2012-10-01

Methods for Solving Modern, Scale-Borne Problems in VLSI Physical Design

Rakai, Logan

UNIVERSITY OF CALGARY

Methods for Solving Modern, Scale-Borne Problems in VLSI Physical Design

by

Logan Rakai

A PH.D. THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

SEPTEMBER, 2012

Abstract

The design automation community is confronted with new challenges every technology node. Many of the challenges are borne out of issues relating to scale, be it the very small or the very large. For example, the extremely large scale of the number of instances in modern designs creates challenges in effectively exploring vast solution spaces in reasonable amounts of time. At the other end of the scale spectrum, the extremely small scale of features created by modern lithography processes are highly susceptible to process variations which affect performance and yield. This thesis deals with the development of methods for solving scale-borne problems in the physical design of integrated circuits.

This thesis addresses challenges faced in two important phases of physical design: placement and clock network synthesis. The importance of these two phases is reflected in the fact that they have been the subject of five out of the last seven ACM/IEEE International Symposium on Physical Design (ISPD) contests. The number of instances and the size of the solution space in performing placement are truly immense. A proven linear-time clustering algorithm is proposed to deal with the explosion of problem sizes being encountered. Several extensions to the algorithm are proposed to further improve the quality of results.

The number of instances in clock network design is also growing at a rapid pace. In order to cope with this challenge, a generic framework to parallelize algorithms that perform the main stages of clock network synthesis is proposed. Theorems are provided to prove asymptotically optimal speedup when the framework is applied to several classes of algorithms. Another challenge addressed regarding clock network synthesis is that of variation. A method is proposed for handling variations in lengths and widths of buffers and interconnects, that arise from the manufacturing process.

Acknowledgements

The contributions made in this thesis would not have been possible without the support of many individuals, whom I cannot possibly all mention by name. Anyone that ever taught me anything, regardless of scale, played a part in the composition of this thesis and is acknowledged herein. Those who are directly linked to the production of this thesis are now distinguished.

To all the past and present members of the Design Automation Group, Jie, Jianhua, Bahar, Yangyang, Amin, Bardia, Delaram, Aysa, and Dr. Behjat, thanks for the insightful discussions and many pleasant memories. Thanks to all of the other very talented researchers that I had the privilege of collaborating with during the course of my degree: Dr. Swartz, Dr. Areibi, Dr. Terlaky, Dr. Martin, Dr. Aguado, Dr. Dimitrov, Dr. Westwick, and Dr. Bustany. To the many exceptional researchers whom I never collaborated but got to know through their lectures and publications, thanks for motivating me and fuelling my desire to follow academic pursuits: Prof. Knuth, Prof. Boyd, Prof. Strang, Prof. Leiserson, Prof. Demaine, among others. Thanks to all the members of my defense committee for there constructive feedback. I would also like to thank the economy for giving me the opportunity to pursue Ph.D. studies.

Thanks to my parents, my brother, Kasey, Kita and all of my family for their love and support as this small town boy chased his big dream. To Yilin, all of my friends, and all of the students I have had the pleasure of teaching, thanks for taking my mind off of my research from time to time. To my unconceived children, and my unborn godchild, thanks for being my inspiration.

This thesis is brought to you in part by the National Science and Engineering Research Council of Canada, Alberta Innovates Technology Futures, and made possible by the resources provided by Compute Canada.

Table of Contents

List of Tables

List of Figures

List of Terms

Acronyms:

AB	:	Absorption
ACM	:	Association for Computing Machinery
AMG	:	Algebraic multigrid
AMGC	:	Algebraic multigrid-based clustering
AMGC-LE	:	Length estimation-based algebraic multigrid-based clustering
BB	:	Balanced bipartitioning
BC	:	Best-choice
C	:	Continuous
CCR	:	Cell clustering ratio
CL	:	Clustering-based algorithm
CNS	:	Clock network synthesis
CP	:	C-point selection
C-point	:	Coarse point
CPU	:	Central processing unit
CTS	:	Clock tree synthesis
D	:	Discrete
DME	:	Deferred-merge-embed
FGC	:	Fine granularity clustering
F-point	:	Fine point
GB	:	Gigabyte
GP	:	Geometric programming
HEM	:	Heavy-edge matching
IEEE	:	Institute of Electrical and Electronics Engineers
ISPD	:	International Symposium on Physical Design
ISPL	:	Intrinsic shortest path length
MMM	:	Method of means and medians
NCR	:	Net clustering ratio
NNZ	:	Number of non-zeros
$\mathcal{NP}$	:	Non-deterministic polynomial-time
PinEC	:	Pin edge coarsening
RAM	:	Random access memory
RC	:	Resistor-capacitor
RE	:	Rent's Rule
RGP	:	Robust geometric programming
RO	:	Robust optimization
SMP	:	Symmetric multiprocessor
VLSI	:	Very large-scale integration
WL	:	Wire length

Scalars:

