

Sheiny Fabre Almeida

**Aceleração da Legalização de Circuitos Integrados
Utilizando Particionamento com *k-d trees***

Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. José Luís A. Güntzel

Coorientador: Prof. Dr. Laércio L. Pilla

Florianópolis
2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Almeida, Sheiny Fabre

Aceleração da Legalização de Circuitos Integrados
Utilizando Particionamento com k-d trees / Sheiny
Fabre Almeida ; orientador, José Luís Almada Güntzel,
coorientador, Laércio Lima Pilla, 2019.

79 p.

Dissertação (mestrado) - Universidade Federal de
Santa Catarina, Centro Tecnológico, Programa de Pós
Graduação em Ciência da Computação, Florianópolis,
2019.

Inclui referências.

1. Ciência da Computação. 2. Legalização na Síntese
Física. 3. k-d tree. 4. Particionamento. 5.
Programação Paralela. I. Almada Güntzel, José Luís.
II. Lima Pilla, Laércio. III. Universidade Federal
de Santa Catarina. Programa de Pós-Graduação em
Ciência da Computação. IV. Título.


Sheiny Fabre Almeida

**Aceleração da Legalização de Circuitos Integrados Utilizando
Particionamento com k -d trees**

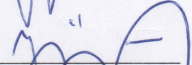
Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 22 de fevereiro de 2019.

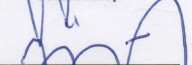
Banca Examinadora:



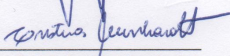
Prof. José Luis A. Guntzel, Dr.
Coordenador do Curso



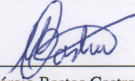
Prof. José Luis A. Guntzel, Dr.
Universidade Federal de Santa Catarina
Presidente



PI Prof. Marcelo de Oliveira Johann, Dr.
Universidade Federal do Rio Grande do Sul
(Videoconferência)



Prof. Cristina Meinhardt, Dr.
Universidade Federal de Santa Catarina



Prof. Márcio Bastos Castro, Dr.
Universidade Federal de Santa Catarina

Dedico este trabalho às pessoas que nunca desistiram de seus sonhos.

AGRADECIMENTOS

À minha mãe, Zilda Fabre por todo amor incondicional, ao meu pai, Francisco R. Almeida que sempre me apoiou em todas decisões e ao meu eterno melhor amigo e também irmão, Michel F. Almeida.

Agradeço à minha namorada Alessandra pelo amor, compreensão e companheirismo, por estar presente desde o início de minha graduação.

Aos meus orientadores Laércio Lima Pilla e José Luís Almada Güntzel, pela amizade, por toda confiança, dedicação e todo aprendizado proporcionado.

Aos colegas de laboratório: Renan O. Netto, Luiz De L. Cancellier, Tiago Fontana, Márcio Monteiro e demais colegas que de alguma forma participaram deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior- Brasil (CAPES) - Código de Financiamento 001.

*“A lesson without pain is meaningless.
For you cannot gain anything without sacrificing something else in return,
but once you have overcome it and made it your own...
you will gain an irreplaceable fullmetal heart.
(Hiromu Arakawa)”*

RESUMO

O avanço tecnológico viabiliza a fabricação de circuitos integrados formados por milhares de células. Devido a grande complexidade das regras de projeto, o posicionamento inicial das células desconsidera algumas dessas regras. A etapa de legalização é responsável por resolver algumas destas violações. Para preservar a qualidade inicial do posicionamento global, a legalização deve ser realizada de forma a minimizar a perturbação no posicionamento. O particionamento do circuito pode acelerar a legalização, pois reduz o tamanho da entrada para o algoritmo de legalização e também por permitir a legalização em paralelo das partições. Neste trabalho, será proposto o uso da estrutura de dados *k-d tree* para particionar o circuito. Como caso de uso, o algoritmo de legalização utilizado foi uma modificação do algoritmo clássico Abacus, juntamente com os circuitos das competições *ICCAD2015* e *ICCAD2017*. Os resultados incluem um *speedup* de até 36 utilizando um processador com 4 núcleos, em relação a versão sequencial sem particionamento para os circuitos da competição *ICCAD2015 CAD contest*. O particionamento também foi capaz de reduzir o deslocamento médio das células em até 17% em relação a versão sem particionamento.

Palavras-chave: Síntese Física. Legalização. Particionamento. *k-d tree*. programação paralela.

ABSTRACT

Technological advances have enabled the manufacture of integrated circuits composed of millions of cells. Due to the high complexity of design rules, the initial placement of cells disregards some of these rules. The legalization step is responsible for solving some of those violations. In order to preserve the quality of the global placement, the legalization must try to minimize the placement perturbation. The partitioning of a circuit may speed up the legalization, because it reduces the input size for the algorithm and it also enables the legalization of partitions in parallel. In this work, we propose the use of the k-d tree data structure to partition the circuit. As a case of use, the legalization algorithm chosen was a modification of the classic Abacus, with the circuits from the ICCAD2015 and ICCAD2017 contests. Experimental results show a speedup of up to 36 using a processor with 4 cores, in relation to the sequential legalization without partitioning for the circuits of the ICCAD2015 CAD contest. The partitioning has also enabled the reduction of the average displacement of cells by up to 17% in relation to the non-partitioning version.

Keywords: Physical Design Automation. Legalization Partitioning. k-d tree. parallel programming.

LISTA DE ILUSTRAÇÕES

Figura 1 – Principais passos do fluxo de projeto VLSI.	26
Figura 2 – Projeção do tempo de execução da legalização.	29
Figura 3 – Exemplo de layout.	32
Figura 4 – Fence region em forma de “L”.	33
Figura 5 – Exemplo de alinhamento com as linhas de energia.	34
Figura 6 – Exemplo de legalização.	34
Figura 7 – Exemplo de quadtree	46
Figura 8 – Exemplo de Rtree	46
Figura 9 – Exemplo de $k-d tree$ em um espaço bidimensional.	47
Figura 10 – Sobreposições entre macroblocos e células.	50
Figura 11 – Desalinhamento de células no particionamento.	51
Figura 12 – Exemplo de legalização utilizando $k-d trees$	54
Figura 13 – <i>Speedup</i> da versão paralela.	59
Figura 14 – <i>Speedup</i> execução paralela.	61
Figura 15 – Tempo de execução da versão paralela.	61
Figura 16 – <i>Speedup</i> da versão sequencial.	62
Figura 17 – Deslocamento médio da legalização com $k-d tree$	63
Figura 18 – Deslocamento máximo da legalização com $k-d tree$	64
Figura 19 – Tempo de execução da versão paralela ICCAD2017.	65
Figura 20 – Deslocamento médio para os circuitos do ICCAD2017.	66
Figura 21 – Deslocamento máximo para os circuitos do ICCAD2017.	66

LISTA DE TABELAS

Tabela 1	– Principais características dos trabalhos relacionados. . . .	42
Tabela 2	– Características dos circuitos da competição <i>ICCAD2015</i> <i>CAD Contest</i>	57
Tabela 3	– Características dos circuitos da competição <i>ICCAD2017</i> <i>CAD Contest</i>	58
Tabela 4	– Número de chamadas a função de união de partições. . .	60

LISTA DE ABREVIATURAS E SIGLAS

ASIC <i>Application-Specific Integrated Circuit</i>	25
CI <i>Circuito Integrado</i>	25, 26, 31
CMOS <i>Complementary Metal-Oxide Semiconductor</i>	25, 26
EDA <i>Electronic Design Automation</i>	25, 78, 79
FPGA <i>Field-Programmable Gate Arrays</i>	25
HPWL <i>Half-Perimeter Wire Length</i>	55
ILP <i>Integer Linear Programming</i>	39
IP <i>Intellectual Property</i>	25
LP <i>Linear Programming</i>	39
MMSIM <i>Modulus-based Matrix Splitting Iteration Method</i>	40, 41
SoCs <i>System-on-Chip</i>	25
VLSI <i>Very-Large-Scale Integration</i>	25

LISTA DE SÍMBOLOS

B	Limites da área do circuito	31
C	Conjunto de células	31, 34, 43, 44
C_{legal}	Lista de células legalizadas	44, 52, 53
F	Lista de células fixas	53
H_{row}	Altura de um <i>site</i>	31, 33
L	Algoritmo de legalização	52, 53
$L'(c_i)$	Posição em \mathbb{R}^2 da i ésima instância $\in C$ referente ao posicionamento global	34
$L(c_i)$	Posição em \mathbb{R}^2 da i ésima instância $\in C$	31, 34
P	Lista de pontos 2-D	48, 49
P_{left}	Pontos a esquerda da mediana	49
P_{right}	Pontos a direita da mediana	49
R	Conjunto de linhas do circuito	31, 43, 44
S	Conjunto de colunas do circuito	31
W_{site}	Largura de um <i>site</i>	31, 33
a	Área de uma partição	48, 49
$area_{left}$	Metade da esquerda da área de a	49
$area_{right}$	Metade da direita da área de a	49
$axis$	Eixo dimensional	48, 49
c_i	i ésima instância de célula $\in C$	31, 34, 43, 44
$circuit_{info}$	Informações do circuito	52, 53
h_{c_i}	Altura da i ésima instância de célula $\in C$	31, 44
l	Nível atual da árvore	48, 49

l_{limit}	Nível limite para construção da $k-d tree$	48, 49, 52
m	Ponto mediano de P ordenado	49
n_{root}	Nó raiz da $k-d tree$	48, 52
$node$	Nó da árvore $k-d tree$	49, 53
$node_{area}$	Área da partição de um nó da árvore	49
$node_{left}$	Nó com os elementos a esquerda da mediana	49
$node_{points}$	Vetor de pontos nó da $k-d tree$	49
$node_{right}$	Nó com os elementos a direita da mediana	49
r	Determinada linha do circuito $\in R$	43, 44
$success$	Valor booleano resultante da legalização	52, 53
w_{ci}	Largura da iésima instância de célula $\in C$	31

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo de legalização Abacus.	44
Algoritmo 2 – Modificação do algoritmo de legalização Abacus. . .	44
Algoritmo 3 – Uma <i>k-d tree</i> em um espaço bidimensional.	48
Algoritmo 4 – <i>kdtree</i> : função recursiva de particionamento	49
Algoritmo 5 – Legalização utilizando <i>k-d trees</i>	52
Algoritmo 6 – <i>kdtree_legalize</i> : Legalização recursiva das partições.	53

SUMÁRIO

1	INTRODUÇÃO	25
1.1	JUSTIFICATIVA	28
1.2	OBJETIVOS E CONTRIBUIÇÕES	29
1.3	ORGANIZAÇÃO DESTE TEXTO	30
2	CONCEITOS BÁSICOS SOBRE <i>LAYOUT</i>, POSICIONAMENTO E LEGALIZAÇÃO	31
2.1	<i>LAYOUT</i> E POSICIONAMENTO	31
2.2	LEGALIZAÇÃO	31
3	TRABALHOS RELACIONADOS	37
3.1	ALGORITMOS CLÁSSICOS DE LEGALIZAÇÃO	37
3.1.1	Hill (2002)	37
3.1.2	Spindler, Schlichtmann e Johannes (2008)	37
3.2	LEGALIZAÇÃO COM PARTICIONAMENTO	38
3.2.1	Brenner, (2012)	38
3.2.2	Karimpour Darav et al. (2017)	39
3.2.3	Hung, Chou e Mak (2017)	39
3.2.4	Ren et al. (2007)	39
3.2.5	Lee, Wu e Chiang (2010)	40
3.2.6	Zhu et al., (2018)	40
3.2.7	Resumo dos trabalhos correlatos	41
4	LEGALIZAÇÃO UTILIZANDO PARTICIONAMENTO	43
4.1	ABACUS MODIFICADO	43
4.2	ESTRUTURAS DE DADOS ESPACIAIS APLICADAS À LEGALIZAÇÃO	45
4.3	<i>K-D TREES</i> E SEUS ALGORITMOS	47
4.4	REQUISITOS ADICIONAIS PARA LEGALIZAÇÃO	49
4.5	LEGALIZAÇÃO UTILIZANDO <i>K-D TREES</i>	51
5	AMBIENTE EXPERIMENTAL	55
5.1	OBJETIVOS E MÉTRICAS	55
5.2	BENCHMARKS	55
5.3	EXECUÇÃO DOS EXPERIMENTOS	56
5.4	INFRAESTRUTURA EXPERIMENTAL	57
6	RESULTADOS	59
6.1	RESULTADOS ICCAD2015	59
6.2	RESULTADOS ICCAD2017	64
7	CONCLUSÕES E TRABALHOS FUTUROS	67
7.1	CONCLUSÃO	67

7.2	TRABALHOS FUTUROS	68
	BIBLIOGRAFIA	69

APÊNDICES **73**

APÊNDICE A – LISTA DE PUBLICAÇÕES E PRÊMIOS **75**

A.1	ARTIGOS PUBLICADOS DIRETAMENTE RELACIONADOS AO TEMA DE MESTRADO	75
A.1.1	XVIII Escola Regional de Alto Desempenho Fórum De Pós-graduação, @ ERAD 2018	75
A.1.2	<i>31 Symposium on Integrated Circuits and Systems Design, @ SBCCI 2018</i>	75
A.1.3	<i>International Symposium on Physical Design, @ ISPD 2019</i>	76
A.2	PARTICIPAÇÃO EM OUTRAS PUBLICAÇÕES DURANTE O MESTRADO	77
A.2.1	<i>30 Symposium on Integrated Circuits and Systems Design, @ SBCCI 2017</i>	77
A.2.2	<i>Workshop on Open-Source EDA Technology, @ ICCAD 2018</i>	78
A.3	PRÊMIOS	79
A.3.1	Richard Newton Young Fellow Award @ 54th DAC and Design Automation Summer School	79
A.3.2	2017 CAD Contest (Problem C:Multi-Deck Standard Cell Legalization) @ ICCAD 2017	79

1 INTRODUÇÃO

Os Circuitos Integrados (CIs) constituem o núcleo de qualquer equipamento eletrônico contemporâneo. A espantosa evolução da tecnologia *Complementary Metal-Oxide Semiconductor* (CMOS) viabilizou a fabricação de CIs com bilhões de transistores integrados em um único *chip*, dando origem ao jargão *Very-Large-Scale Integration* (VLSI) (KAHNG et al., 2011). Atualmente, o termo VLSI envolve diversas classes de CIs potencialmente muito complexos, tais como processadores de alto desempenho, *Field-Programmable Gate Arrays* (FPGAs), *Application-Specific Integrated Circuits* (ASICs) e *System-on-Chip* (SoCs) (D. A. PAPA, 2010). Consequentemente, o projeto VLSI de CIs contemporâneos tornou-se uma tarefa extremamente complexa que demanda o uso extensivo de ferramentas computacionais sofisticadas (ferramentas de *Electronic Design Automation* (EDA)) e de metodologias rígidas.

A fim de lidar com tal complexidade e ainda minimizar o tempo de projeto, e consequentemente, o tempo para o mercado (*time-to-market*), o fluxo de projeto VLSI faz uso de bibliotecas de componentes pré-projetados (chamadas de bibliotecas *standard cell*) e de blocos de propriedade intelectual (*Intellectual Property* (IP)), também referenciados como macroblocos), os quais podem ser memórias, processadores e/ou outros componentes arquiteturais específicos. Uma biblioteca *standard cell* (KAHNG et al., 2011) contém as informações geométricas referentes aos *layouts* necessários à fabricação de cada componente básico do CI, que são as portas lógicas, *latches*, *flip-flops* (mais recentemente, também registradores com dois ou mais *bits*), *full-adders* etc. Costuma-se utilizar o termo **célula** para se fazer referência a uma instância de *layout* de um componente básico do CI. Neste contexto, vale observar que um dado componente básico, como por exemplo uma porta NAND de duas entradas, normalmente possui diversos *layouts* distintos (tipicamente, entre 4 e 8)¹, os quais se diferenciam pelos tamanhos dos transistores: células com transistores maiores apresentam atrasos menores, porém consomem mais energia. Além dos *layouts*, a biblioteca *standard cell* também contém informações de atraso, de potência e outras propriedades elétricas de cada componente, obtidas por meio de simulação elétrica.

