are reviewed the basic principles behind modern approaches to the problem and are presented the most popular academic routers. The use of some parallelism techniques and their importance for global routing runtime is shown in section V. In the last part is introduced recent work developed by the Microelectronics Group (GME) of the *Universidade Federal do Rio Grande do Sul*, Brazil (UFRGS) related to GR.

II. BACKGROUND

Global routing precedes the detailed routing and follows placement and clock tree synthesis in the physical synthesis. It receives a given placement result with fixed locations of blocks and pins.

The global router must distribute the interconnections specified on the netlist across the available routing channels respecting the imposed constraints. After placement, the global router partitions the routing region (the circuit) into tiles and then decides the paths between the tiles for all nets. Figure 1 shows the general process of GR.

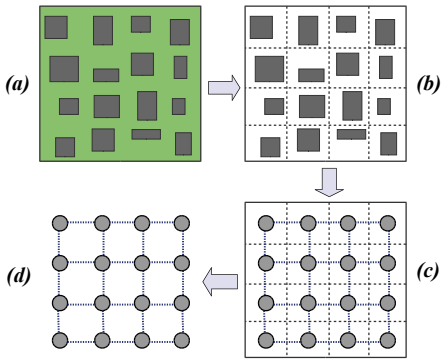


Fig. 1 (a) Placement result. (b) Circuit partitioned. (c) Cells and boundaries representation. (d) Global routing graph

In order to treat mathematically the global routing problem, it is characterized using the graph theory, where the circuit is represented by a grid-graph G that specifies two groups of components, a set of vertices V and a set of edges E. Each $v_i \in V$ denotes a particular region (cell or tile) of a metal layer; meanwhile each $e_{ij} \in E$ corresponds to a boundary between adjacent tiles. As explained in [5], these frontiers have a maximum allowable resource (m_{ij}) which can be used to measure the *overflow*, denoted as the total

amount of demand that exceeds the edge's capacity. At last, there is a set of nets N, and each $n_i \in N$ is composed of a set P_i of pins. Each one of these pins corresponds to a vertex v_i.

A solution is reached when all nets are routed while meeting the capacity constraint of each edge in the minimum runtime and satisfying any other constraint like wirelength, if specified. In some approaches, the use of techniques for parallelization shows results with substantial improvement in terms of execution speed [6].

III. Basic Algorithms

Numerous algorithms for global routing have been presented over the past three decades [7]. Nevertheless, the routers with better performance employ only a subset of these algorithms, using them like ingredients for a successful recipe. A brief description of the basic algorithms used in global routing is presented below.

- *Maze routing* [5] seeks the shortest path (avoiding obstacles) between two points on a grid. It is known as a *brute-force* method because allows all possible paths employing breadth-first search, Dijkstra's algorithm and A* search.
- *Pattern routing* [5] makes point-to-point connections following a small number of fixed shapes, usually minimal-length 'L' and 'Z' paths. It examines fewer grid edges but does not provide guarantees of best local solution.
- *Monotonic routing* follows a similar description as Pattern routing but with a less limiting technique and based on the monotonic function concept.
- *Steiner trees* are used to route multi-pin nets finding minimum total wirelength.
- *Multi-commodity flow (MCF)* uses the flow problem to solve a linear programming relaxation of GR [8].

IV. Most Famous Global Routers

Remarkable progress has been achieved recently in routing algorithms and EDA tools by university-industry researchers groups. Some routers have gained wide popularity inside academic spheres because of their performance, demonstrated at routing contests.

FastRoute [9], *FastRoute* 2.0 [10] and *FastRoute* 4.0 [11]. FastRoute use a congestion map to deform the structure of a Hanan grid [12] during Steiner tree generation followed by edge shifting and pattern routing. In [10], the router is enhanced with monotonic routing and multi-source multi-target maze routing. The latest version addresses the via number optimization problem throughout the entire global routing flow.

BoxRouter [13] and *BoxRouter* 2.0 [14]. The main idea of BoxRouter is to progressively expand a box initiated from the most congested region of the chip, applying an integer linear programming (ILP) formulation considering L-shaped patterns to re-route connections between successive boxes. The BoxRouter 2.0 is an improvement that uses a dynamic version of A* search and incorporates topology-aware rip-up to move wires from congested regions without changing the net topology.

Fairly Good Router (FGR) [15] is based on the PathFinder router originally developed for Field-Programmable Gate Arrays. It uses a particular function for congestion penalty and performs a fast layer assignment followed by a 3D clean-up.

MaizeRouter [16]. It uses two elementary edge-based operations (extreme edge shifting and edge retraction) and manipulates individual segments of nets one-at-a-time. Its approach is founded on interdependent net decomposition, in which routing solutions are implicitly maintained by collections of intervals instead of defined topologies.