α	:	Positive real constant of a monomial
β	:	Logarithm of α
ϵ	:	Small real constant
γ	:	Positive technology dependent parameter
λ	:	Measure of desirability as a C-point
μ	:	Dummy variable
ω	:	AMG interpolation weight coefficient
$\omega_{\min}$	:	AMG interpolation weight threshold
ρ	:	Robust variable
θ	:	Strong dependence parameter in AMG
a	:	Positive real constant
b	:	Positive real constant
c	:	Positive real constant
$c_{b_i^{\triangleright},\text{in}}$	:	Input capacitance of buffer $b_i^{\triangleright}$
$c_{b_i^{\triangleright},\text{out}}$	:	Output capacitance of buffer $b_i^{\triangleright}$
D_i	:	Downstream capacitance from node i
$d_{b_i^{\triangleright}}$	:	Intrinsic delay of buffer $b_i^{\triangleright}$
d_i	:	Delay of node i
$d_{\min}$	:	Minimum delay
e	:	Euler's number
g	:	Number of equality constraints
i	:	index variable
J	:	Number of monomials in a posynomial
j	:	index variable
k	:	Real constant
L	:	Coarsest level
l	:	Level in AMG
$l_{\min}$	:	Minimum length of buffer
m	:	index variable
N	:	Problem size
N_0	:	Integer constant
n	:	index variable
$n_c^{C_i}$	:	Number of cells in cluster C_i
n_c^l	:	Number of cells in a circuit at cluster level l
$n_e^{C_i}$	:	Number of external net pins of nets in cluster C_i
n_n^l	:	Number of nets in a circuit at cluster level l
$n_{\overline{p}}^{C_i}$	:	Average number of pins in cluster C_i
$n_{p_c}^{C_i}$	:	Number of pins in cluster C_i
P	:	Number of homogeneous processing units in a machine
p	:	Number of inequality constraints
q	:	Input dimension
$r_{b_i^{\triangleright}}$	:	Output resistance of buffer $b_i^{\triangleright}$

t_{skew}	:	Target skew
v	:	Length of all nets connecting fixed cells
$x_{l_{b^{\triangleright}}}$	:	Length of buffer $b^{\triangleright}$
$x_{w_{b^{\triangleright}}}$	:	Width of buffer $b^{\triangleright}$
$w_{\min}$	:	Minimum width of buffer

Vectors:

$\boldsymbol{\beta}$	:	Constants in GP equality constraints
$\boldsymbol{\beta}_i$	:	Constants in GP objective (i=0) or inequalities ($i > 0$)
$\boldsymbol{\phi}_1$	:	Positive gradient terms
$\boldsymbol{\phi}_2$	:	Negative gradient terms
$\mathbf{b}$	:	Linear system right-hand side vector
$\mathbf{c}$	:	Arbitrary vector
$\mathbf{d}$	:	Arbitrary vector
$\mathbf{e}$	:	Error in approximation
$\mathbf{f}$	:	AMG variables
$\mathbf{f}_{C_i}$	:	Location of cells in cluster C_i
$\mathbf{m_i}$	:	GP equality constraints coefficients
$\mathbf{r}$	:	Function residual
$\mathbf{t}$	:	Weighted horizontal cell locations
$\mathbf{u}$	:	Nominal value
$\tilde{\mathbf{u}}$	:	Uncertain variables
$\mathbf{v}$	:	Approximate solution to linear system
$\mathbf{x}$	:	Buffer widths and lengths
$\mathbf{y}$	:	Real variables
$\tilde{\mathbf{y}}$	:	Uncertain variables
$\mathbf{z}$	:	Real variables

Matrices:

$\boldsymbol{\Sigma}$	:	Covariance matrix
$\mathbf{A}$	:	Square matrix
$\mathbf{A}_i$	:	GP objective (i=0) or inequality constraint ($i > 0$) matrix
$\mathbf{A}^l$	:	Modified connectivity matrix in AMG at level l
$\mathbf{P}^l$	:	Proximity matrix in AMG at level l
$\mathbf{U}$	:	Weighted connectivity matrix
$\mathbf{W}_l^{l+1}$	:	Restriction matrix

Singletons:

$b^{\triangleright}$	:	Buffer

c_{max}	:	Cell of maximum degree
p_i	:	AMG point i
s_i	:	Clock source ($i = 0$) or sink ($i > 0$)

Sets:

$\emptyset$	:	Empty set
B	:	Clock buffers
C_i	:	Cluster of cells
C^l	:	C-points at level l in AMG
F^l	:	F-points at level l in AMG
H	:	All nets/hyperedges
$H_{i,j}$	:	Nets containing cells indexed by i and j
h	:	A particular net/set of cells
h_{max}	:	Net of maximum degree
P^l	:	Cells at level l representing points in AMG
$\mathbb{R}$	:	Real numbers
S	:	Clock sinks
U	:	Uncertainty set
$\mathbb{Z}$	:	Integers

Functions:

$\lvert \cdot \rvert$	:	Cardinality
$\lfloor \cdot \rfloor$	:	Floor function
$< \cdot, \cdot >$	:	Inner product
$\Omega(\cdot)$	:	Best-case asymptotic complexity/Big Omega Notation
$\Theta(\cdot)$	:	Asymptotically equal complexity/Big Theta Notation
$\mathrm{AB}(\cdot)$	:	Absorption cost of a set of clusters
$\mathrm{Area}(\cdot)$	:	Total area of buffers
$\mathrm{ab}(\cdot)$	:	Absorption cost of a cluster
$c(\cdot, \cdot, \cdot)$	:	Connection weight between two given cells in a given net
$d_i(\cdot)$	:	Delay from source to sink i
$f(\cdot)$	:	Positive function on $\mathbb{R} \rightarrow \mathbb{R}$
$f_i(\cdot)$	:	Cost function (i=0) or inequality constraint function ($i > 0$)
$\tilde{f}_i(\cdot, \cdot)$	:	Cost (i=0) or inequality constraint ($i > 0$) with uncertainty
$g(\cdot)$	:	Positive function on $\mathbb{R} \rightarrow \mathbb{R}$
$h_i(\cdot)$	:	Posynomial objective (i=0) or inequality constraint ($i > 0$)
$il\,(\cdot)$	:	Inverse of the estimated length of a net
$l_{\text{est}}\,(\cdot)$	:	Estimate length of a net
$\lg(\cdot)$	:	Base-2 logarithm
$\log_b(\cdot)$	:	Base-b logarithm
$\mathrm{lse}(\cdot)$	:	Log-sum-exponential

$m(\cdot)$	:	Monomial
$O(\cdot)$	:	Worst-case asymptotic complexity/Big O Notation
$\mathrm{RE}(\cdot)$	:	Rent exponent of a set of clusters
$\mathrm{re}(\cdot)$	:	Rent exponent of a cluster
$T(\cdot)$	:	Recursive function
$w(\cdot)$	:	Work function

Part I

Introduction

Chapter 1

Introduction

Very large-scale integration (VLSI) is the process of bringing together thousands of electronic components to create an integrated circuit. In VLSI, the physical design of circuits refers to the process of generating a layout for a circuit from its logic description to be used in fabrication. A typical input to the physical design process is a circuit *netlist* represented as a hypergraph, where the vertices represent circuit components, or *cells*, and the hyperedges represent the wires, or *nets*, that connect the cells. The number of nets that connect to a cell, is the degree of the cell, and the number of cells that a net connects, is the degree of the net.

The physical design process can be broken down in three major steps: floorplanning, placement and routing. In floorplanning, size, shape and locations of the main blocks of the circuit are decided. During placement, the locations of the cells of each main block are determined, while the nets are routed as part of the routing step.

The design automation community is confronted with new challenges every time manufacturing technology advances, or every *technology node*. Many of the challenges are borne out of issues relating to scale, be it the very small or the very large [1]. For example, the extremely large scale of the number of instances in modern designs creates challenges in effectively exploring vast solution spaces in reasonable amounts of time. At the other end of the scale spectrum, the extremely small scale of features created by modern lithography processes are highly susceptible to process variations which affect performance and yield. This thesis deals with the development of methods for solving scale-borne problems in the physical design of integrated circuits.

The main objective of placement is to reduce the amount of wires needed to connect

the cells. In order to achieve this objective, cells that are highly connected need to be placed close to each other. Therefore, the main goal in a clustering algorithm used in placement is to identify and group highly connected cells into a new cell, which will be referred to as a cluster. This cluster represents the original cells in the coarsened circuit as a single cell. All nets that connect only to cells contained in a cluster are taken out of the hypergraph representing the coarsened circuit. Once the size of the hypergraph has been reduced to a size that the placer can easily handle, the placer assigns a region on the circuit to each cluster. This process is usually referred to as global placement. After placing all clusters, a local placement algorithm is used to find the exact locations of the cells within each cluster in the region assigned to that cluster.

An important step of physical design which happens after placement or after routing is the synthesis of a clock network. The clock network distributes the oscillating clock signal which is used to time operations as data flows through a circuit. Clock networks must be carefully designed so that the clock signal arrives at nearly the same time to all circuit elements receiving the signal. The larger the discrepancy in clock signal arrival times, the slower the circuit needs to operate to avoid violating timing constraints. Due to the large number of elements receiving the clock signal and the nanometer scale of modern technology nodes, the problem of designing a clock network can be very complicated.

1.1 Thesis Overview

This thesis addresses challenges faced in two important phases of physical design: placement and clock network synthesis. The importance of these two phases is reflected in the fact that they have been the subject of five out of the last seven IBM Research and Intel Research sponsored ACM/IEEE International Symposium on Physical De-

sign (ISPD) contests [2]. The number of instances and the size of the solution space in performing placement are truly immense. Two clustering algorithms and an unclustering algorithm are proposed to deal with the explosion of problem sizes. Highlights of the algorithm include:

- Proven linear-time complexity,
- Improved performance by several measures over other state-of-the-art clustering algorithms,
- Demonstrated to benefit several existing placers.