O fluxo de projeto VLSI (Figura 1) inicia com a especificação do sistema em alto nível de abstração usando alguma linguagem adequada (por exemplo, SystemVerilog, SystemC) e segue uma sequência de passos, comumente

¹ Para o passo de síntese física, apresentada mais adiante, as informações geométricas necessárias são a altura e a largura da célula e as posições e os materiais usados nos seus pinos de entrada e saída.

referenciados por projeto arquitetural, projeto funcional e lógico e síntese física, os quais refinam a descrição inicial agregando informações necessárias à fabricação. Após a fabricação, o CI é encapsulado e testado.

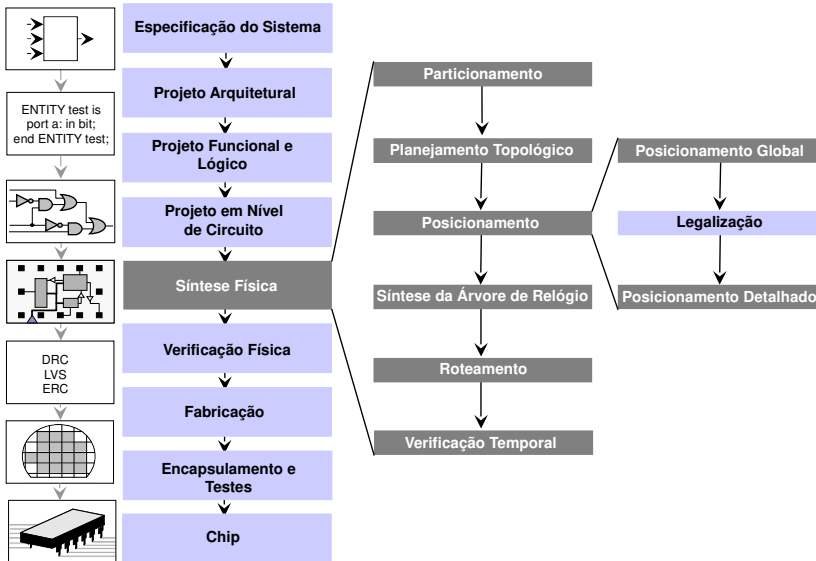


Figura 1: Principais passos do fluxo de projeto VLSI. Adaptado de (KAHNG et al., 2011).

A síntese física (Figura 1), escopo do presente trabalho, gera uma descrição compatível com as especificações da tecnologia CMOS a ser usada na fabricação do CI. A execução do fluxo de projeto não é necessariamente linear. Caso alguma etapa não seja possível de ser finalizada, etapas anteriores serão refeitas. Para tanto, ela precisa posicionar e rotear os elementos do circuito sobre uma região 2-D (KAHNG et al., 2011), por meio das seguintes etapas:

Particionamento: Dado o grande número de componentes dos circuitos atuais, o objetivo do particionamento é dividir o circuito em subcircuitos ou módulos, de forma que estes possam ser analisados e projetados individualmente. O particionamento pode tirar proveito da hierarquia contida na descrição inicial do sistema, ou pode ser feito de forma automática, sem respeitar tal hierarquia.

Planejamento Topológico: Durante esta etapa são identificados os grandes componentes do circuito como blocos de memória, blocos de propriedade intelectual e grandes módulos. Nesta etapa são definidas as posições

desses grandes módulos, dos pinos de entrada e saída e como eles se conectarão. O planejamento topológico também serve para estimar o comprimento das interconexões, os atrasos etc.

Posicionamento: Esta etapa é importante pois, afeta o Roteamento, distribuição de calor, consumo energético e desempenho do circuito (L.-T. WANG; CHANG; CHENG, 2009). Para reduzir a complexidade, esta etapa é subdivida em Posicionamento Global, Legalização e Posicionamento Detalhado, conforme detalhado a seguir.

- **Posicionamento Global:** É responsável por encontrar uma solução inicial que otimize alguma figura de mérito como por exemplo, comprimento das interconexões, acessibilidade aos pinos das células etc (ALPERT et al., 2012; CHEN et al., 2017). No entanto, para encontrar uma solução para circuitos com milhões de portas lógicas dentro de um tempo de execução aceitável, o posicionamento global desconsidera algumas restrições de *layout* como, por exemplo, sobreposições de células, desalinhamento em relação às linhas e colunas do circuito.
- **Legalização:** É responsável por remover as violações causadas pelo Posicionamento Global. Para que a qualidade do Posicionamento Global seja preservada, a legalização deve ser feita de forma a perturbar o mínimo possível a solução encontrada pelo posicionamento.
- **Posicionamento Detalhado:** Realiza otimizações locais, rearranjando pequenos grupos de células, porém, mantendo a legalidade do circuito. Para otimizar o processamento, a Legalização dos subconjuntos de células pode ser feita de forma incremental.

Síntese da Árvore de Relógio: Devido à alta atividade de chaveamento em circuitos síncronos, a rede de relógio, comumente referenciada por árvore de relógio, é responsável pela maior parte do consumo energético. Além disso, o atraso na propagação do sinal de relógio também influencia a frequência máxima na qual o circuito pode operar. Logo, para reduzir o impacto dos fatores mencionados acima e ao mesmo tempo minimizar o congestionamento ao acesso dos pinos de elementos sequenciais, a árvore de relógio é definida antes da realização das demais interconexões do circuito.

Roteamento: Após o posicionamento dos elementos, estes precisam ser conectados. Devido a sua grande complexidade, esta tarefa é subdividida em duas etapas:

- **Roteamento Global:** O circuito é particionado em um *grid* onde cada elemento (chamado de *gcell*), possui uma capacidade de roteamento. O

objetivo desta etapa é estimar quais regiões do circuito serão utilizadas para realizar o roteamento de cada interconexão.

- **Roteamento Detalhado:** O roteamento detalhado determina com exatidão em quais trilhas cada interconexão será roteada. Essas informações servem para garantir que todas as conexões serão roteáveis e que todos os pinos de células podem ser alcançados. Após o roteamento detalhado, estimativas mais precisas sobre potência e atraso podem ser obtidas.

Verificação Temporal: Responsável por estimar o atraso máximo do circuito através de simulações. Caso o circuito não corresponda as especificações de projeto, será necessário repetir etapas anteriores até alcançar a convergência (*time closure*).

1.1 JUSTIFICATIVA

A legalização de um circuito pode demandar um tempo considerável da etapa de síntese física, conforme o nível de otimização requerido, restrições tecnológicas, número e distribuição das células etc. Por outro lado, é relatado na literatura que o tempo para uma execução de todas as etapas da síntese física para circuitos contemporâneos não deve ultrapassar 12 horas, para que os projetistas possam explorar o espaço de solução mediante novas rodadas de síntese com parâmetros alternativos, sem comprometer o tempo total de projeto (D. PAPA et al., 2011). A fim de se ter uma ideia mais concreta do problema, a Figura 2 demonstra o crescimento do tempo de execução (sequencial) de um algoritmo quadrático de legalização, aplicado a circuitos com formatos industriais fornecidos pelas competições ICCAD2015 e ICCAD2017 (KIM et al., 2015; DARAV et al., 2017).

Estas questões, juntamente com o contínuo crescimento do número de células e restrições de *layout* contemporâneas, tornam a legalização um tópico relevante onde melhorias substanciais ainda são necessárias.

Para reduzir o tempo de execução, diversos trabalhos encontrados na literatura utilizam técnicas de particionamento do circuito (KARIMPOUR DARAV et al., 2017; LEE; WU; CHIANG, 2010; REN et al., 2007; BRENNER, 2012; HUNG; CHOU; MAK, 2017). No entanto, todos eles dividem o circuito em *bins* retangulares de mesmo tamanho e atribuem as células ao *bin* mais próximo. Nenhum deles considera utilizar uma estrutura de dados espacial, o que seria mais apropriado, pois se trata de um problema computacional geométrico. Adicionalmente, utilizar partições de mesmo tamanho pode não ser adequado pois as células não se distribuem uniformemente por todo o circuito. Sendo assim, esses trabalhos ainda necessitam de algum mecanismo adicional para redistribuir as células de partições densas, como por exemplo,

fluxo em redes, o que pode aumentar o tempo de execução da legalização. O uso de uma estrutura de dados espacial como a *k-d tree* (BENTLEY, 1975) garante um particionamento balanceado (partições com mesmo número de células), além de viabilizar a legalização em paralelo de cada partição.

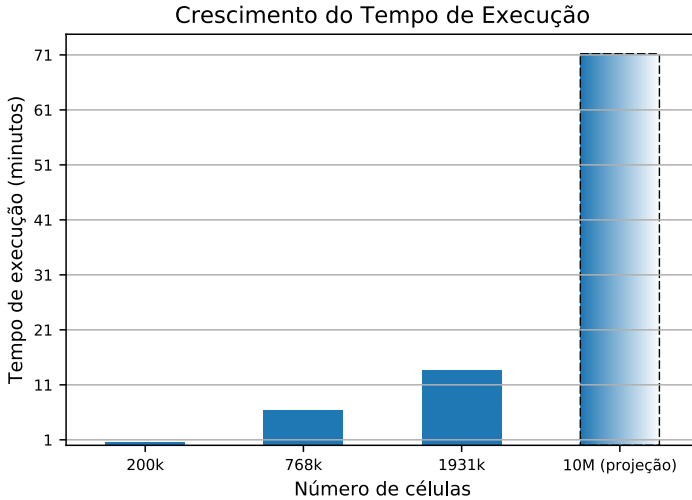


Figura 2: Tempo de execução de um algoritmo quadrático de legalização ao variar o número de células, onde a projeção otimista de 10M de células segue uma extrapolação linear em relação ao maior circuito (1931K).

1.2 OBJETIVOS E CONTRIBUIÇÕES

O objetivo deste trabalho é acelerar a legalização por meio de uma técnica de particionamento do circuito utilizando uma estrutura de dados espacial denominada *k-d tree*, reduzindo o tamanho das instâncias a serem tratadas e viabilizando a legalização em paralelo das partições

Os objetivos específicos deste trabalho são:

- Implementar a estrutura de dados *k-d tree* juntamente com as adaptações necessárias para realizar a legalização das partições;
- Verificar o impacto do particionamento na legalização em comparação com o tempo de execução (execução paralela) e a escalabilidade da técnica;

- Verificar o escalabilidade do particionamento quanto ao tamanho das partições;
- Aplicar a técnica utilizando *benchmarks* disponibilizados pela indústria.

1.3 ORGANIZAÇÃO DESTE TEXTO

O Capítulo 2 contém conceitos básicos sobre *layout*, posicionamento e a descrição do problema da legalização. O Capítulo 3 apresenta os trabalhos correlatos a respeito dos algoritmos de legalização e os que realizam a legalização utilizando particionamento. No Capítulo 4 são apresentados os detalhes relacionados à estrutura de dados *k-d tree* e o de como aplicá-la na legalização. Os Capítulos 5 e 6 descrevem o ambiente experimental e os resultados, respectivamente. Por fim, o Capítulo 7 apresenta as conclusões obtidas e propostas de trabalhos futuros.

2 CONCEITOS BÁSICOS SOBRE *LAYOUT*, POSICIONAMENTO E LEGALIZAÇÃO

Este capítulo tem como objetivo apresentar informações sobre *layout*, posicionamento e o problema de legalização para CIs, conceitos básicos necessários para o entendimento deste trabalho. A Seção 2.1 apresenta conceitos relacionados ao *layout*, posicionamento e restrições de projeto contemporâneas referentes à legalização. Em seguida, na Seção 2.2, é apresentado o problema de legalização para CIs e sua formulação matemática.

2.1 *LAYOUT* E POSICIONAMENTO

Os elementos do circuito são representados por formas geométricas retangulares (inclusive as interconexões, porém estas em geometria *manhattan*, onde apenas orientações verticais e horizontais são permitidas). Alguns elementos podem ter suas posições fixas ao passo que outros podem ser móveis. Usualmente, blocos de propriedade intelectual (macroblocos) possuem posição fixa e tamanho maior que as células do circuito.

O circuito é formado por um conjunto de células $C = \{c_1, c_2, \dots, c_n\}$. Cada c_i do conjunto C representa uma instância da biblioteca de células e portanto possui uma geometria associada. Essa geometria é composta de um par $(x(c_i), y(c_i))$ que define a altura (h_{c_i}) e largura (w_{c_i}) da instância. A área do circuito é dividida por um conjunto de linhas $R = \{r_1, r_2, \dots, r_n\}$, e colunas $S = \{s_1, s_2, \dots, s_n\}$, formando uma grade. Cada uma dessas linhas e colunas possui uma determinada altura H_{row} e largura W_{site} . Sendo assim, os elementos devem ser posicionados em uma superfície bidimensional delimitada por $B = (X_{left}, X_{right}, Y_{bottom}, Y_{top})$. Um posicionamento é uma função que mapeia cada c_i em uma localização no \mathbb{R}^2 representada por $L(c_i)$. Neste trabalho todas as informações referentes ao posicionamento de células se referem a posição do canto inferior esquerdo das células.

A Figura 3 ilustra o *layout* de um circuito, onde as células do circuito estão representadas por retângulos (c_1 a c_{11}), sendo o retângulo preto no centro do circuito um macrobloco (posição fixa). A posição das células reflete a saída de um posicionamento global, onde as células não necessariamente estarão alinhadas às linhas e colunas do circuito, como pode ser verificado para o caso de c_4 .

2.2 LEGALIZAÇÃO

Esta subseção apresenta algumas das principais regras de legalização que devem ser satisfeitas para que a fabricação do circuito seja possível. Também

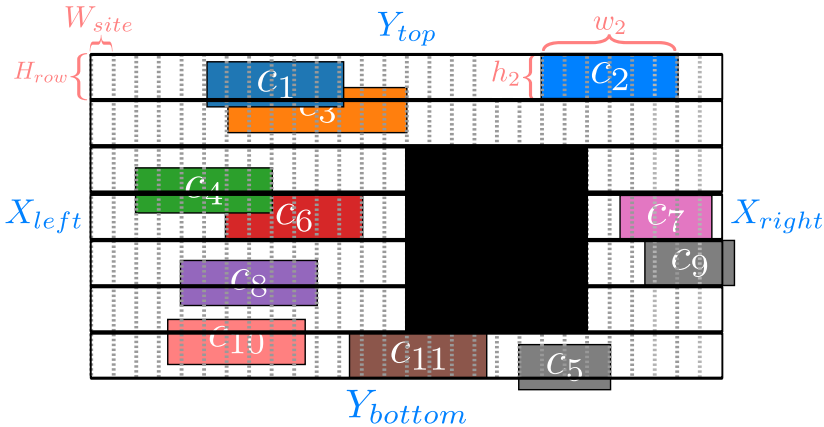


Figura 3: Exemplo de layout. Cada retângulo representa uma instância da biblioteca de células ou um macrobloco. Neste caso c_1 a c_{11} são células e o retângulo preto no centro do circuito um macrobloco (posição fixa).

será apresentada a formulação matemática para o problema de legalização. Na legalização as seguintes regras são verificadas:

1. Não podem existir sobreposições entre células.
2. Todos os elementos devem estar posicionados dentro dos limites do circuito.
3. As células devem estar alinhadas com as linhas e colunas do circuito.