NTHU-Route 2.0 [17] is based on rip-up and re-route. It uses a history-based cost function to distribute overflow and employs an identification method to specify the order for rip-up congested regions. Wirelength reduction is achieved through an adaptive multi-source multi-sink maze routing method.

Table I shows a comparison of the routers presented above, showing most of the known techniques used by each router.

TABLE I
Academic Global Routers Comparison

	FastRoute	FastRoute 2.0	FastRoute 4.0	BoxRouter	BoxRouter 2.0	FGR	MaizeRouter	NTHU-Route 2.0
Pattern routing	■	■	■	■	■		■	■
Monotonic routing		■	■					■
Maze routing	■	■	■	■	■	■	■	■
A* search					■	■	■	
FLUTE dependence	■	■	■	■	■		■	■
Topology reconstruction	■	■	■		■	■	■	■
Incremental						■	■	
Edge "sliding"		■	■				■	
Resource sharing	■	■	■			■	■	
ILP or MCF				■	■			
Congestion manipulation	■	■	■		■	■	■	■
History-based						■	■	■
Layer Assignment			■			■	■	■
Open source						■	■	

* Based on [5]

V. Parallelizing the Global Routing

Like many other processes in real world, some computer programs work sequentially, performing one operation after another, but many complex or large computational problems would take too much time to be completed using sequential processing. Those problems can often be divided into smaller ones and then computed concurrently. In other words the execution of tasks or calculations needed to solve the problem could be done at the same time (in parallel). Parallelism has

been employed for many years, mainly in high-performance computing. Focusing on power consumption reduction, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multicore processors [18].

In global routing, despite the use of heuristic methods or the implementation of improved algorithms, the process is still sluggish. This occurs for circuits with high integration scale and due to imposed constraints or physical variations attributable to fabrication processes like chemical mechanical planarization [1]. The execution time is an important constraint inside VLSI projects. The GR demands a large runtime when executed sequentially and a way to speed up the process is using parallelism (when possible) if the algorithm is scalable. That is whether it can be accelerated linearly with the use of various processors and whether the serial portion of the application is not too big to make the total runtime converge to it (*Amdahl's Law* [19]).

Madden [20] exposes some myths of parallel computing, he shows that the fact an algorithm is scalable, does not mean necessarily it will have high performance. In some cases is better to use the serial approach than the parallel, because the amount of effort and quantity of resources required to achieve a quality solution are important constraints. Much of the success of parallelism falls on the algorithmic efficiency and the main performance limitation of parallel approaches is due to some intrinsic serial characteristics of the tools.

From the state-of-the-art on global routing using parallel approaches, two main methods can be distinguished:

- Routing every net independently [21], this method leads to a problem because not all nets are independent, a situation that ends up generating sequential blocks with various nets and difficult a priori exploration of massive parallelism.
- The second method consists in partitioning the circuit [6] and treats each new part independently. With this approach, the biggest challenges are the partitioning itself and the interconnection of the boundaries partitions.

VI. Graphics Processing Units

We are in the golden era of computing through graphics processing units (GPUs), attributable to recent advances of the two largest companies in the industry, *AMD* and *nVIDIA* with the *CUDA* [22] technology and the open standard *OpenCL* [23]. This last one is a framework for writing programs that execute across heterogeneous platforms consisting not only of GPUs but also CPUs, digital signal processors and others.

What makes interesting of the GPU computing is its high processing capacity with a low cost on energy consumption, bigger area for components means more complexity and hence better performance. It is supported by the Pollack's rule [24], which states that performance of a chip is approximately equal to the square of its complexity ($performance \approx \sqrt{complexity}$). Put differently, "high-end video cards" have a considerably higher performance compared to high-end processors, with almost equivalent energy consumption. This is only possible due to the fact that the complexity on a GPU is partitioned over several smaller processing units (called *CUDA Cores* in *nVIDIA* models), also known as *shader units*. The argument of Fred Pollack is the same used by Intel to sell their CPUs with multiple processing cores [25].

The use of GPUs and parallel model for routing algorithms is an approach that could uncover an excellent solution to reduce execution time and improve router performance. Besides the GPUs, other form of parallelism is the use of a set of computers interconnected in *clusters* or *grids*. Clusters are groups of computers in a controlled environment and often dedicated to perform a specific task. Grids in this sense are no-common machines at different places that generally cede part of their processing capacity to projects through the internet [26]. In both cases the programming methodology is similar, they are multi-computers and are generally used message passing protocols for communication between nodes.