The number of instances in clock network design is also growing at a rapid pace. In order to cope with this challenge, a generic framework to parallelize algorithms that perform the main stages of clock network synthesis is proposed. The framework is shown to exhibit linear speedup in each stage when applied to several popular algorithms. Features of the proposed parallelization technique include:

- Proven linear speedup,
- Application to existing algorithms without change,
- Easy implementation.

Another challenge addressed regarding clock network synthesis is that of variation. A method is proposed for handling variations in lengths and widths of buffers and interconnect, that arise from the manufacturing process. The problem is formulated as a convex mathematical program to minimize area under given timing constraints and an ellipsoidal variation model. Merits of the proposed method include:

- Obtaining a globally optimal solution,

- Handling various delay models,

- Incorporating spatially-correlated variations.

1.2 Thesis Organization

The remainder of this thesis is divided into three remaining parts:

- Part II is titled Clustering for Placement Problems. Within it are two chapters, the first is Chapter 2 which reviews clustering in the context of circuit placement and discusses concepts required to understand the contributions made in the following chapter. The second chapter in Part II is Chapter 3. In Chapter 3, a clustering technique is proposed based upon a technique for solving large-scale systems of equations. Furthermore, an extension of the clustering technique and an unclustering technique geared towards obtaining the best placement results possible are also proposed.

- Part III is titled Clock Tree Synthesis problems and contains three chapters. In Chapter 4, background material for the following chapters in Part III are presented. The material includes a review of the main aspects of clock tree design as well as an overview of parallel computing concepts and two particular mathematical optimization techniques. The first contributions of Part III are given in Chapter 5 where a framework is proposed for parallelizing different classes of algorithms used in designing clock trees. In Chapter 6, a novel formulation of the clock tree buffers sizing problem is proposed. The formulation is extended to consider process variations to provide solutions which are robust to changes in ideal parameter values.

- Part IV is titled End Matter and includes Chapter 7 which concludes the thesis.

Also included is Appendix A which proves a variety of results used in the work in Chapter 5. Finally, a list of references is provided in the Bibliography.

Part II

Clustering for Placement Problems

Chapter 2

Background Material for Part II

2.1 Introduction

Digital circuits have doubled their size every 18 months for the last 50 years, as predicted by Moore [3]. Today's algorithms for the physical design of circuits need to be able to handle the large scale and the complexity of the designs. Many physical design problems are shown to be $\mathcal{NP}$-hard [4] and cannot be solved optimally in reasonable amounts of time. Hence, heuristics with low computational complexity are needed to provide high quality solutions.

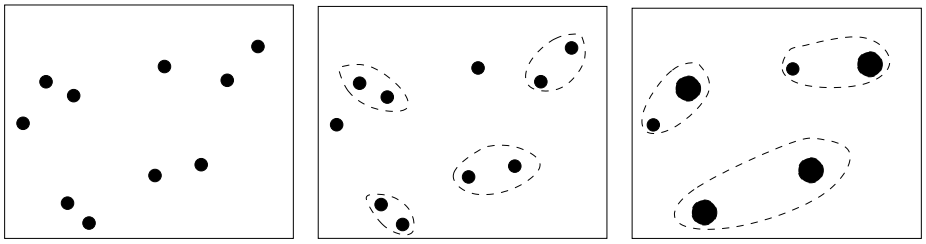
One of the techniques used to increase the efficiency of the placement step of the physical design is to coarsen the hypergraph representing a circuit in several stages using heuristics, which are usually referred to as clustering algorithms. The clustering algorithms used are based on finding small groups of cells with high connectivity and putting each of them in a cluster. This scheme has proven effective, to a large extent, as there is a high correlation between the cells' connectivity and the lengths of the nets that connect them [5, 6]. Hence, by clustering cells that are close to one another, the lengths of the nets between them, and hence the total wire length, will be reduced.

However, these clustering techniques have two major limitations: they only consider local connections when forming clusters and they do not actually use any measure of net lengths to measure the quality of clusters. In this thesis, a new clustering algorithm is proposed, which considers local and global pictures of the circuit for clustering using a mathematical modeling technique called algebraic multigrid [7] in a creative way. In addition, a pre-placement length estimation is used to measure the quality of clusters and directly reduce the lengths of the nets. The proposed approach requires knowledge

and application of several different mathematical fields: clustering, AMG and estimation. The background material relevant to these fields are briefly discussed in the rest of this chapter. The remainder of this chapter is organized as follows: In Section 2.2, a review of existing clustering algorithms is given. The background relevant to AMG techniques is reviewed in Section 2.3. In Section 2.4 existing length estimation techniques are reviewed. Finally, a summary of the chapter is given in Section 2.5.