Para que as máscaras de fotolitografia sejam geradas corretamente é necessário remover as sobreposições entre as células. As sobreposições podem ter sido inseridas pelo posicionamento global ou até mesmo por melhorias incrementais no posicionamento, pois as sobreposições entre células podem ser desconsideradas durante essas etapas devido a suas altas complexidades. Pelo mesmo motivo citado anteriormente, também é necessário verificar se todas as posições das células estão dentro dos limites do circuito e também alinhar as células às linhas e colunas do circuito.

Para melhorar o desempenho do circuito é desejado que determinados conjuntos de células que compõem determinada funcionalidade sejam posicionados em uma região específica. Para atender a esta reserva de espaço, as *fence regions* são utilizadas. *Fence regions* são regiões compostas por retângulos não necessariamente adjacentes onde somente os membros de um

conjunto específico de células devem ser posicionados. A Figura 4 ilustra um exemplo de *fence region* em forma de “L” onde os retângulos violetas são células pertencentes a *fence region* e os retângulos em azul são células externas a *fence region*.

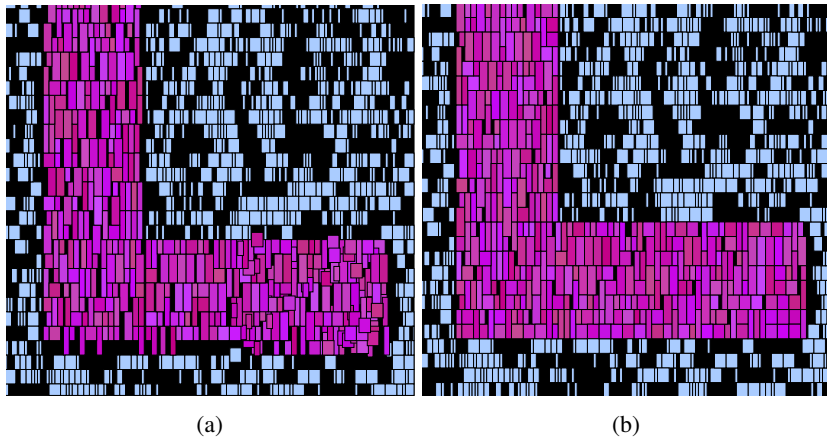


Figura 4: *Fence region* em forma de “L”: (a) células da *fence region* antes da legalização; (b) uma possível solução para a legalização da *fence region*.

Células com altura de múltiplas linhas estão cada vez mais presentes em projetos contemporâneos para atingir melhor desempenho, roteabilidade e otimização de área (DARAV et al., 2017). As linhas de alimentação *vss* (positiva) e *vdd* (negativa) estão distribuídas de forma entrelaçada entre o topo e a base de cada linha do circuito. Para energizar apropriadamente as células é necessário que os pinos de alimentação sejam apropriadamente conectados na correspondente linha de alimentação. Conforme ilustrado pela Figura 5 cada célula deve ser posicionada com seus pinos de alimentação apropriadamente conectados. As células de tamanho ímpar podem ser posicionadas em qualquer linha, onde para as linhas pares o espelhamento vertical se faz necessário. Para as células com altura par o espelhamento não tem efeito, sendo assim estas devem estar posicionadas somente em linhas pares ou ímpares. Por exemplo, C_4 em linhas pares e C_2 em linhas ímpares.

Na Figura 6, a legalização de um circuito com sete células é ilustrada. O circuito é dividido em linhas e colunas, onde as linhas são segmentadas em *sites* com altura H_{row} e largura W_{site} . A legalização deve remover as sobreposições entre células como, por exemplo, c_2 e c_4 na Figura 6(a). Todas as células devem estar posicionadas dentro dos limites do circuito e alinhadas

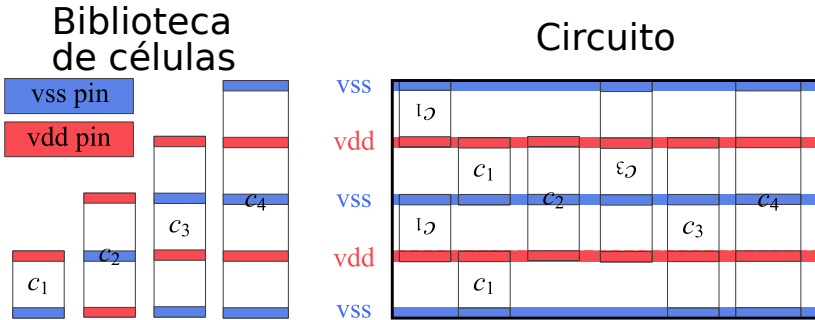


Figura 5: Exemplo de alinhamento com as linhas de energia, onde à esquerda estão ilustrados os pinos de alimentação para células de altura 1 à 4 e à direita um possível posicionamento para estas instâncias. Adaptado de (ZHU et al., 2018)

aos *sites*. Por exemplo, na Figura 6(a) c_4 não está alinhada. Além das restrições mencionadas, a legalização deve **minimizar o deslocamento** das células para que seja preservada a qualidade das otimizações já realizadas.

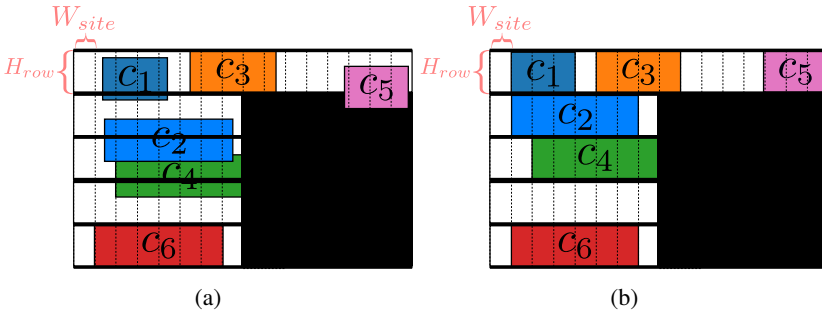


Figura 6: Exemplo de legalização: (a) circuito ilegal com sobreposições e células desalinhadas; (b) circuito legalizado.

O problema de legalização pode ser formulado matematicamente da seguinte forma: Seja $C = \{c_1, c_2, \dots, c_n\}$ o conjunto de células do circuito onde $(x(c_i), y(c_i))$ representa a largura e altura de uma determinada instância de célula c_i . O objetivo da legalização é minimizar o deslocamento horizontal e vertical da solução fornecida pelo posicionamento global, satisfazendo restrições de *layout* conforme a Equação 2.1, sendo $L(c_i)$ a posição do canto inferior esquerdo de c_i e $L'(c_i)$ a posição referente ao posicionamento global. É importante ressaltar que neste trabalho todos os deslocamentos considerados

estão se referindo a distância *manhattan*.

$$\text{Minimizar } \sum_{c_i \in C} |(x_{L(c_i)} - x_{L'(c_i)})| + |(y_{L(c_i)} - y_{L'(c_i)})| \quad (2.1a)$$

Sujeito a

$$x_{L(c_j)} - x_{L(c_i)} \geq w_i \vee x_{L(c_i)} - x_{L(c_j)} \geq w_j$$

$$\vee y_{L(c_j)} - y_{L(c_i)} \geq h_i \vee y_{L(c_i)} - y_{L(c_j)} \geq h_j \forall c_i, c_j \in C \quad (2.1b)$$

$$X_{left} \leq x_{L(c_i)} + w_i \leq X_{right} \text{ e } Y_{left} \leq y_{L(c_i)} + h_i \leq Y_{right} \forall c_i \in C \quad (2.1c)$$

$$x_{L(c_i)} \bmod W_{site} = 0 \text{ e } y_{L(c_i)} \bmod H_{row} = 0 \forall c_i \in C \quad (2.1d)$$

A Equação 2.1b garante que não existe nenhuma sobreposição entre as células do circuito. A Equação 2.1c verifica que todas as células estão dentro dos limites de área do circuito. Por fim a Equação 2.1d garante o alinhamento vertical e horizontal das células em relação as linhas e colunas do circuito.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados à legalização. Na Seção 3.1 são mencionados algoritmos clássicos de legalização. A Seção 3.2 inclui os algoritmos de legalização que consideram estratégias de particionamento. É importante ressaltar que este capítulo não faz uma análise exaustiva de cada trabalho citado, mas aborda as características relevantes dos trabalhos que realizam a legalização utilizando particionamento. E ao final deste capítulo é apresentado um resumo sobre principais trabalhos relacionados.

3.1 ALGORITMOS CLÁSSICOS DE LEGALIZAÇÃO

3.1.1 Hill (2002)

Hill (2002) propôs um algoritmo guloso chamado Tetris que legaliza uma célula por vez. As células são ordenadas em relação ao eixo x e posicionadas uma a uma na linha que fornece o menor custo. O custo de posicionar uma célula é calculado utilizando a distância *manhattan* entre a posição alvo e a posição inicial fornecida ao algoritmo de legalização. Após avaliar um conjunto de linhas próximas à posição inicial, a célula é então posicionada na linha que oferecer o menor custo.

Após posicionada, a célula não pode ser movida, e pelo fato do algoritmo seguir um ordenamento pelo eixo x , todo o espaço à esquerda da célula posicionada não poderá ser explorado. Sendo assim, a principal limitação desta técnica consiste em não permitir o movimento de células já posicionadas, o que pode levar a soluções subótimas.

3.1.2 Spindler, Schlichtmann e Johannes (2008)

Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008) é uma melhoria em relação ao Tetris. Este algoritmo também legaliza as células uma a uma utilizando ordenamento. Dessa forma, a ordem relativa entre as células é preservada. No entanto, a melhoria apresentada reside no fato de permitir o movimento de células que já foram legalizadas.

Para cada célula, o algoritmo avalia o custo de posicioná-la em cada uma das linhas próximas a sua posição inicial. O custo de posicionar uma célula é descrito na Equação 3.1, onde o conjunto de células em uma determinada linha r_j é descrito por $C(r_j)$. Tal custo considera que as células já posicionadas na linha podem ser movidas de forma a minimizá-lo. Como consequência disso,

as células podem ser realocadas mesmo após terem sido legalizadas, levando a uma melhor solução em relação ao Tetris.

$$\text{custo}(L) = \sum_{r_j \in \mathcal{R}} \sum_{c_i \in C(r_j)} \omega(c_i) \sqrt{(x(c_i) - x'(c_i))^2 + (y(c_i) - y'(c_i))^2} \quad (3.1)$$

A função custo avalia a soma ponderada das distâncias euclidianas entre a posição legal das células com suas posições iniciais. O algoritmo proposto considera o número de pinos da célula como o peso $\omega(c_i)$. Entretanto, o peso poderia ser a influência da célula em relação ao atraso crítico do circuito, potência e diversas outras figuras de mérito.

Apesar do aumento no tempo de processamento devido ao uso de programação dinâmica, este ainda é drasticamente reduzido ao limitar o número de linhas avaliadas pelo algoritmo, pois não é necessário que todas as linhas sejam avaliadas (aumento de somente 7% no tempo de execução em relação a Hill (2002)).

3.2 LEGALIZAÇÃO COM PARTICIONAMENTO

Algoritmos clássicos como Tetris (HILL, 2002) e Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008) não foram concebidos para tratar das restrições de *layout* mencionadas na Seção 2.2. Além disso, eles não possuem o desempenho necessário para tratar circuitos que contêm milhões de células. Apesar disso, um grande número de legalizadores do estado da arte dependem fortemente do Abacus (C.-H. WANG et al., 2017; KENNINGS; DARAV; BEHJAT, 2014; POPOVYCH et al., 2014).

3.2.1 Brenner, (2012)

Diferente de abordagens anteriores, (BRENNER, 2012) propôs um algoritmo de fluxo de redes baseado em *successive shortest path*. O algoritmo computa os menores caminhos entre nós fonte e destino utilizando uma versão modificada do algoritmo de Dijkstra. Durante as iterações são selecionados somente caminhos aumentadores que não gerem violações, capazes de mover todas as células excedentes para as demais partições. O custo entre os *bins* é calculado considerando o deslocamento quadrático das células excedentes. A maior limitação deste trabalho é não considerar o deslocamento máximo da solução durante a busca pelos caminhos.

3.2.2 Karimpour Darav et al. (2017)

O problema de legalização foi modelado como um problema de fluxo de redes (KARIMPOUR DARAV et al., 2017). O circuito é dividido em *bins* de forma semelhante a (BRENNER, 2012), e um grafo bipartido é construído conectando os nós sobrecarregados e o os nós subcarregados. O particionamento do circuito em *bins* é feito utilizando as linhas do circuito, segmentadas por cortes horizontais de mesmo tamanho, onde caso um **bin** se torne pequeno devido a sobreposição de macroblocos este é unificado com os *bins* horizontais. O fluxo de redes é resolvido para o grafo com o objetivo de identificar movimentos entre *bins*. Neste trabalho, diferente de (BRENNER, 2012) a restrição de máximo deslocamento é considerada durante a execução do fluxo de redes, pois ela se torna mais difícil de ser resolvida durante a legalização. Após a distribuição das células, cada *bin* é legalizado separadamente. O algoritmo proposto mantém controle sobre a ordem das células nos *bins* para que sejam respeitadas regras de espaçamento mínimo entre células¹.

3.2.3 Hung, Chou e Mak (2017)

Em (HUNG; CHOU; MAK, 2017) o circuito é particionado para legalizar cada partição utilizando um algoritmo de legalização *Integer Linear Programming* (ILP), pois legalizar o circuito inteiro com ILP se torna muito custoso devido ao grande número de variáveis. Inicialmente, é criado o *grid* de *bins*, onde cada célula é assinalada ao *bin* mais próximo. Após o assinalamento, *bins* que se tornaram muito densos distribuem células para *bins* vizinhos utilizando algoritmo de fluxo de redes. Para cada linha, de baixo para cima, os *bins* são legalizados em paralelo permitindo que as células ultrapassem somente os limites superiores do *bin*. A legalização de cada *bin* é feita utilizando ILP onde cada posição candidata não gera sobreposições e está alinhada com as linhas de energia. Finalmente para cada linha um algoritmo *Linear Programming* (LP) é utilizado para minimizar o deslocamento horizontal da solução.

3.2.4 Ren et al. (2007)

Algoritmos como o Tetris e Abacus legalizam uma célula por vez, Ren et al. (2007) propuseram um método baseado no fenômeno físico de difusão que iterativamente espalha as células para remover as sobreposições.

¹ A regra de espaçamento mínimo entre cada modelo de porta lógica é determinada pela biblioteca e serve para que o acesso aos pinos da célula não sejam bloqueados durante a etapa de roteamento.

A aplicação deste trabalho é melhorar a qualidade de um posicionamento já legalizado, como por exemplo: para satisfazer violações de congestionamento durante o roteamento, remover violações de sobreposição após a inserção de *buffers*, dimensionamento de portas e até mesmo problemas gerados por um roteamento muito denso. A vantagem do espalhamento é a preservação das características de vizinhança e portanto das otimizações já realizadas. Os melhores resultados obtidos foram utilizando *bins* de tamanho igual a altura de quatro linhas.

As células são assinaladas a um *grid* de *bins*, onde a densidade de cada *bin* é responsável pela direção e velocidade, espalhando as células aos *bins* vizinhos. Isso resulta em movimentos mais suaves. No entanto, pode ser que mesmo assim seja necessário executar uma legalização final para remover as violações restantes.