VII. Work Developed at the UFRGS

Important research has been done at the *Universidade Federal do Rio Grande do Sul* in the field of EDA. Despite the valuable investigation constantly performed at UFRGS that covers all steps of VLSI projects, here only will be referenced a couple of works related to global routing. Approaches that have shown improvements over the best ranked routers of the ISPD'08 contest.

Reimann [27] presented a global router that uses as principal tool the *rip-up and re-route* technique, 'with a differential method for sorting the nets'. In his approach were developed two versions of the same router, one (*WL version*) to achieve the shortest wirelength and the other (*RT version*) to seek the convergence of the solution as quickly as possible, with the lower number of iterations. *Minimum Spanning Tress* (MST) and *FLUTE* are the two forms used to build the routing nets. The Figure 2 shows the execution flow of the router.

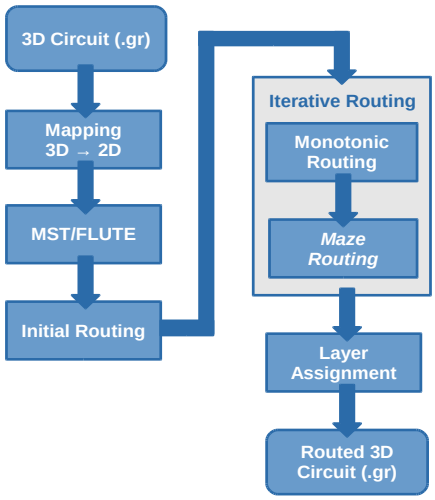


Fig. 2 Execution flow [27]

Despite the global router does not use techniques to identify congestion areas, nor post-routing optimizations and avoids any form of tuning for the benchmark circuits; it shows that is able to generate good results when compared with others academic solutions for the 3D circuits exhibited at ISPD'08. The WL version of the tool presents a difference, on average, of 1.78% more for the wirelength metric without considering the cost of the vias and 15.56% considering the via cost like unit of the wirelength; as for the RT version the difference was 3.82% and 17.03% more respectively.

In the context of this paper, is presented another work but focused on a different process of the physic synthesis. It is relevant by its approach and because the technology used on it can be extrapolated to the GR problem.

The placement is the VLSI process where the basic logic elements (cells) are organized inside the integrated circuit. This step is a NP-Hard combinatory problem and one form to find a satisfactory solution is using quadratic functions.

In [28], Flach et al., present an interesting form of using the power processing of a GPU to treat the cell placement problem. When the paper was written, there was not available a programming language like *CUDA* to employ easily the GPU for general purposes, thus the authors implemented a quadratic placement on *OpenGL*. They use *OpenGL* directives to manipulate elements of texture (matrix of pixels compounded of red, green, blue and alpha color channel), converting those elements in the tool memory and implementing simple algebra operations like *matrix-vector multiplication* and *dot product*.

The results present significant improvements of performance. In the *dot product*, *multiply and add*, and the *sparse-vector multiplication* operations the tool had a speed-up on runtime in the order of 1.95x, 3.45x and 3.05x respectively.

VIII. CONCLUSIONS

In this work, we took a look at the panorama of global routing nowadays, exposing the best ranked routers and comparing their characteristics.

We presented two important works developed at the UFRGS in the specific field. The Reinmann's approach shows results close to the obtained by the ISPD'08 contest competitors. From this can be inferred that the success of a router is not necessarily related to its complexity. In some cases the simplicity of a solution guarantees its success.

The work of Flach et. al, is important not only because it covers a physical synthesis step indispensable for the global routing process but also because it demonstrates the feasibility of using GPUs (employing either *CUDA* or *OpenGL*) to speed up computational problems like GR. Through the time, has been noticed that approaches originally conceived for different problems could be easily extrapolated for implementation inside EDA tools.

Most of modern global routers are a combination of well-known techniques and algorithms. How they are implemented inside the router and novel approaches could make a difference in terms of quality results.

The use of parallelism methods is a promising proposal to reduce the execution time of global routers. Despite the large power processing that could be achieved with those methods, they demand a considerable amount of resources that end up being a restriction in many cases. As exposed previously, various considerations have to be done when using parallelism as an approach in global routing inside EDA tools.

In order to conquer the open challenges, is not enough with just analyze current methods of global routing and learn from them, is equally important to have an open mind to imagine new efficient approaches.

REFERENCES

[1] C. J. Alpert, Z. Li, M. D. Moffitt, G. Nam, J. A. Roy, and G. Tellez, "What Makes a Design Difficult to Route", *Proceedings of the 2010 International Symposium on Physical Design* (*ISPD'10)*, pages 7-12, 2010.