2.2 Clustering Background

Clustering is usually employed while solving large-scale problems encountered in today's designs to speed up the design process and improve the solution quality. In essence, clustering is a classification technique that partitions the input space into groups according to certain metrics. The output generated by clustering may again be used as an input to a clustering. Each application of clustering can be thought of as a level in a multilevel clustering paradigm. An illustration of multilevel clustering is presented in Figure 2.1. A set of points is given as input in Figure 2.1(a). One level of



(a) Input (b) First level (c) Second level

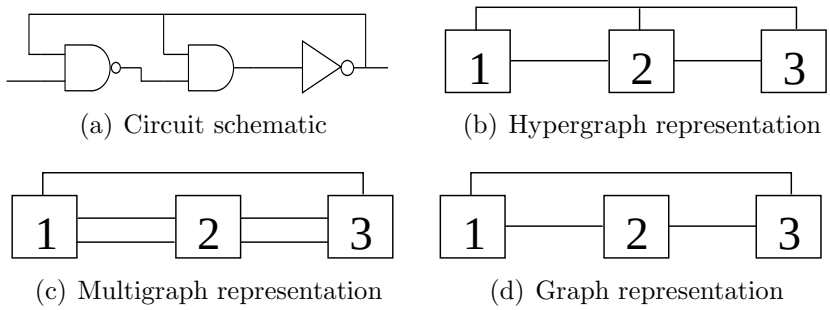
Figure 2.1: Abstract representation of multilevel clustering.

clustering is performed and the generated clusters are surrounded by dashed lines in Figure 2.1(b). Points that were clustered together in the first level are joined together to form larger points seen as input for the second level in Figure 2.1(c). Note that points that are not clustered with other points persist in the output of the clustering.

In this thesis, clustering algorithms designed for the placement stage of physical design are studied. In the context of placement, all clustering algorithms include the following: an abstract circuit representation, a clustering heuristic and a cluster quality metric. Each of these will be briefly discussed in the following sections.

2.2.1 Abstract Circuit Representation

In clustering algorithms, a circuit can be represented by either a hypergraph, multigraph or graph [8]. In Figures 2.2(a) to 2.2(d), an example of a circuit schematic along with its hypergraph, multigraph and graph representations are shown, respectively. In the representations, cells are represented by squares and nets are represented by solid lines. In the hypergraph representation, each net is represented by a hyperedge that



(a) Circuit schematic (b) Hypergraph representation

(c) Multigraph representation (d) Graph representation

Figure 2.2: An example of a circuit schematic, and its hypergraph, multigraph and graph representations.

connects two or more cells. A multigraph representation converts each hyperedge to a set of edges forming a complete graph of the vertices connected by the hyperedge. Because the edges in a multigraph do not always represent entire nets, inaccuracies can arise in such representations [5]. A graph representation combines parallel edges in a multigraph to further simplify the circuit representation. The simplicity of a graph is often desired during clustering, but further inaccuracies can be introduced that can detract from the quality of the clustering technique [5]. Many graph or hypergraph representations have an associated weight for each edge or hyperedge [5]. The weight

should reflect the goal of clustering and can partially account for inaccuracies in a graph or multigraph representation.

2.2.2 Existing Clustering Algorithms Used in Physical Design

During the past decade, numerous clustering algorithms have been proposed for the placement stage of physical design [5, 9–16]. In clustering algorithms, groups of highly connected cells are sought. The reason for this, is because such groups of cells should be placed close together by a placer to reduce wire length. In many algorithms [5, 9, 10, 13, 14], to find highly connected cells, a cell that does not already belong to a cluster is designated as a *seed cell*. A new cluster is grown using the seed cell. Cells that are directly connected to this seed cell are examined and if they pass certain criteria, they are added to the cluster. The criteria usually involve a measure of connectivity. If no suitable cell exists that can be grouped with the seed cell, then the seed cell loses its seed status. Cells are examined as seed cells to form clusters until the circuit size is reduced to a desirable size or all cells have been examined. Other algorithms consider nets as seeds for clustering [9, 11, 12, 15].

In [9], edge coarsening, hyperedge coarsening, and modified hyperedge coarsening were introduced. In edge coarsening, a cell is randomly assigned as a seed and its connectivity with all of its unclustered neighbors is computed. A cluster is formed with the seed cell and its most highly connected unclustered neighbor. In hyperedge coarsening, a hyperedge is chosen randomly. If the hyperedge does not contain cells that already belong to another cluster, its cells are clustered together. Modified hyperedge coarsening is based on the same procedure as hyperedge coarsening with the difference that hyperedges containing clustered cells can still be considered for further clustering. Hyperedge and modified hyperedge coarsening use *seed nets* instead of seed cells to form clusters.

FirstChoice clustering is proposed in [10]. In FirstChoice, seed cells are chosen randomly, and can be clustered with any neighbor cell even though it may already belong to a cluster. In [11] and [12], two variations of edge coarsening, called heavy-edge matching and PinEC, are presented. In heavy-edge matching [11], a connectivity measure for pairs of cells is calculated based on the areas of the cells and the degrees of the hyperedges that connect them. Then, pairs of cells with the highest connectivity measure are clustered. PinEC [12] is a modification to the heavy-edge matching where nets of degree two are given higher weights.