3.2.5 Lee, Wu e Chiang (2010)

Para reduzir o tempo de execução, Lee; Wu; Chiang (2010) propuseram uma estratégia de particionamento que divide o circuito em *bins* de mesmo tamanho para executar o Abacus em cada partição. Cada célula é assinalada ao *bin* mais próximo, onde caso este se torne muito denso², então é feita uma união entre as partições vizinhas. Essa união gera partições em forma de cruz ou quadrado³ que potencialmente serão menos densas. O tamanho utilizado para os *bins* está descrito nas Equações 3.2 e 3.3.

$$altura_bin = altura_linha \times \sqrt{num_linhas} \quad (3.2)$$

$$largura_bin = 0.1 \times largura_do_circuito \quad (3.3)$$

3.2.6 Zhu et al., (2018)

O trabalho proposto por (ZHU et al., 2018) aprimora o *solver* matemático *Modulus-based Matrix Splitting Iteration Method* (MMSIM) (CHEN et al., 2017) que utiliza programação quadrática. A função objetivo foi modificada para incluir o deslocamento vertical e nas restrições do problema foram adicionadas regras de espaçamento mínimo entre as células.

Em um fluxo iterativo, o *solver* é utilizado e então células com alto deslocamento são identificadas e movidas para outras linhas. Após a convergência, é possível que existam sobreposições entre as células devido a imprecisão

² Neste trabalho o limite de densidade máximo de uma partição é de 97% em relação a área.

³ A partição em forma de cruz é criada a partir da união com os *bins* vizinhos verticais e horizontais, a partição quadrada também pode ser formada adicionando os vizinhos diagonais.

computacional. As células também podem ficar fora dos limites do circuito pois esta restrição é relaxada pelo MMSIM.

Para tratar esses problemas o circuito é particionado utilizando um *grid* onde cada *bin* possui altura e largura equivalente ao tamanho de oito linhas. A legalização de cada *bin* é feita utilizando um grafo bipartido, onde para cada tamanho de célula é realizado o melhor assinalamento para a redução do deslocamento.

3.2.7 Resumo dos trabalhos correlatos

A Tabela 1 resume as principais características dos trabalhos relacionados a respeito da técnica proposta. Todos os trabalhos utilizam técnicas similares, que consistem em particionamento do circuito utilizando um *grid* de *bins*. Dentre os trabalhos, nenhum deles disponibiliza informações relacionadas ao tipo de estrutura de dados utilizado para o particionamento e poucos deles consideram a execução em paralelo das instâncias.

Neste contexto, é proposta uma técnica de particionamento para legalização através do uso de *k-d trees*. Este mecanismo fornece um particionamento balanceado do circuito, e também viabiliza soluções paralelas e determinísticas, pelo fato das partições serem independentes. Além disso, esta técnica é independente ao algoritmo de legalização utilizado. O algoritmo de legalização Abacus foi utilizado como caso de uso da técnica proposta, juntamente com os circuitos da competição *ICCAD2015* (KIM et al., 2015) e *ICCAD2017* (DARAV et al., 2017).

	Estrutura de dados	Método de particionamento	Algoritmo de legalização	Etapa da síntese física	Infraestrutura experimental	Uso de paralelismo
Bremer (2012)	Não informada	Grid de bins	Fluxo de redes	Legalização	Circuito Industriais (IBM) ISPD 2006	Não
Karimpour Darav et al. (2017)	Não informada	Grid de bins	Fluxo de redes	Legalização	ISPD 2004 ISPD 2007	Não
Hung, Chou e Mak (2017)	Não informada	Grid de bins	Programação linear inteira	Legalização	ISPD 2015 (modificados)	Parcial*
Ren et al. (2007)	Não informada	Grid de bins	Difusão com fluxo de redes	Posicionamento Detalhado	Circuitos industriais (IBM) ISPD 2004	Não
Lee; Wu; Chiang (2010)	Não informada	Grid de bins**	Abacus	Legalização	ISPD 2005	Não
Zhu et al., (2018)	Não informada	Grid de Bins	MMSIM com fluxo de redes e refinamentos	Posicionamento Detalhado	ICCAD 2017	Não
Este trabalho	k-d tree	k-d tree	Abacus modificado	Legalização	ICCAD 2015 ICCAD 2017	Sim

* Paralelismo limitado ao número de bins por linha.

** Após espalhamento das células, partições em formatos retangulares ou cruz.

Tabela 1: Principais características dos trabalhos relacionados.

4 LEGALIZAÇÃO UTILIZANDO PARTICIONAMENTO

Este capítulo apresenta os principais aspectos da estrutura de dados espacial *k-d tree*. Inicialmente, na Seção 4.1 será apresentado o algoritmo Abacus modificado, o qual foi utilizado como caso de uso para a técnica proposta. Em seguida, na Seção 4.2 serão apresentados os principais aspectos das estruturas de dados espaciais consideradas na concepção deste trabalho. Na Seção 4.3 são apresentados os algoritmos de construção da *k-d tree*. Posteriormente na Seção 4.4 estão os requisitos necessários para utilizar o particionamento fornecido pela *k-d tree* na legalização. Finalmente, na Seção 4.5 está demonstrado como a *k-d tree* pode ser aplicada na legalização.

4.1 ABACUS MODIFICADO

Nesta seção é apresentado de forma geral o funcionamento do Algoritmo Abacus com as modificações necessárias para tratar células com altura de múltiplas linhas. Este algoritmo foi escolhido por se tratar de um algoritmo clássico, e portanto, muito conhecido na literatura.

O Algoritmo 1 recebe como entrada uma lista de células C e um conjunto de linhas R . Caso o circuito possua macroblocos é assumido que as linhas fornecidas ao algoritmo já foram segmentadas, e portanto, estão livres de sobreposições com macroblocos. Na Linha 1, as células são ordenadas em relação ao eixo x . Nas Linhas 2 a 14, para cada célula c_i do circuito é avaliado o custo caso esta seja posicionada em uma determinada linha r . Ao final de cada iteração, a célula é posicionada na linha que resultar em um menor custo (Linhas 12 e 13). É importante observar que a rotina `Place_Row` é invocada em versão temporária (Linha 7) e versão final (Linha 13). Essa rotina de posicionamento utiliza programação dinâmica e uma técnica de agrupamento que permite o movimento de um conjunto de células que já foram legalizadas.

Para considerar as restrições de células com altura de múltiplas linhas, mencionadas na Seção 2.2, uma modificação do Algoritmo 1 foi implementada. Nessa modificação o algoritmo original é chamado passando apenas células de mesma altura conforme o Algoritmo 2. Nas Linhas 1 e 2 as células são inseridas em uma lista bidimensional onde o primeiro índice é o tamanho em número de linhas da célula. As células desta lista são percorridas em ordem decrescente, onde as células são fixadas após a legalização, tornando necessária a atualização das linhas do circuito R (Linhas 4 a 8).

Algoritmo 1: Abacus

Input: Lista de células C , conjunto de linhas segmentadas do circuito R .

Output: Lista de células legalizadas C_{legal} .

```

1  $C \leftarrow \text{sort\_all\_cells\_according\_to\_x}()$ 
2 for  $i = 0$  to  $\text{size}(C)$  do
3    $c_i \leftarrow C[i]$ 
4    $\text{best\_cost} \leftarrow \infty$ 
5   foreach  $r \in R$  do
6      $\text{insert\_cell\_in\_row}(c_i, r)$ 
7      $\text{cost} \leftarrow \text{Place\_Row}(c_i, r)$  // Trial
8     if  $\text{cost} < \text{best\_cost}$  then
9        $\text{best\_cost} \leftarrow \text{cost}$ 
10       $\text{best\_r} \leftarrow r$ 
11     $\text{remove\_cell\_from\_row}(c_i, r)$ 
12   $\text{insert\_cell\_in\_row}(c_i, \text{best\_r})$ 
13   $C_{legal} \leftarrow \text{Place\_Row}(c_i, \text{best\_r})$  // Final
14   $i \leftarrow i + 1$ 
15 return  $C_{legal}$ 

```

Algoritmo 2: Abacus Modificado

Input: Lista de células C , conjunto de linhas segmentadas do circuito R .

Output: Lista de células legalizadas C_{legal} .

```

1 foreach  $c_i \in C$  do
2    $\text{cells\_by\_height}[h_{c_i}].\text{add}(c_i)$ 
3  $i \leftarrow \text{size}(\text{cells\_by\_height})$ 
4 while  $i \neq 0$  do
5   if  $\text{cells\_by\_height}[i].\text{not\_empty}()$  then
6      $C_{legal} \leftarrow \text{Abacus}(\text{cells\_by\_height}[i], R)$ 
7      $\text{fix\_cells}(C_{legal})$ 
8      $R \leftarrow \text{update\_segments}(C_{legal})$ 
9    $i \leftarrow i - 1$ 
10 return  $C_{legal}$ 

```

4.2 ESTRUTURAS DE DADOS ESPACIAIS APLICADAS À LEGALIZAÇÃO

Esta seção apresenta os principais aspectos de estruturas de dados espaciais que são potencialmente relevantes ao problema da legalização. Algumas estruturas de dados são apresentadas, assim como os motivos que levaram a adoção da *k-d tree* como a estrutura de dados utilizada para particionar o problema da legalização.

Estruturas de dados espaciais servem para gerenciar e representar dados multidimensionais que possuem informações geométricas e/ou posicionamento. Essas estruturas são frequentemente utilizadas em sistemas de informação geográficos, ferramentas de CAD e computação gráfica. Para este trabalho, tais estruturas são utilizadas para armazenar informações de posicionamento das células e também realizar o particionamento do problema. Dentre as principais estruturas de dados espaciais, as seguintes foram consideradas.

A *Quadtree* é formada por uma estrutura de árvore onde cada nó possui exatamente quatro filhos ou nenhum, (Figura 7). Esta estrutura subdivide regiões retangulares recursivamente em quatro quadrantes de mesmo tamanho. As informações armazenadas ficam registradas nas folhas da árvore, onde cada folha pode armazenar somente uma informação (*Point Quadtree*) ou um conjunto de informações (*Point-Region Quadtree*).

A *quadtree* não foi utilizada pois o número de elementos em cada partição é diretamente influenciado pela forma como estes estão distribuídos no espaço. É importante que o número de elementos em cada partição não seja muito diferente, caso contrário pode dificultar a distribuição da carga entre os elementos de processamento. Outro empecilho encontrado foi que a união de nós se torna mais complexa, pois para preservar a integridade da árvore quatro nós precisam ser combinados. A união de nós é útil para a legalização pois, caso alguma partição se torne muito densa, a formação de uma partição maior pode aumentar as chances de legalização.

A *Rtree* é uma estrutura de árvore formada por diversos retângulos onde estes agrupam elementos próximos. Esta estrutura é útil para realizar buscas por proximidade, pois é uma árvore balanceada onde a busca é feita verificando se um ponto está contido em um destes retângulos. O tamanho destes retângulos está restrito ao tamanho mínimo necessário para incluir determinados elementos sendo que a granularidade dos nós reduz da raiz até as folhas. O grau da árvore é arbitrário. Na Figura 8, cada nó possui no máximo três filhos.

A principal limitação da *Rtree* é que ela não mantém registro da área delimitada por cada nó e a legalização necessita dessa informação, sendo assim ela não pode ser utilizada para particionar o problema.

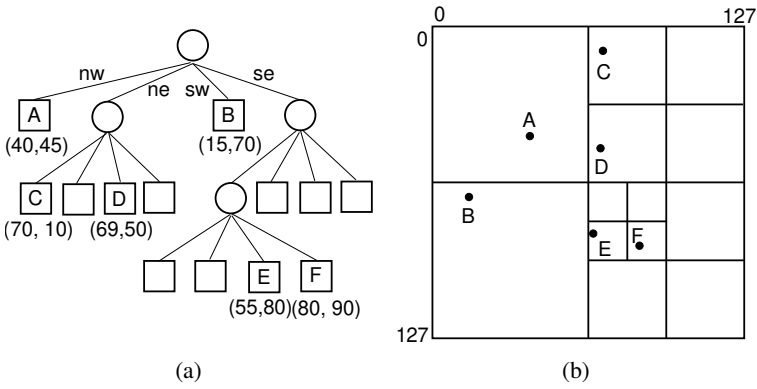


Figura 7: Exemplo de quadtree: (a) estrutura de árvore e (b) representação gráfica (SHAFFER, 2013).

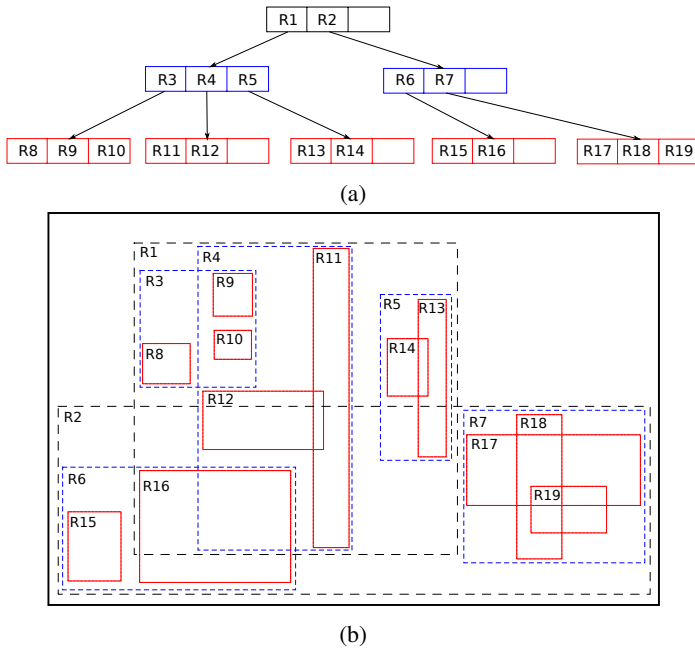


Figura 8: Exemplo de *Rtree*: (a) estrutura de árvore e (b) representação gráfica, adaptado de (WIKIPEDIA, 2018).

4.3 *K-D TREES* E SEUS ALGORITMOS

O tempo de execução da legalização pode ser reduzido com o uso de algoritmos paralelos. No entanto, o uso de paralelismo oferece três desafios: 1) espera-se que algoritmos da síntese física sejam determinísticos (sempre retornam a mesma saída para uma determinada entrada), o que potencialmente pode limitar o paralelismo da solução; 2) dependências de dados também estão presentes na legalização porque o posicionamento de uma célula precisa levar em consideração a posição das células que já foram legalizadas; e 3) soluções paralelas exigem mudanças em algoritmos consolidados, o que é propenso a erros, e portanto, não trivial.

A *k-d tree* é uma estrutura de dados em árvore binária, empregada para organizar um conjunto de pontos em um espaço k dimensional. Essa estrutura suporta inserção, deleção e buscas espaciais com complexidade computacional $\mathcal{O}(n)$ para uma *k-d tree* com n pontos (BENTLEY, 1975). A *k-d tree* pode ser construída recursivamente com complexidade $\mathcal{O}(n \log n)$. Em cada chamada recursiva, de acordo com uma determinada dimensão, o ponto mediano é selecionado para ser raiz. Um nó da árvore e os subconjuntos antes e após a mediana darão origem às subárvores a esquerda e direita respectivamente. A Figura 9 ilustra esse comportamento para um conjunto de dez pontos em um espaço bidimensional.

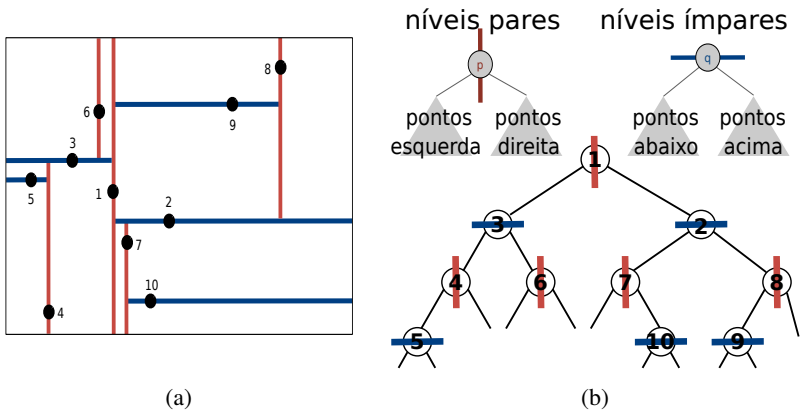


Figura 9: Uma *k-d tree* formada por pontos em um espaço bidimensional. Em (a), os pontos representam as posições dos elementos, e cada linha vermelha ou azul representa o posicionamento em relação a ao eixo dimensional x e y , respectivamente. Em (b), a *k-d tree* resultante de (a). Adaptado de (BENTLEY, 1975).