[2] G. J. Nam, C. C. N. Sze, and M. C. Yildiz, "The ISPD global routing benchmark suite", *Proceedings of the 2008 International Symposium on Physical Design (ISPD'08)*, pages 156–159, 2008.

[3] G. J. Nam, M. C. Yildiz, D. Z. Pan, and P. H. Madden, "ISPD placement contest updates and ISPD 2007 global routing contest", *Proceedings of the 2007 International Symposium on Physical Design (ISPD'07)*, page 167, 2007.

[4] M. D. Moffitt, "Global Routing Revisited", *Proceedings of the 2009 International Conference on Computer-Aided Design* (*ICCAD'09)*, pages 805-808, 2009.

[5] M. D. Moffitt, J. A. Roy, and I. L. Markov, "The Coming of Age of (Academic) Global Routing [Invited Paper]", *Proceedings of the 2008 International Symposium on Physical Design (ISPD'08)*, pages 148-155, 2008.

[6] Y. Han, D. M. Ancajas, K. Chakraborky, S. Roy, "Exploring high throughput computing paradigm for global routing", *Proceedings of the International Conference on Computer-Aided Design (ICCAD'11),* pages 298-305, 2011.

[7] J. Hu and S. S. Sapatnekar, *A survey on multi-net global routing for integrated circuits*, Integration, the VLSI Journal, vol. 31, no. 1, pages. 1-49, 2001.

[8] C. Albrecht, "Global routing by new approximation algorithms for multi-commodity flow", *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems (TCAD)*, vol. 20, no. 5, pages 622-632, 2001.

[9] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement", *Proceedings of the 2009 International Conference on Computer-Aided Design* (*ICCAD'06)*, pages 464- 471, 2006.

[10] ——, "FastRoute 2.0: A high-quality and efficient global router", *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC'07)*, pages 250-255, 2007.

[11] Y. Xu, Y. Zhang and C. Chu, "FastRoute 4.0: Global Router with Efficient Via Minimization", *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC'09)*, pages 576-581, 2009.

[12] M. Hanan, "On Steiner's problem with rectilinear distance", *SIAM Journal of Applied Mathematics, vol. 14*, pages 255–265, 1966.

[13] M. Cho and D. Z. Pan, "BoxRouter: A new global router based on box expansion and progressive ILP", *Proceedings of the 43rd Design Automation Conference (DAC'06)*, pages 373-378, 2006.

[14] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router", *Proceedings of the International Conference on Computer-Aided Design (ICCAD'07)*, pages 503-508, 2007.

[15] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale", *Proceedings of the International Conference on Computer-Aided Design (ICCAD'07)*, pages 496-502, 2007.

[16] M. D. Moffit, "MaizeRouter: Engineering an effective global router", *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC'08)*, pages 226-231, 2008.

[17] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs", *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC'08)*, pages 232-237, 2008.

[18] K. Asanovic, et al., "The Landscape of Parallel Computing Research: A View from Berkeley", Technical Report No. UCB/EECS-2006-183, University of California, Berkeley, 2006.

[19] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities", *Proceedings AFIPS Conference*, pages 483–485, 1967.

[20] P. H. Madden, "Dispelling the Myths of Parallel Computing", *IEEE Design & Test of Computers*, *February*, pages 1–1, 2012.

[21] T.-H. Wu, A. Davoodi, J. T. Linderoth, "A parallel integer programming approach to global routing", *Proceedings of the 47th Design Automation Conference (DAC'10)*, pages 194-199, 2010.

[22] nVIDIA, "CUDA Parallel Computing Platform" [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[23] Khronos Group, "The open standard for parallel programming of heterogeneous systems", OpenCL [Online]. Available: http://www.khronos.org/opencl

[24] P. P. Gelsinger, "Microprocessors for the New Millennium: Challenges, Opportunities, and New Frontiers", in *IEEE International Solid-State Circuits Conference*, pages 22-25, 2010.

[25] S-L. Garver, B. Crepps, "The New Era of Tera-scale Computing", Intel Software [Online]. Available: software.intel.com/en-us/articles/the-new-era-of-tera-scale-computing

[26] M. A. R. Dantas, *Computação Distribuída de Alto Desempenho: redes, clusters e grids computacionais*, Axcel Books do Brasil Editora, Rio de Janeiro, 2005.

[27] T. J. Reimann, *Roteamento Global de Circuitos VLSI*, Master Dissertation, Institute of Informatics, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011.

[28] G. Flach, M. Johann, R. Hentschke, and R. Reis, "Cell placement on graphics processing units", *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design (SBCCI'07)*, 2007, page 87.