A clustering technique, *edge separability*, is presented in [13]. In this technique, the netlist is first converted into its corresponding graph. All the cells in the graph are considered as seed cells and a computationally-efficient approximation of the maximum flows between each cell and all of its neighbour cells are computed. Subsequently, pairs of cells with the highest approximate flow are clustered.

In [14], a two-phase improvement algorithm, called fine granularity clustering (FGC), is presented. Clusters are initially formed by randomly assigning seed cells and greedily moving all neighbouring cells into the seed cell cluster until the cluster reaches an upper bound size. In the first phase of improvements, cells are moved out of clusters into neighbouring clusters if it results in more nets being included within a single cluster. In the second phase of improvements, groups of two or three cells, as opposed to individual cells, are moved between clusters to increase the number of nets within clusters.

Another technique called best-choice is presented in [5]. In best-choice, clustering scores are first calculated for each cell and all its neighbours. Then, all pairs of cells are placed into a priority queue according to their scores. Pairs of cells with the highest score are clustered and the priority queue is updated. Best-choice has been successfully used in the clustering stage of placers such as mPL6 [17].

In Net Cluster [15], groups of neighboring cells that have more connections between themselves than with other cells are formed. The constructed clusters are then used as guidelines to determine the best set of nets or groups of cells that are supposed to be clustered. Net Cluster has been applied as a preprocessing step for placement.

A clustering technique called SafeChoice is introduced in [16]. This clustering is based on the principle that the quality of placement should not be degraded by any cluster formation. A priority queue of pairs of cells to be clustered is formed, and the clustering is automatically stopped when clustering can result in longer net lengths.

The main commonality between these algorithms is that they use connectivity between cells as the main measure for clustering cells. One of the main differences in existing clustering algorithms explained above is how seed cells are chosen and what criteria are used for determining the connectivity between cells. For example, consider the circuit abstraction in Figure 2.3. If the size of the circuit depicted in Figure 2.3 needs to be reduced by about 60%, choosing cells ②, ⑤, and ⑦ as seed cells and forming clusters: $C_1 = \{$②: ①, ③, ④$\}$, $C_2 = \{$⑤: ⑥$\}$, and $C_3 = \{$⑦: ⑧, ⑨$\}$, results in the best clustering solution. This is due to the fact that the number of nets between the clusters is minimized and the number of nets entirely inside clusters is maximized. However,

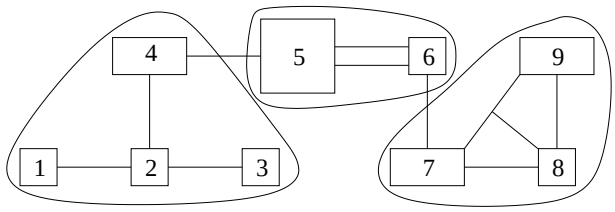


Figure 2.3: A circuit to be clustered and example clusters encircled with solid lines.

even in this simple example, it might not be very obvious which cells need to be chosen as seed cells and which cells need to be grouped with them.

2.2.3 Clustering Performance Metrics

There are several metrics that can be used to measure the quality of a clustering algorithm. The most relevant metrics for clustering are: (i) the quality of the clusters that are produced, such as the number of nets that are totally inside the resulting clusters, (ii) the characteristics of the clustered circuit, and (iii) final placement results. In the rest of this section, specific techniques or formulae to quantify the metrics that are most commonly used in clustering are discussed.

Rent's Rule (RE): Rent's rule measures the quality of a cluster, C_i, and is expressed as a relation between the number of pins, or net terminals, external to C_i belonging to nets connecting to one or more cells in C_i, $n_e^{C_i}$, and the number of cells in C_i, $n_c^{C_i}$:

$$n_e^{C_i} = n_{\overline{p}}^{C_i} \times ({n_c}^{C_i})^{re(C_i)},$$

where $n_{\overline{p}}^{C_i}$ is the average number of pins per cell, and $re(C_i)$ is the Rent exponent.

The Rent exponent can be used as a quality measure to indicate the connectivity within a cluster. A loosely connected cluster would have a Rent exponent value near unity, the maximum. A highly connected cluster on the other hand will have a smaller, possibly negative, Rent exponent. According to [18], the Rent exponent can be approximated by:

$$\mathrm{re}(C_i) \approx 1 + \frac{\ln(n_e^{C_i}/n_{p_c}^{C_i})}{\ln(n_c^{C_i})},$$

where $n_{p_c}^{C_i}$ is the total number of pins in C_i.

The Rent exponent, RE($\cdot$), for a given clustering solution $\{C_1, C_2, ..., C_k\}$ with k clusters is:

$$\mathrm{RE}(C_1, C_2, ..., C_k) = \frac{\sum_{i=1}^{k} \mathrm{re}(C_i)}{k},$$

which is equal to the average of all the Rent exponents.