As *k-d trees* fornecem um particionamento natural para o circuito por quatro principais razões. Primeiro, como o ponto mediano é selecionado em cada nível da árvore e como a posição de todos elementos é conhecida a priori, a *k-d tree* resultante será sempre balanceada (mesmo número de elementos em cada partição). Segundo, nenhuma célula pode atravessar os limites do particionamento, sendo assim a legalização pode ser feita de forma independente para cada partição (apesar disso a solução se mantém determinística). Terceiro, o número de partições dobra a cada nível da árvore, tornando mais fácil o controle do número de elementos em cada partição. Quarto e último, se a legalização de uma partição não for possível, basta tentar novamente utilizando o nó pai da partição.

O algoritmo original da *k-d tree* foi modificado para se adequar às necessidades do problema por dois principais motivos: foi incluída a informação de área para cada subárvore e foi estabelecido um nível limite l_{limit} para a construção da árvore. O primeiro serve para que o algoritmo de legalização conheça as fronteiras de cada partição, e o segundo serve para limitar o número de partições. Isto também reduz o tempo de construção de $\mathcal{O}(n \log n)$ para $\mathcal{O}(n \times l)$. O algoritmo modificado para circuitos 2-D está representado pelos Algoritmos 3 e 4, detalhes relacionados ao algoritmo de legalização serão discutidos na Seção 4.4.

O Algoritmo 3 recebe como entrada uma lista de pontos (canto inferior esquerdo das células) P , a área do circuito a , e l_{limit} para limitar o número de partições (2^l partições). Cada nó da árvore inclui um ponto, a área da partição e os filhos a esquerda e direita (ou sublista de pontos para partição caso seja um nó folha). A função *first_axis()*, juntamente com a função *next_axis()* do Algoritmo 4, fornece a sequência de particionamento para construção da *k-d tree* (ex: eixo x, eixo y, eixo x, eixo y, etc., para um domínio 2-D).

Algoritmo 3: *kdtree_build*: construção *k-d tree* (modificado).

Input: Lista de pontos P , área do circuito a , limite de níveis l_{limit} .

Output: Nó raiz da *k-d tree* n_{root} .

```

1 /* eixo dimensional inicial para o particionamento          */
2 axis ← first_axis()
3 nroot ← kdtree(P,a,axis,1,llimit)
4 return nroot

```

O Algoritmo 4 recursivamente constrói a *k-d tree* e as partições. Além das informações dadas pelo Algoritmo 3, para realizar o particionamento também é necessário passar o nível atual da árvore l e o eixo dimensional $axis$. Após ordenar parcialmente P de acordo com o eixo dimensional $axis$ (Linha 2),

o ponto da mediana será a raiz da subárvore, dividindo os pontos a esquerda e a direita (P_{left} e P_{right} respectivamente). Na Linha 9 a área da partição será dividida para repassar essa informação para as próximas chamadas ao algoritmo. Esse processo será repetido até atingir o nível limite l_{limit} .

Algoritmo 4: *kdtree*: função recursiva de particionamento

Input: Lista de pontos P , área do circuito a , eixo atual $axis$, nível atual l , limite de níveis l_{limit} .

Output: k-d tree *node*.

```

1  $node_{area} \leftarrow a$ 
2 if  $l \leq l_{limit}$  then
3    $P \leftarrow nth\_element(P, \lfloor \frac{|P|}{2} \rfloor, axis)$ 
4    $m \leftarrow P[\lfloor \frac{|P|}{2} \rfloor]$  // ponto mediano de  $P$  parcialmente ordenado
5    $node_{points} \leftarrow m$ 
6    $P_{left} \leftarrow P[0 : \lfloor \frac{|P|}{2} \rfloor - 1]$  // pontos antes de  $m$ 
7    $P_{right} \leftarrow P[\lfloor \frac{|P|}{2} \rfloor + 1 : |P| - 1]$  // pontos após  $m$ 
8   /* divide área antes e após  $m$  */
9    $area_{left}, area_{right} \leftarrow split\_area(m, area, a)$ 
10  /* recursão para esquerda e direita */
11   $node_{left} \leftarrow kdtree(P_{left}, area_{left}, next\_axis(a), l + 1, l_{limit})$ 
12   $node_{right} \leftarrow$ 
     $kdtree(P_{right}, area_{right}, next\_axis(a), l + 1, l_{limit})$ 
13 else
14    $node_{points} \leftarrow P$  // nó folha
15 return  $node$ 
```

O sobrecusto de construção da *k-d tree* pode ser facilmente superado com os benefícios que o particionamento fornece, por exemplo, redução do tamanho da entrada para o algoritmo de legalização e a possibilidade de uso de algoritmos paralelos.

4.4 REQUISITOS ADICIONAIS PARA LEGALIZAÇÃO

Para empregar o uso de *k-d trees* no particionamento do circuito, é necessário realizar um pré-processamento nos dados do circuito. Esta seção descreve os dois passos necessários para que a *k-d tree* seja utilizada de forma efetiva.

Todas as sobreposições entre células e macroblocos precisam ser removidas antes da construção da *k-d tree*, caso contrário o deslocamento

máximo pode aumentar desnecessariamente, pois o particionamento pode restringir a direção de remoção de sobreposição. Isto está ilustrado pela Figura 10 (a), onde a célula verde $c1$ sobrepõe o macrobloco azul. Isto ocorre porque, após o particionamento, as células não podem ser movidas para outra partição, caso contrário, as partições não poderiam ser legalizadas de forma independente. Nesta situação, a direção de remoção da sobreposição que resulta no menor deslocamento seria vertical. Porém, este movimento não será possível após o particionamento, sendo assim, a célula seria movida para o retângulo vermelho aumentando o deslocamento da solução.

A remoção de sobreposições não resolve algumas situações onde uma célula é assinalada a um lugar estreito (onde não há espaço suficiente para a célula). Esta situação está ilustrada na Figura 10 (b). A célula vermelha $c2$ foi movida para fora do macrobloco, porém ainda não cabe dentro da Partição 1. Como consequência, esta célula vai acabar sendo movida para o retângulo tracejado vermelho, resultando em um alto deslocamento.

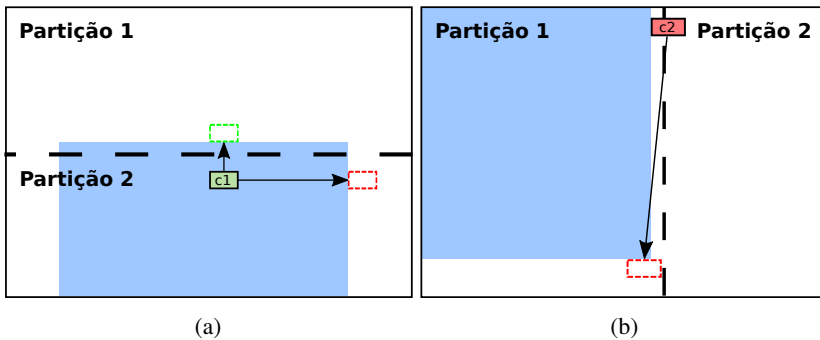


Figura 10: As linhas tracejadas dividem o circuito em duas partições. A célula verde $c1$ sobrepõe o macrobloco azul, enquanto que a célula vermelha $c2$ está com falta de espaço na Partição 1, sendo assim é movida para baixo do macrobloco na posição ilustrada pelo retângulo vermelho.

Para tratar esses casos, ao final da legalização, um refinamento da solução é realizado. Este refinamento legaliza sequencialmente somente as k^1 células que ficaram com maior deslocamento. Nesta legalização sequencial, com exceção do conjunto das k células, todas as demais células do circuito são tratadas como fixas.

A Figura 11 mostra o que acontece quando as células em magenta e azul não estão alinhadas antes da construção da k - d tree. As fronteiras do

¹ Neste trabalho, este parâmetro foi ajustado para $k = 1000$

particionamento podem atravessar os *sites*, dando origem a um *site* incompleto e portanto sem utilidade para o posicionamento de células. Portanto, para evitar esses tipos de problemas, antes da construção da árvore é realizado um pré-processamento para alinhar as células.

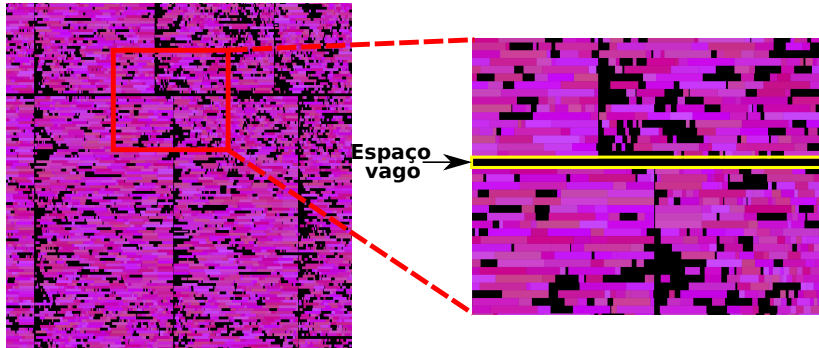


Figura 11: Efeito do desalinhamento das células antes do particionamento. Os retângulos em magenta são células do circuito, o fundo preto ilustra o espaço vago desperdiçado.

4.5 LEGALIZAÇÃO UTILIZANDO K - D TREES

Para possibilitar a legalização utilizando o Algoritmo 5 foram combinadas as etapas de pré-processamento apresentadas na Seção 4.4 e os algoritmos apresentados na Seção 4.3. Este algoritmo recebe como entrada informações do circuito (incluindo sua área, macroblocos e uma lista de células que serão legalizadas), o algoritmo de legalização L , e o número de níveis a serem criados pela k - d tree l e retorna uma lista de posições legalizadas para as células juntamente com o resultado da legalização total do circuito. O algoritmo remove as sobreposições e alinha as células (Linhas 3 e 4) antes de construir a k - d tree. Após a construção da árvore, o algoritmo de legalização é chamado (Linha 8) utilizando a k - d tree conforme descrito no Algoritmo 6.

O Algoritmo 6 utiliza a estrutura topológica de árvore da k - d tree para guiar a legalização. Se for passado por parâmetro um nó folha (Linha 1), então será chamado o algoritmo de legalização L (Linha 3). O algoritmo L recebe uma lista de pontos, área da partição e informações do circuito (como posições ocupadas por macroblocos e células fixas). Caso o nó passado por parâmetro não seja folha, então será legalizada somente a célula do nó (Linha 6) e em seguida, em paralelo, será chamada a legalização para as subárvores à esquerda e direita (Linhas 10 e 11). Se a legalização das subárvores for bem sucedida,

Algoritmo 5: Legalização utilizando *k-d trees*.

Input: Informações do circuito $circuit_{info}$, algoritmo de legalização L , nível limite l_{limit} .

Output: Lista de células legalizadas C_{legal} , booleano $success$.

```

1 /* Pré processamento */
2  $P \leftarrow circuit_{cells}$ 
3  $P \leftarrow remove\_overlaps(P, circuit_{info})$ 
4  $P \leftarrow align\_cells(P, circuit_{info})$ 
5 /* Construção k-d tree */
6  $n_{root} \leftarrow kdtree\_build(P, circuit_{area}, l)$ 
7 /* Legalização utilizando k-d tree */
8  $C_{legal}, success \leftarrow kdtree\_legalize(L, n_{root}, circuit_{info}, \emptyset)$ 
9 return  $C, success$ 

```

o algoritmo retornará a concatenação das listas de pontos legalizados das subárvores, caso contrário ele tentará legalizar todos os pontos das subárvores (Linha 18).

A Figura 12 ilustra uma simples execução do Algoritmo 6 para um circuito com onze células (A a K) e $l = 2$. A *k-d tree* com dois níveis (ilustrada pela Figura 12) é fornecida para o algoritmo que vai legalizar e fixar a célula A e então executar em paralelo as subárvores a esquerda e direita, B e C , respectivamente. O algoritmo vai legalizar e fixar a célula B , e então executar em paralelo suas subárvores. Como ambos são folhas, o algoritmo de legalização vai ser chamado para as Partições 1 e 2, e seus resultados serão combinados para retornar a subárvore com a célula B como raiz. O mesmo será feito com a subárvore com raiz C , resultando na completa legalização do circuito. Finalmente, é possível notar que não é permitido que parte de uma célula fique fora de sua partição, e que todas as células não folha (células brancas na Figura 12 (a)) já foram legalizadas antes da legalização das partições, tornando possível a paralelização da solução.

Algoritmo 6: *kdtree_legalize*: Legalização recursiva das partições.

Input: Algoritmo de legalização L , k - d tree node, informações do circuito $circuit_{info}$, lista de posições fixas F .

Output: Lista de células legalizadas C_{legal} , booleano *success*.

```

1 if is_leaf(node) then
2   | /* Legalizar as células da partição */
3   |  $C, success \leftarrow L(node_{cells}, no_{area}, circuito, F)$ 
4 else
5   | /* Legalizar e fixar a célula da partição */
6   |  $C_{root}, success_{root} \leftarrow$ 
7   |    $L(node_{ponto}, node_{area}, circuito, F)$ 
7   | if success_root then
8   |   | /* Legalizar partições em paralelo */
9   |   | run in pallel
10  |   |   |  $C_l, success_l \leftarrow$ 
11  |   |   |   |  $kdtree\_legalize(node_l, circuito, F + C_{root})$ 
12  |   |   |   |  $C_r, success_r \leftarrow$ 
13  |   |   |   |  $kdtree\_legalize(node_r, circuito, F + C_{root})$ 
14  |   |   | /* Checar se os nós filhos foram legalizados */
15  |   |   | if success_l e success_r then
16  |   |   |   | /* Concatenar soluções dos filhos */
17  |   |   |   |  $C, success \leftarrow C_l + C_r + C_{root}, success$ 
18  |   |   | else
19  |   |   |   | /* Legalizar partição do nível atual */
20  |   |   |   |  $C, success \leftarrow L(node_{points}, node_{area}, circuito, F)$ 
21  |   | else
22  |   |   |  $C, success \leftarrow \emptyset, false$  // Falha ao legalizar a célula
23  |   |   | raiz
24 return  $C, success$ 

```

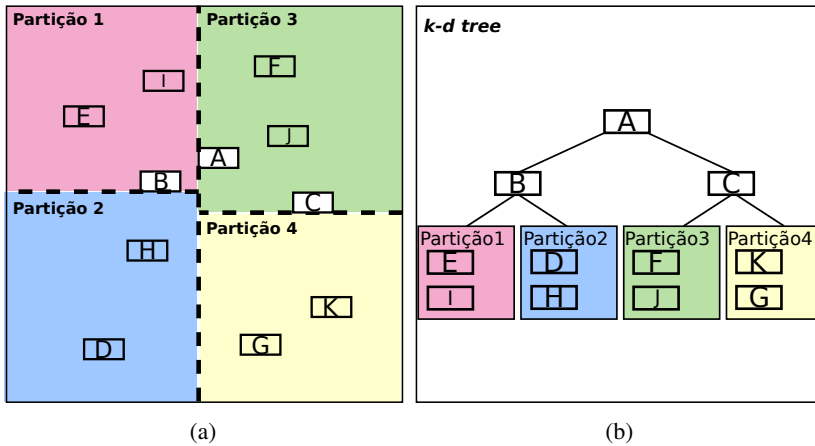


Figura 12: (a): As células brancas são as células fixas e as linhas tracejadas representam as partições. (b): A *k-d tree* do particionamento do circuito.

5 AMBIENTE EXPERIMENTAL

Este capítulo descreve o ambiente experimental utilizado para a validação do esquema de particionamento de circuitos proposto e para a avaliação de sua qualidade e desempenho. A Seção 5.1 apresenta os principais objetivos e métricas dos experimentos realizados e os efeitos da técnica de particionamento no algoritmo de legalização. Na Seção 5.2 são detalhados os conjuntos de *benchmarks* utilizados para testar a técnica proposta bem como o motivo da escolha destes circuitos. A Seção 5.3 descreve a forma como os experimentos foram executados bem como os métodos estatísticos utilizados para os resultados. A Seção 5.4 detalha a infraestrutura experimental, informações de compilação e detalhes técnicos de implementação.

5.1 OBJETIVOS E MÉTRICAS

Os experimentos realizados têm como principal objetivo demonstrar que o uso da técnica pode reduzir o tempo de execução da legalização. Isso ocorre através do particionamento do problema que viabiliza soluções paralelas e reduz o tamanho da entrada fornecida ao algoritmo. Também é necessário avaliar se a técnica degrada a qualidade da solução e, se for o caso, o quanto de degradação ocorre. Portanto, métricas de deslocamento médio de células e o deslocamento máximo foram monitoradas. Adicionalmente, o *Half-Perimeter Wire Length* (HPWL)¹ também foi registrado para verificar o impacto da técnica na etapa de roteamento.

5.2 BENCHMARKS

A técnica de particionamento proposta foi avaliada utilizando os *benchmarks* das competições *ICCAD2015 CAD contest* (Problema C: posicionamento incremental guiado por atraso (KIM et al., 2015)) e *ICCAD2017 CAD Contest* (legalização de células com alturas de múltiplas linhas) (DARAV et al., 2017), todos derivados de formatos industriais.

- Os circuitos da competição **ICCAD2015** foram escolhidos devido ao grande número de células presentes. Os circuitos contêm de 760 mil células até 1,93 milhão de células. Nenhuma das células possui uma altura de múltiplas linhas e elas não incluem *fence regions*. Como esses circuitos já estavam legalizados, foi necessário aplicar um vetor de

¹ HPWL é a metade do perímetro formado pelo retângulo mínimo que contém todos os pinos da interconexão.

movimentos aleatórios em cada célula móvel utilizando uma distribuição uniforme de $-10k$ à $+10k$ microns. Informações detalhadas sobre esses circuitos estão registradas na Tabela 2.

- Os circuitos da competição **ICCAD2017** foram escolhidos devido à sua complexidade. Eles são circuitos pequenos contendo no máximo 130 mil células. Porém, eles possuem células com altura de múltiplas linhas e *fence regions*. Esses circuitos foram utilizados para demonstrar que a técnica também pode ser aplicada em circuitos mais complexos que consideram as restrições apresentadas no Capítulo 2. Informações detalhadas sobre esses circuitos estão registradas na Tabela 3.

5.3 EXECUÇÃO DOS EXPERIMENTOS

O algoritmo de legalização escolhido como caso de uso é uma modificação do Abacus (SPINDLER; SCHLICHTMANN; JOHANNES, 2008). Sabendo que a complexidade do Abacus é $\mathcal{O}(n^2)$, o uso de *k-d trees* para particionar o circuito em partições menores pode reduzir o tempo de execução para $\frac{n^2}{2l}$ para n células e l níveis da árvore. No entanto, esta mudança no tempo de execução não está levando em conta o sobrecusto de criação e gerenciamento da *k-d tree* (que deve aumentar o tempo total de execução) e a paralelização (que deve reduzir o tempo de execução).

A técnica de particionamento não foi aplicada às células das *fence regions*, pois o tamanho do problema não justifica o particionamento. O circuito *des_perf_b_md1* ficou fora dos experimentos, pois, devido a alta densidade de uma de suas *fence regions*, a técnica modificada do Abacus não foi capaz de realizar a legalização.

Os tempos de execução apresentados neste trabalho são a média aritmética de dez execuções, sendo todos os tipos de métricas avaliadas determinísticas com exceção do tempo de execução. Todos os tempos de execução registrados incluem o pré-processamento mencionado na Seção 4.4 e descrito pelo Algoritmo 5. Para facilitar a compreensão dos resultados, é importante que fique claro a forma como estes estão divididos:

- **Sequencial (sem particionamento):** versão normal da solução sem realizar particionamento e em sequencial, equivalente a execução do algoritmo de legalização onde a *k-d tree* possui só um nível e portanto só um nó raiz.
- Com **Particionamento:** versão que utiliza a *k-d tree* para particionar o circuito, reduzindo o tamanho da entrada e portanto o tempo de execução,

porém essa não faz uso de paralelismo ou seja toda a execução é feita em uma única unidade de processamento.

- Em **paralelo**: também foi utilizada a *k-d tree* para particionamento do circuito e adicionalmente se beneficia da remoção da dependência de dados entre as partições para resolver o problema em paralelo.

5.4 INFRAESTRUTURA EXPERIMENTAL

Os experimentos foram executados em uma máquina com a seguinte especificação: Sistema operacional Arch Linux, kernel 4.19.8-arch1-1-ARCH com processador Intel[®] Xeon[®] E5430 4 cores 2.667 GHz, 16GB RAM (8 × 2GB DDR2/667 MHz quad channel). O código fonte foi feito na linguagem C++ e compilado utilizando gcc versão 8.2.1 com ‘-O3’ como *flag* de otimização e todas as bibliotecas foram incluídas estaticamente. Essas informações servem para que o compilador realize diversas otimizações no código de forma automática. O código da *k-d tree* foi implementado utilizando os modelos geométricos da biblioteca Boost 1.68.0-2. Foi utilizada a API de programação paralela OpenMP (DAGUM; MENON, 1998) que através de diretivas de compilação gera um programa paralelo utilizando memória compartilhada para a sincronização da execução. Os resultados preliminares foram obtidos utilizando 4 unidades de processamento (uma por núcleo). Para reduzir o ruído no tempo de execução durante os experimentos, as *threads* foram fixadas aos núcleos utilizando a API HWLOC (BROQUEDIS et al., 2010) versão 1.11. Todo código fonte utilizado para a execução dos experimentos, resultados em formato csv e código para geração dos gráficos estão disponíveis em (GITHUB/ECLUFSC, 2019).

Circuito	#Linhas	#Macroblocos	#Células	#Interconexões	Densidade
superblue18	1790	181	744947	771542	65.41
superblue4	1842	290	756979	802513	74.52
superblue16	1790	91	981140	999902	73.84
superblue5	2530	669	1014341	1100825	69.83
superblue1	1831	395	1159346	1215710	75.10
superblue3	1842	518	1160765	1224979	74.52
superblue10	3439	1628	1786523	1898119	75.54
superblue7	3165	376	1865884	1933945	74.67

Tabela 2: Informações sobre os circuitos da competição *ICCAD2015 CAD Contest*.

Circuito	#Linhas	#Macroblocos	#Células	#Interconexões	#Fence regiões	Altura em linhas				Densidade
						#1	#2	#3	#4	
des_perf_1*	222	0	112644	112878	0	100	0	0	0	90.37
des_perf_a_md1*	450	4	108288	110281	4	95.66	4.34	0	0	77.73
des_perf_a_md2	450	4	108288	110281	4	96.99	1	1	1	78.13
des_perf_b_md1	300	0	112679	122951	12	94.8	5.20	0.00	0.00	54.98
des_perf_b_md2	300	0	112679	122951	12	90.47	6.02	2.01	1.50	64.69
edit_dist_1_md1	361	0	130661	133223	0	90.31	6.12	2.04	1.53	67.47
edit_dist_a_md2	400	6	127414	134051	1	90.31	6.12	2.04	1.53	59.42
edit_dist_a_md3*	400	6	127413	131134	1	93.88	2.04	2.04	2.04	69.75
fft_2_md2	171	0	32281	33307	0	89.62	6.56	2.18	1.64	83.12
fft_a_md2	400	6	30625	32090	0	89.57	6.59	2.19	1.65	32.41
fft_a_md3	400	6	30625	32090	0	93.42	2.19	2.19	2.19	31.24
pci_bridge32_a_md1	200	4	29533	34058	3	90.39	6.07	2.02	1.52	49.57
pci_bridge32_a_md2*	200	4	29517	29985	4	85.51	7.08	4.05	3.37	66.23
pci_bridge32_b_md1*	400	6	28914	29417	3	90.39	6.07	2.02	1.52	58.16
pci_bridge32_b_md2*	400	6	28914	29417	3	96.97	1.01	1.01	1.01	53.45
pci_bridge32_b_md3*	400	6	28914	29417	3	94.94	1.01	2.02	2.02	55.69

* Circuito oculto revelado somente após a competição.

** Nos circuitos ocultos o cálculo da densidade leva em consideração a área ocupada por macroblocos.

Tabela 3: Informações sobre os circuitos da competição *ICCAD2017 CAD Contest*, adaptado de (DARAV et al., 2017).

6 RESULTADOS

Este capítulo apresenta os principais resultados alcançados durante os experimentos com a técnica proposta de particionamento de circuitos para a legalização. Nas Seções 6.1 e 6.2 estão os resultados obtidos com os circuitos das competições ICCAD2015 e ICCAD2017, respectivamente (circuitos detalhados na Seção 5.2).

6.1 RESULTADOS ICCAD2015

A Figura 13 apresenta a *speedup*¹ da versão paralela com particionamento sobre a versão sequencial sem particionamento. O *speedup* reflete a aceleração alcançada com a técnica de particionamento proposta juntamente com a execução em paralelo, neste caso, utilizando quatro *threads*. O número inicial de partições utilizadas varia de 1 a 8192, onde cada linha representa os resultados de um determinado circuito. Maiores valores representam menores e melhores tempos de execução, e valores iguais a 1 indicam que os tempos de execução paralelo e sequencial são iguais.

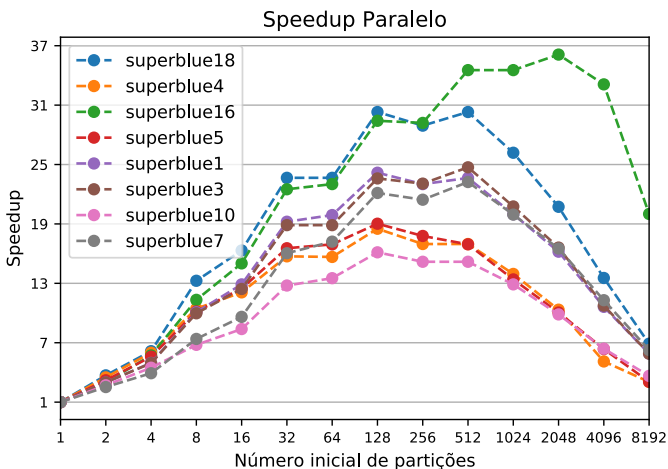


Figura 13: *Speedup* da versão paralela (eixo y) ao variar o número inicial de partições (eixo x) para os circuitos da competição ICCAD2015.

¹ *speedup*: fator de aceleração, razão entre o tempo de execução da versão sequencial sobre o tempo de execução da versão paralela.

A redução do tempo de execução do algoritmo original é causada pela redução do tamanho da entrada utilizando particionamento, execução paralela, ganhos relacionados a localidade da memória e a redução do tamanho da linha a ser considerada pelo Abacus. A saturação do *speedup* ocorre para mais de 128 partições, onde ocorre o aumento do número de combinações necessárias entre partições (Tabela 4), e portanto retrabalho para a legalização. As combinações das partições são um reflexo da dificuldade de legalização, ou seja, elas são influenciadas pela densidade das partições, números de macroblocos e a forma como eles estão distribuídos.

Circuitos	Número inicial de Partições													
	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
superblue18	0	0	0	0	0	0	0	0	0	0	3	23	75	614
superblue4	0	0	0	0	0	0	0	0	0	1	11	118	346	1740
superblue16	0	0	0	0	0	0	0	0	0	0	0	5	23	344
superblue5	0	0	0	0	0	0	0	0	0	1	3	22	82	520
superblue1	0	0	0	0	0	0	0	0	0	0	0	7	42	309
superblue3	0	0	0	0	0	0	0	0	0	0	4	25	96	543
superblue10	0	0	0	0	0	0	0	0	1	5	20	65	184	598
superblue7	0	0	0	0	0	0	0	0	0	0	13	43	132	815

Tabela 4: Número de chamadas a função de união de partições.

A Figura 14 apresenta a aceleração fornecida pela execução paralela, calculado pela razão entre o tempo de execução da versão sequencial com particionamento dividido pela execução em paralelo com particionamento. É importante lembrar que na execução paralela foram utilizadas quatro *threads*, sendo assim, quatro seria o *speedup* ideal. Porém, esse valor não é alcançado, pois nem toda execução pode ser realizada de forma paralela. Por exemplo, existe uma sincronização necessária ao final das iterações do Algoritmo 6 Linha 13. Adicionalmente, os sobrecustos de processamento e criação da árvore também estão inclusos na versão paralela (Seção 4.3 e 4.4).

Na Figura 15 estão ilustrados os tempos de execução médios obtidos com a versão paralela com particionamento, onde essa acompanha o mesmo comportamento da Figura 13 em relação ao aumento do número inicial de partições. O circuito *superblue16* continua reduzindo o tempo de execução até 2048 partições, pois ele é o circuito com o menor número de macroblocos, o que influencia no número de chamadas à função de união de partições. Mesmo sem realizar uniões de partições, o tempo de execução do circuito *superblue4* aumenta antes do tempo de execução para o circuito *superblue18*, pois os macroblocos também aumentam o número de sub-linhas que o algoritmo de legalização precisa avaliar. A redução do tempo de execução mais significativa foi de 25 minutos para menos de 1 minuto utilizando mais de 128 partições no circuito *superblue7*.

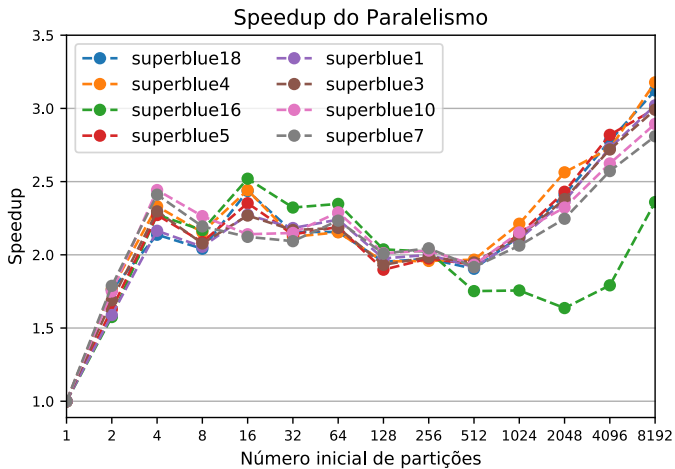


Figura 14: *Speedup* (eixo y) da execução paralela em relação a versão sequencial com particionamento ao variar o número inicial de partições de 1 a 8192 (eixo x) para os circuitos da competição ICCAD2015.