Absorption (AB): For a given clustering solution, the absorption cost function calculates the number of nets that are *absorbed* by a cluster [19], or connect only to cells

in a single cluster. A good clustering algorithm can maximize the number of nets absorbed. Absorption cost measures the quality of clusters that are produced. A formal definition of absorption cost, ab(·), can be expressed as follows [19]:

$$\mathrm{ab}(C_i) = \sum_{\{h \in H | h \cap C_i \neq \emptyset\}} \frac{|h \cap C_i| - 1}{|h| - 1},$$

where h is a net that has at least one cell in cluster C_i, and H is the set of nets in the circuit. For a given clustering solution, $\{C_1, C_2, ..., C_k\}$, with k clusters, the total absorption cost, AB(·), is equal to the summation of all the absorption weights:

$$\mathrm{AB}(C_1, C_2, ..., C_k) = \sum_{i=1}^{k} \mathrm{ab}(C_i).$$

Cell and Net Clustering Ratios: Cell Clustering Ratio (CCR) is the ratio of cells in a circuit after the clustering, n_c^{l+1}, to the total number of cells, n_c^l:

$$\mathrm{CCR} = \frac{n_c^{l+1}}{n_c^l}.$$

Net Clustering Ratio (NCR) is the ratio of nets in a circuit after the clustering, n_n^{l+1}, to the total number of nets, n_n^l:

$$\mathrm{NCR} = \frac{n_n^{l+1}}{n_n^l}.$$

Cell and net clustering ratios show some of the characteristics of the clustered circuit. The NCR should be as low as possible to make the connectivity of clustered circuit less complex.

Final Placement Results: In the context of circuit placement for physical design, one method to measure the quality of a clustering solution is to obtain placement solutions with and without clustering the circuit and compare placement quality measures, such as the total half-perimeter, or half bounding box, wire length and total runtime, between the two placement solutions. Because the quality of placement solutions given by placers is good, improvements in wire length of a few percent are respectable.

2.3 Algebraic Multigrid

AMG is a technique to efficiently solve large, sparse systems of linear equations, $\mathbf{Af} = \mathbf{b}$ [7, 20–23]. In AMG, to solve $\mathbf{Af} = \mathbf{b}$, a multilevel scheme consisting of three stages is used: restriction or coarsening, direct solution, and interpolation. The coarsening stage is performed in several levels. At each level, the size of the problem is reduced from a *finer* problem to a *coarser* problem. At each level the approximate solution to the coarsened problem is improved. During the direct solution stage, the coarsest problem is solved exactly. In the interpolation stage, the solution of a smaller problem is used to update the approximate solution of a larger problem. A multilevel AMG scheme is illustrated in Figure 2.4. In this Figure, in each level, l, of coarsening, a restriction

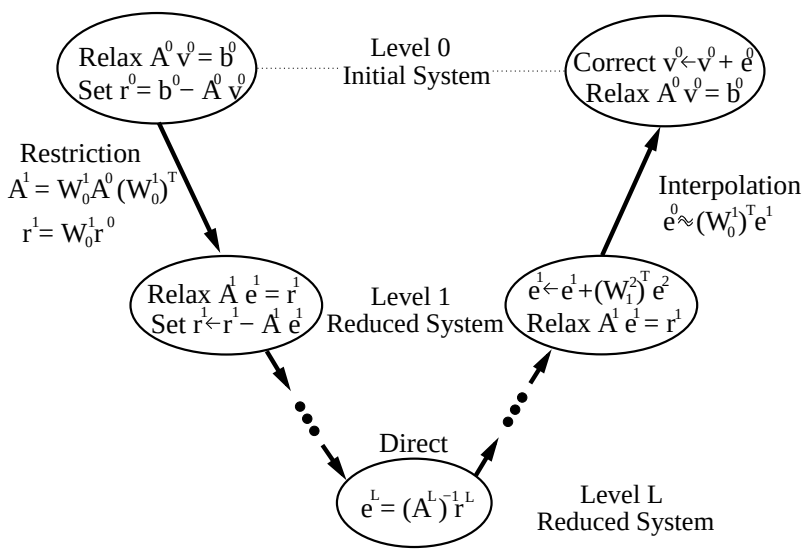


Figure 2.4: A multilevel AMG scheme.

matrix, $\mathbf{W}_l^{l+1}$, is calculated. To reduce the size of the equation matrix at level l, $\mathbf{A}^l$, is pre-multiplied by $\mathbf{W}_l^{l+1}$ and post-multiplied by $(\mathbf{W}_l^{l+1})^T$, the interpolation matrix, and the equation matrix at level $l+1$ is $\mathbf{A}^{l+1} = \mathbf{W}_l^{l+1}\mathbf{A}^l(\mathbf{W}_l^{l+1})^T$ [7]. Matrix $\mathbf{A}^{l+1}$ has lower dimensions than $\mathbf{A}^l$. At the coarsest level, L, an exact solution to $\mathbf{e}^L$ is found, i.e., $\mathbf{e}^L = (\mathbf{A}^L)^{-1}\mathbf{r}^L$ as shown in this figure.