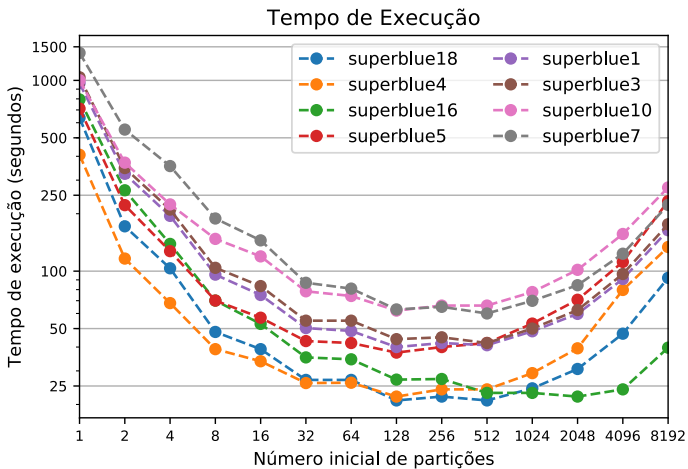


Figura 15: No eixo y em escala logarítmica, o tempo de execução em segundos (quanto menor melhor), e no eixo x a variação do número inicial de partições em potências de base 2 (2^l) de 1 a 8192.

A Figura 16 demonstra o *speedup* da versão sequencial com particionamento sobre a versão sequencial sem particionamento. Vale lembrar que isto ocorre pois o tamanho da entrada fornecida ao algoritmo de legalização Abacus é reduzido com o aumento do número de partições. Portanto, mesmo executando sequencialmente ainda é possível reduzir o tempo de execução utilizando a técnica de particionamento proposta. A saturação do *speedup* também ocorre após 256 pelo aumento retrabalho da legalização.

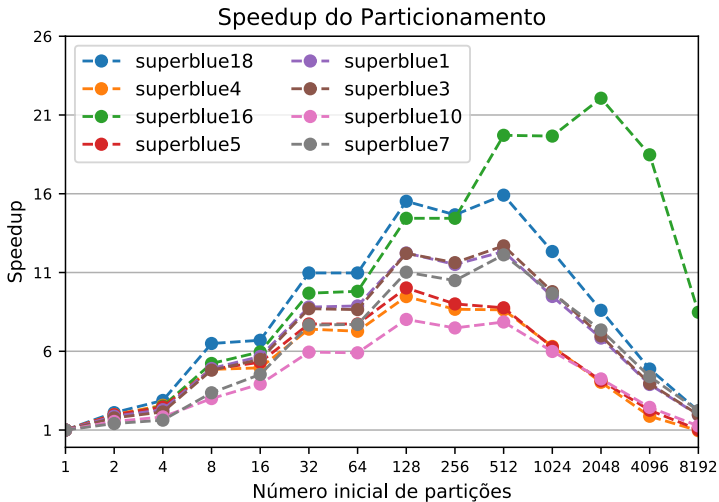


Figura 16: No eixo y o *Speedup* da versão com particionamento em relação a versão sequencial (quanto maior melhor), e no eixo x a variação do número inicial de partições em potências de base 2 (1 a 8192 partições).

A respeito da qualidade da solução, na Figura 17 está ilustrada a melhoria no deslocamento médio das células para diferentes números iniciais de partições. Para um dado circuito e nível l , a melhoria no deslocamento médio é computada como a razão entre o deslocamento médio das células provocado pela versão sequencial sobre a versão paralela que utiliza particionamento. Pode-se notar que o deslocamento médio para todos os circuitos foi reduzido em todos os casos com até 64 partições (valores acima de 1) e para a maioria dos casos para qualquer quantidade de partições testadas. Para os menores circuitos *superblue18* e *superblue4* o efeito do particionamento se torna mais relevante, pois com o aumento do particionamento a solução se torna mais limitada.

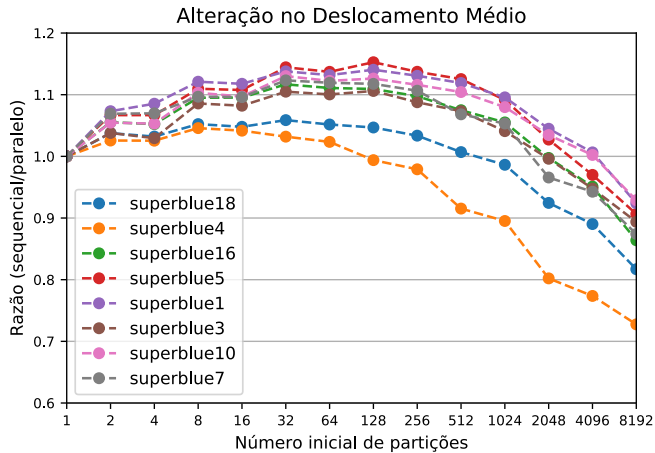


Figura 17: Melhoria no deslocamento médio das células (quanto maior melhor, 1 é indiferente). O eixo x representa o número inicial de partições 2^l e o y a razão entre a versão sem particionamento sobre a versão com particionamento.

A Figura 18 apresenta a alteração no deslocamento máximo para cada circuito ao variar o número inicial de partições (versão sem particionamento sobre versão com particionamento). Utilizar métodos que buscam soluções locais (como é o caso do particionamento) em circuitos que possuem macroblocos ou alta densidade pode aumentar o deslocamento máximo (KARIMPOUR DARAV et al., 2017). Por este motivo, e para tratar situações como a apresentada pela Figura 10 (b), foi utilizado o refinamento proposto na Seção 4.4. Para poucas partições os circuitos *superblue1* e *superblue16* apresentaram uma melhoria no deslocamento máximo. Nesses casos, o particionamento reduziu o congestionamento alterando a distribuição e ordem em que as células são legalizadas. As melhorias no deslocamento para muitas partições (*superblue3* e *superblue5*), foram causadas pelas uniões de partições replicando o comportamento do uso de poucas partições em regiões congestionadas. É importante notar que as uniões de partições são feitas em relação ao nó pai da partição, ou seja o número de células cresce exponencialmente a cada chamada da função união para um mesmo ramo da árvore.

Sobre o HPWL, métrica importante para a etapa de roteamento, para o pior caso foi observado um aumento de menos de 4% e na média entre todos circuitos menos de 2%.

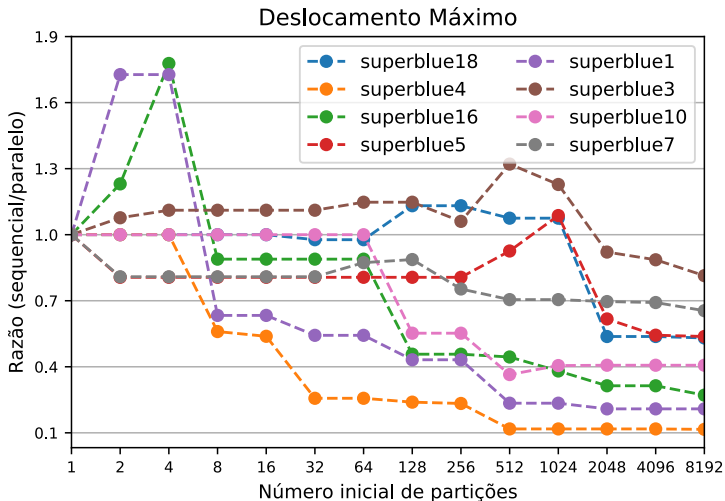


Figura 18: Melhoria no deslocamento máximo da solução (quanto maior melhor, 1 é indiferente). O eixo x representa o número inicial de partições 2^l e o y a razão entre a versão sem particionamento sobre a versão com particionamento.

6.2 RESULTADOS ICCAD2017

Estes circuitos são menores que os apresentados anteriormente, porém possuem mais tipos de restrições tecnológicas como *fence regions* e células de alturas de múltiplas linhas. Todos os tempos de execução também incluem a legalização das *fence regions*. Na Figura 19, é possível verificar que os circuitos maiores e mais densos levam mais tempo para serem legalizados (*des_perf_1*, *des_perf_a_md1* e *md2*). Sendo assim, o algoritmo precisa avaliar mais linhas para encontrar um posicionamento legal para todas as células. O circuito *fft_2_md2* também possui alta densidade, porém duas ordens de grandeza menos células em relação aos circuitos citados anteriormente. Portanto, o tempo de legalização é dominado não apenas pelo número de células mas também pela densidade do circuito. Os tempos de execução também demonstram que, para o caso do Abacus modificado, o uso de células com diferentes alturas não afeta de forma significativa o tempo de execução, uma vez que este trabalha apenas com um tamanho de célula por vez.

Sobre a qualidade da solução, na Figura 20 é possível verificar que apenas os circuitos *des_perf_1* e *pci_bridge32_b_md1* apresentaram uma

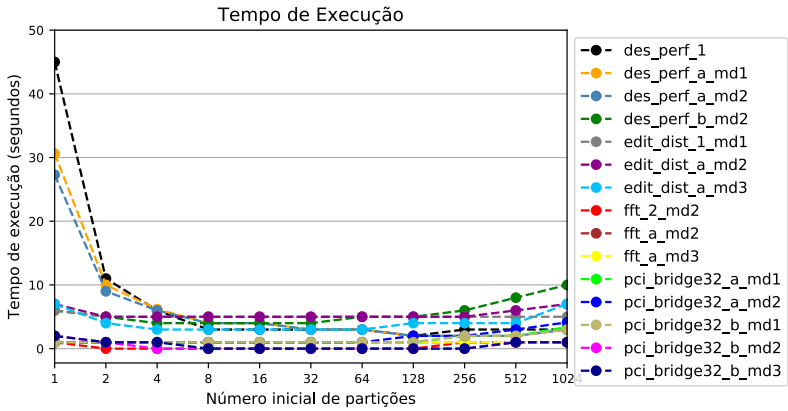


Figura 19: No eixo y o tempo de execução em segundos (quanto menor melhor), e no eixo x a variação do número inicial de partições em potências de base 2 (2^l).

deterioração significativa no deslocamento médio. Isso ocorre pois o primeiro circuito apresenta uma alta densidade (90%), e o segundo circuito possui células muito próximas entre macroblocos, tais fatores dificultam o particionamento.

O deslocamento máximo, ilustrado na Figura 21, também não demonstrou melhorias, pois os circuitos são muito pequenos (no mínimo 25 vezes menores que os do ICCAD2015). Sendo assim é esperado um comportamento similar aos circuitos do ICCAD2015 para mais de 16 partições: uma tendência ao aumento do deslocamento máximo. Portanto, isso não justifica o uso de particionamento, pois neste caso acaba restringindo o espaço de solução.

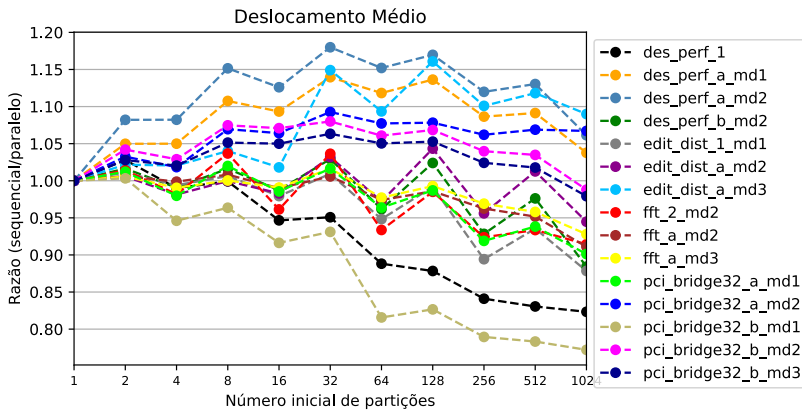


Figura 20: Alteração no deslocamento médio das células (quanto maior melhor, 1 é indiferente). O eixo x representa o número inicial de partições 2^l e o y a razão entre a versão sequencial sobre a paralela.

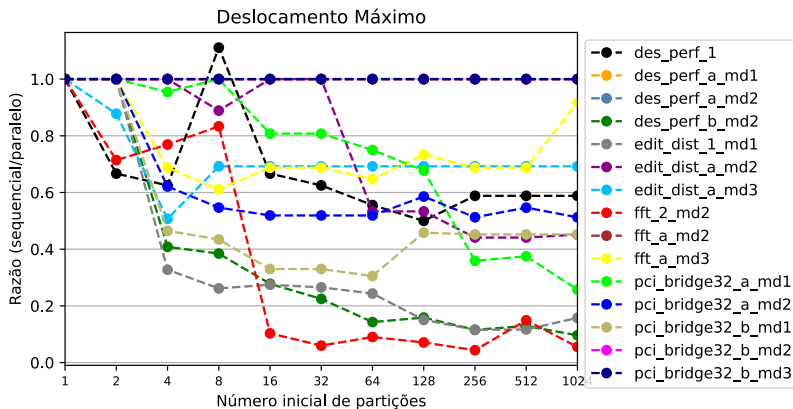


Figura 21: Alteração no deslocamento máximo da solução (quanto maior melhor, 1 é indiferente). O eixo x representa o número inicial de partições 2^l e o y a razão entre a versão sequencial sobre a paralela com *k-d tree*.

7 CONCLUSÕES E TRABALHOS FUTUROS

7.1 CONCLUSÃO

O aumento do número de células, o uso de *fence regions*, macroblocos e bibliotecas com células de múltiplas alturas tornam a legalização de circuitos integrados desafiadora. Essas e outras questões exigem algoritmos de legalização mais complexos capazes de tratar tais restrições. No entanto, esses algoritmos tendem a apresentar maiores tempos de execução. Para tratar objetivos conflitantes como baixo tempo de execução e alta qualidade da solução, diversos trabalhos apresentados na Seção 3.2 utilizam particionamento para tornar viável a legalização utilizando algoritmos mais complexos. Porém, todos estes trabalhos desconsideram o uso de uma estrutura de dados espacial para realizar tal particionamento, sendo necessário o uso de algoritmos auxiliares como, por exemplo, algoritmos de fluxo em redes para distribuir as células de partições congestionadas.

Neste trabalho foram propostas e implementadas uma estrutura de dados *k-d tree* e suas adaptações necessárias para particionar o circuito durante a legalização. Dentre as adaptações, foram incluídas informações de área e o pré-processamento apresentado na Seção 4.4. Para tratar os casos quando uma partição muito densa não pode ser legalizada, foi desenvolvido um mecanismo de união de partições onde, a partir do nó pai de uma partição, uma nova é formada da união de seus filhos. Para validar a técnica de particionamento proposta, foram realizados experimentos com circuitos das competições ICCAD 2015 e 2017.

Em todos os casos a abordagem foi capaz de reduzir o tempo de execução em relação a legalização completa do circuito sem utilizar particionamento. Para os circuitos da competição ICCAD 2015, a aceleração alcançada foi de 15 a 36 vezes. Para circuitos com restrições modernas (ICCAD 2017) como células de múltiplas alturas, macroblocos e *fence regions* a técnica se mostrou aplicável e capaz de reduzir o tempo de execução em relação a versão sequencial sem particionamento, por exemplo, em até 22 vezes para o circuito "des_perf_1" com 128 partições. Esses resultados vêm da capacidade da *k-d tree* em controlar a granularidade do particionamento e remover parte da dependência de dados viabilizando o paralelismo da solução. Entretanto, o particionamento deve ser utilizado com moderação, pois foram encontrados casos onde a redução do tamanho das partições reduziu as possibilidades de solução e, conseqüentemente, a união de partições e o retrabalho aumentaram o tempo de execução.

Apesar do particionamento restringir o espaço de solução, o algoritmo de legalização Abacus utilizado foi capaz de se beneficiar da alteração no

ordenamento das células provocando uma melhoria no deslocamento médio da solução. Para os circuitos do ICCAD 2015 e 2017 a melhoria foi de 5% a 15% e de 1% a 17%, respectivamente. A redução do tempo de execução pode ser utilizada para realizar refinamentos na solução. Isso pode ser feito utilizando algoritmos mais complexos em uma pequena região do circuito ou até mesmo explorar diferentes opções de legalização e escolher dentre elas a que obteve melhor qualidade.

7.2 TRABALHOS FUTUROS

Como trabalho futuro, seria de interesse verificar o impacto do uso de outro algoritmo de legalização mais complexo que o algoritmo utilizado (Abacus modificado, Seção 4.1). Para alcançar melhores resultados a respeito da qualidade da solução as seguintes questões precisam ser investigadas:

- Desenvolver um mecanismo para identificar de forma automática o número ideal de partições para um dado circuito.
- Implementar um mecanismo de tomada de decisão para escolher dentre o particionamento horizontal ou vertical durante a construção da árvore.
- Utilizar a redução do tempo de execução para refinar a qualidade da legalização utilizando algum algoritmo de posicionamento detalhado.

BIBLIOGRAFIA

ALPERT, Charles et al. Placement: Hot or not? In: ACM. PROCEEDINGS of the International Conference on Computer-Aided Design. New York, NY, United States, 2012. p. 283–290.

BENTLEY, Jon Louis. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, ACM, New York, NY, United States, v. 18, n. 9, p. 509–517, 1975.

BRENNER, Ulrich. VLSI legalization with minimum perturbation by iterative augmentation. In: EDA CONSORTIUM. PROCEEDINGS of the Conference on Design, Automation and Test in Europe. San Jose, CA, United States, 2012. p. 1385–1390.

BROQUEDIS, François et al. hwloc: A generic framework for managing hardware affinities in HPC applications. In: IEEE. PARALLEL, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on. Piscataway, NJ, United States, 2010. p. 180–186.

CHEN, Jianli et al. Toward optimal legalization for mixed-cell-height circuit designs. In: ACM. PROCEEDINGS of the 54th Annual Design Automation Conference 2017. New York, NY, United States, 2017. p. 52.

DAGUM, Leonardo; MENON, Ramesh. OpenMP: an industry standard API for shared-memory programming. **IEEE computational science and engineering**, IEEE, Piscataway, NJ, United States, v. 5, n. 1, p. 46–55, 1998.

DARAV, Nima Karimpour et al. ICCAD-2017 CAD contest in multi-deck standard cell legalization and benchmarks. In: IEEE PRESS. PROCEEDINGS of the 36th International Conference on Computer-Aided Design. 2017. p. 867–871.

GITHUB/ECLUFSC, 2019.

HILL, Dwight. **Method and system for high speed detailed placement of cells within an integrated circuit design**. United States: Google Patents, abr. 2002. US Patent 6,370,673.

HUNG, Chung-Yao; CHOU, Peng-Yi; MAK, Wai-Kei. Mixed-cell-height standard cell placement legalization. In: ACM. PROCEEDINGS of the on Great Lakes Symposium on VLSI 2017. New York, NY, United States, 2017. p. 149–154.

KAHNG, Andrew B et al. **VLSI physical design: from graph partitioning to timing closure**. New York, NY, United States: Springer Science & Business Media, 2011.

- KARIMPOUR DARAV, Nima et al. A fast, robust network flow-based standard-cell legalization method for minimizing maximum movement. In: ACM. PROCEEDINGS of the 2017 ACM on International Symposium on Physical Design. New York, NY, United States, 2017. p. 141–148.
- KENNINGS, Andrew; DARAV, Nima Karimpour; BEHJAT, Laleh. Detailed placement accounting for technology constraints. In: IEEE. VERY Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on. Piscataway, NJ, United States, 2014. p. 1–6.
- KIM, Myung-Chul et al. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. In: IEEE. COMPUTER-AIDED Design (ICCAD), 2015 IEEE/ACM International Conference on. Piscataway, NJ, United States, 2015. p. 921–926.
- LEE, Yu-Min; WU, Tsung-You; CHIANG, Po-Yi. A hierarchical bin-based legalizer for standard-cell designs with minimal disturbance. In: IEEE. DESIGN Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific. Piscataway, NJ, United States, 2010. p. 568–573.
- PAPA, David Anthony. Broadening the Scope of Multi-Objective Optimizations in Physical Synthesis of Integrated Circuits., 2010.
- PAPA, David et al. Physical synthesis with clock-network optimization for large systems on chips. **IEEE Micro**, IEEE, v. 31, n. 4, p. 51–62, 2011.
- POPOVYCH, Sergiy et al. Density-aware detailed placement with instant legalization. In: ACM. PROCEEDINGS of the 51st Annual Design Automation Conference. New York, NY, United States, 2014. p. 1–6.
- REN, Haoxing et al. Diffusion-based placement migration with application on legalization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, Piscataway, NJ, United States, v. 26, n. 12, p. 2158–2172, 2007.
- SHAFFER, Clifford A. Data Structures and Algorithm Analysis. **Update**, v. 3, 2013.
- SPINDLER, Peter; SCHLICHTMANN, Ulf; JOHANNES, Frank M. Abacus: fast legalization of standard cell circuits with minimal movement. In: ACM. PROCEEDINGS of the 2008 international symposium on Physical design. New York, NY, United States, 2008. p. 47–53.
- WANG, Chao-Hung et al. An effective legalization algorithm for mixed-cell-height standard cells. In: IEEE. DESIGN Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific. Piscataway, NJ, United States, 2017. p. 450–455.

WANG, Laung-Terng; CHANG, Yao-Wen; CHENG, Kwang-Ting Tim. **Electronic design automation: synthesis, verification, and test**. Morgan Kaufmann, 2009.

WIKIPEDIA, the free encyclopedia, 2018.

ZHU, Ziran et al. Mixed-cell-height legalization considering technology and region constraints. In: ACM. PROCEEDINGS of the International Conference on Computer-Aided Design. 2018. p. 65.

Apêndices

APÊNDICE A : LISTA DE PUBLICAÇÕES E PRÊMIOS

A.1 ARTIGOS PUBLICADOS DIRETAMENTE RELACIONADOS AO TEMA DE MESTRADO

A avaliação quantitativa da técnica de legalização proposta, resultou em duas publicações diretamente relacionadas ao tema deste trabalho de mestrado. Os detalhes destas publicações estão listados nesta Seção.

A.1.1 XVIII Escola Regional de Alto Desempenho Fórum De Pós-graduação, @ ERAD 2018

- **Título:** *Exploração de paralelismo na etapa de legalização de circuitos digitais através do uso de estruturas de dados geométricas*
- **Autores:** **Sheiny Fabre** ; Laércio Pilla ; José Luís Güntzel
- **Resumo:** *Após o posicionamento inicial dos elementos de um circuito integrado, estes precisam ser legalizados para considerar regras de fabricação. Algoritmos de legalização devem tratar grandes quantidades de dados, produzindo uma solução determinística, com a menor perturbação possível do posicionamento. Este trabalho propõe o uso da estrutura de dados k-d tree para particionar o circuito viabilizando a legalização em paralelo das partições.*
- **Justificativa:** Durante esta publicação a técnica proposta ainda estava em fase inicial de implementação, portanto a mesma foi submetida a um fórum de pós graduação da ERAD. Embora o fórum não seja diretamente relacionado a área de microeletrônica, questões relacionadas a computação de alto desempenho auxiliaram na produção da técnica proposta.

A.1.2 31 Symposium on Integrated Circuits and Systems Design, @ SBCCI 2018

- **Qualis CC 2016:** B2
- **Título:** *Enhancing Multi-Threaded Legalization Through k-d Tree Circuit Partitioning*
- **Autores:** **Sheiny Fabre** ; José Luís Güntzel ; Laércio Pilla ; Renan Netto ; Tiago Fontana ; Vinicius Livramento
- **Abstract:** *In the physical synthesis of integrated circuits the legalization step may move all circuit cells to fix overlaps and misalignments. While*

doing so, it should cause the smallest perturbation possible to the solution found by previous optimization steps to preserve placement quality. Legalization techniques must handle circuits with millions of cells within acceptable runtimes, besides facing other issues such as mixed-cell-height and fence regions. In this work we propose a k - d tree data structure to partition the circuit, thus removing data dependency. Then, legalization is sped up through both input size reduction and parallel execution. As a use case we employed a modified version of the classic legalization algorithm Abacus. Our solution achieved a maximum speedup of 35 times over a sequential version of Abacus for the circuits of the ICCAD2015 CAD contest. It also provided up to 10% reduction on the average cell displacement.

- **Justificativa:** O presente documento reflete diretamente a técnica e resultados presentes nesta publicação, com exceção do refinamento proposto no final da Seção 4.4 e melhorias técnicas de implementação.

A.1.3 *International Symposium on Physical Design, @ ISPD 2019*

- **Qualis CC 2016:** B1
- **Título:** *How deep learning can drive physical synthesis towards more predictable legalization*
- **Autores:** Renan Netto ; **Sheiny Fabre** ; Tiago Augusto Fontana ; Vinicius Livramento ; Laércio Pilla ; José Luís Güntzel
- **Abstract:** *Machine learning has been used to improve the predictability of different physical design problems, such as timing, clock tree synthesis and routing, but not for legalization. Predicting the outcome of legalization can be helpful to guide incremental placement and circuit partitioning, speeding up those algorithms. In this work we extract histograms of features and snapshots of the circuit from several regions in a way that the model can be trained independently from region size. Then, we evaluate how traditional and convolutional deep learning models use this set of features to predict the quality of a legalization algorithm without having to executing it. When evaluating the models with holdout cross validation, the best model achieves an accuracy of 80% and an F-score of at least 0.7. Finally, we used the best model to prune partitions with large displacement in a circuit partitioning strategy. Experimental results in circuits (with up to millions of cells) showed that the pruning strategy improved the maximum displacement of the legalized solution by 5% to 94%. In addition, using the machine*

learning model avoided from 22% to 99% calls to the legalization algorithm, which speeds up the pruning process by up to 3×.

- **Justificativa:** Este trabalho foi inspirado em decisões necessárias para o refinamento da técnica proposta, portanto esta foi utilizada como caso de uso para a elaboração deste trabalho.

A.2 PARTICIPAÇÃO EM OUTRAS PUBLICAÇÕES DURANTE O MESTRADO

Durante a realização deste trabalho de mestrado, houve participação em outras publicações que, apesar de não estarem diretamente relacionados com o problema da legalização, utilizaram alguma das técnicas de legalização incremental apresentadas neste trabalho. Portanto, os detalhes destas publicações estão listados nesta Seção.

A.2.1 *30 Symposium on Integrated Circuits and Systems Design, @ SBCCI 2017*

- **Qualis CC 2016:** B2
- **Título:** *Exploiting Cache Locality to Speedup Register Clustering*
- **Autores:** Tiago Augusto Fontana ; **Sheiny Almeida** ; Renan Netto ; Vinicius Livramento ; Chrystian Guth ; Laércio Pilla ; José Luís Güntzel
- **Abstract:** *Physical design tools must handle huge amounts of data in order to solve problems for circuits with millions of cells. Traditionally, Electronic Design Automation tools are implemented using Object-Oriented Design. However, using this paradigm may lead to overly complex objects that result in waste of cache memory space. This memory wasting harms cache locality exploration and, consequently, degrades software runtime. This work proposes applying Data-Oriented Design on the register clustering problem. Differently from the traditional Object-Oriented design, the Data-Oriented Design programming model focus on how the data is organized in the memory. As consequence, this programming model may better explore cache spatial locality. In order to evaluate the impact of using the Data-Oriented Design programming model for register clustering, we implemented two software prototypes (a sequential and a parallel implementation) of the K-means clustering algorithm for each programming model. Experimental results showed that the sequential Data-Oriented Design implementation is on average 7.5% faster when compared to the Object-Oriented Design*

implementation, while its parallel version is 15% faster when compared to the Object-Oriented one.

- **Justificativa:** Embora esta publicação não seja diretamente relacionada ao presente trabalho, conceitos relacionados a orientação a dados e organização de computadores foram aplicados com o objetivo de melhorar o desempenho da ferramenta de EDA desenvolvida pela equipe do laboratório (GITHUB/ECLUFSC, 2019). Esta mesma ferramenta possibilitou o desenvolvimento e aplicação da técnica de legalização proposta.

A.2.2 *Workshop on Open-Source EDA Technology, @ ICCAD 2018*

- **Título:** *Ophidian: an Open-Source Library for Physical Design Research and Teaching*
- **Autores:** Renan Netto ; Tiago Augusto Fontana ; **Sheiny Fabre** ; Bernardo Ferrari ; Vinicius Livramento ; Thiago Barbato ; João Souto ; Chrystian Guth ; Laércio Pilla ; José Luís Güntzel
- **Abstract:** *There is a lack of open source physical design automation software infrastructure that could be used by researchers to try new algorithms with as little effort as possible, and by students to easily understand and experiment classical algorithms. Such infrastructure should be easy to use and highly modular to allow programmers to develop new algorithms without being required to understand details of the underlying library code. However, such modularity should come without compromising the software performance. This paper presents Ophidian: an open-source library for physical design research and teaching. Ophidian aims to provide a basic infrastructure to develop algorithms for different physical design automation steps, while providing simple implementations of classical algorithms in order to help students understand the physical design flow. In addition, Ophidian makes use of modern software engineering design patterns to allow the development of programs with short runtimes while providing high code modularity. To highlight the benefits of using Ophidian, we implemented a software prototype that executes the A* algorithm for each interconnection segment of a circuit. The experimental results showed that the prototype requires 30% shorter runtime when compared to a traditional Object-Oriented implementation.*
- **Justificativa:** A participação neste *workshop* realizado em um dos eventos mais relevantes da área de automação em microeletrônica,

possui importância estratégica para a divulgação do presente trabalho através da apresentação da ferramenta de EDA desenvolvida pela equipe do laboratório (GITHUB/ECLUFSC, 2019).

A.3 PRÊMIOS

A.3.1 Richard Newton Young Fellow Award @ 54th DAC and Design Automation Summer School

Esta premiação seleciona alunos para participar da conferência DAC e da escola de verão ofertada pela conferência. Os alunos selecionados recebem auxílio financeiro para participarem da conferência e apresentarem seus trabalhos em uma sessão de pôsteres, onde dentre os tópicos de interesse do evento está incluída a automação de projetos eletrônicos. O trabalho apresentado foi desenvolvido durante uma disciplina de programação paralela cursada no mestrado, sendo o tema do trabalho apresentado, *register clustering* utilizando *k-means* em GPU.

A.3.2 2017 CAD Contest (Problem C:Multi-Deck Standard Cell Legalization) @ ICCAD 2017

A técnica de legalização utilizada como caso de uso, Abacus modificado Seção 4.1, foi utilizada na ferramenta de legalização submetida à competição ICCAD 2017 CAD Contest (problem C:Multi-Deck Standard Cell Legalization), a qual proporcionou à equipe do Embedded Computing Laboratory da UFSC o terceiro lugar na competição.

- **Equipe:** Ophidian (cada001)
- **Membros:** Renan Netto, Tiago Augusto Fontana, Sheiny Fabre, Thiago Barbato, Chrystian Guth, Prof. José Güntzel.
- **Colocação:** 3º Lugar

Universidade Federal de Santa Catarina

Programa de Pós-graduação em Ciência da Computação

ppgcc.posgrad.ufsc.br

Campus Universitário Trindade Florianópolis-SC

Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. José Luís A. Güntzel

Coorientador: Prof. Dr. Laércio L. Pilla

Florianópolis, 2019

Aceleração da Legalização de Circuitos Integrados Utilizando Particionamento com *k-d trees*

Sheiny Fabre Almeida

Este trabalho propõe o uso da estrutura de dados espacial denominada *k-d tree* para realizar o particionamento do circuito durante a etapa de legalização em um fluxo de projeto *standard cell*. O uso de tal estrutura promove a redução do tamanho das instâncias fornecidas ao algoritmo de legalização e viabiliza o uso de algoritmos paralelos através da remoção da dependência de dados. Os resultados apresentados incluem redução do tempo de execução em relação à versão sequencial sem particionamento, e melhoria no deslocamento médio da solução.

Orientador: Prof. Dr. José Luís A. Güntzel
Coorientador: Prof. Dr. Laércio L. Pilla