The efficiency of AMG in finding a solution to $\mathbf{Af} = \mathbf{b}$ is due to solving the residual

equation, $\mathbf{Ae} = \mathbf{r}$, where $\mathbf{e}$ is the error in the approximation $\mathbf{v}$ of $\mathbf{f}$, and $\mathbf{r}$ is the residual, $\mathbf{r} = \mathbf{b} - \mathbf{Av}$. The basic idea is that during the coarsening the components of the error, $\mathbf{e}$, which can be removed with low computational cost in each level are eliminated and only components of the error which cannot be easily removed are passed to the next level. In the coarsest level, only the components of the error that have high computational cost are present, and an exact solution methodology can be used to solve for the error efficiently. In the interpolation stage the error is propagated to the finer levels using the appropriate interpolation matrix. Finally using the calculated error in the finest level, $l = 0$, the solution approximation $\mathbf{v}$ is updated: $\mathbf{v}^0 \leftarrow \mathbf{v}^0 + \mathbf{e}^0$. In the proposed clustering algorithm only the coarsening stage of AMG is used. Therefore, this step is described in further detail.

Each element in the solution vector $\mathbf{f}$ is referred to as a *point*, p_i. From the set of all p_i^l at level l, the points that are selected to represent the problem at the coarser level are called *C-points*. The set of points which are not selected are known as *F-points* since they exist only in the finer level. To find C-points at each level, first the notion of strong dependence is introduced, [21]. A point p_i^l is said to *strongly depend* on p_j^l if, for a given $\theta \in (0, 1]$

$$\left|a_{i,j}^l\right| \geq \theta \times \max_{k \neq i} \left\{\left|a_{i,k}^l\right|\right\}, \; k = 1, ..., |P^l|. \tag{2.1}$$

where, $a_{i,j}^l$ is the (i, j)th element of $\mathbf{A}^l$, and $|P^l|$ is the number of points in level l.

To select C-points and F-points, two selection criteria are considered:

Criterion 1: For each F-point, p_i^l, the points that it strongly depends on should either be C-points, or strongly depend on a C-point.

Criterion 2: No C-point should strongly depend on another C-point.

These two criteria may not be mutually satisfiable, so the first criterion is enforced and the second is only violated when necessary [7]. With the C-points and F-points

selected, the interpolation matrix for moving from level $l+1$ to l, $\left(\mathbf{W}_l^{l+1}\right)^T$ is defined using the elements of the matrix $\mathbf{A}^l$, and the matrix $\mathbf{A}^{l+1}$ is recalculated. The output at the end of each coarsening step, l, is a set of 2-tuples $\{\mathbf{A}^{l+1}, \mathbf{W}_l^{l+1}\}$.

There are several different design decisions that need to be made implementing an AMG algorithm. As examples, the parameter θ which determines the definition of strong dependence, the exact method for selecting C-points and F-points, and the formulae for populating entries of the interpolation matrix all need to be decided. The implementation issues relevant to the algorithms developed in this thesis are further discussed as they arise in Chapter 3.

2.4 Length Estimation Background

Estimating the lengths of nets before the placement of their pins are known is very hard problem as the length of nets can depend on several factors. Several *a-priori* length estimation techniques, e.g. [24–27], have been proposed that try to estimate the length of *individual nets*. Older techniques such as [28–33] can only estimate the average length of a set of nets or the distributions of net lengths. As this thesis is only concerned with finding individual lengths, only these techniques are discussed in this chapter.

In [27], a variable referred to as the Intrinsic Shortest Path Length (ISPL) is developed and used to estimate the individual net lengths. Although the estimation results obtained using this technique are exceptional, this approach is not very useful for modern mixed-size circuits since it only considers cells with unit area and nets with degree two.

Different properties of nets and cells are modelled by several variables in [24]. These variables are then used to make a third-order polynomial model for length estimation. The estimation results are well-correlated for relatively small circuits that only include

standard cells. However, none of these variables considers the effects of macro blocks on net lengths and so the estimation results for mixed-size circuits are unreliable. In [26], this technique is further studied using a quadratic polynomial. Three new variables are proposed to account for the effects of macro blocks. The estimation results for modern mixed-size circuits show around 10% improvement over those obtained using [24].

Some applications of a-priori net length estimation techniques are introduced in [14, 25, 34]. A variable called Mutual Contraction, is proposed and used for net length estimation, in [14, 34]. This variable is then used in placement. The estimated net lengths are utilized to quantify the quality of potential clusters.

In [35], another application of pre-placement length estimation is proposed where the negative effects of clustering are corrected based on the estimation results. The technique also adjusts its estimates according to the placer performing the placement.

2.5 Summary

In this chapter, mathematical techniques and background material related to the clustering algorithm proposed in Chapter 3 are discussed. An overview of the existing clustering algorithms for placement and the metrics to measure the efficiency of them are reviewed.

In addition, AMG and a-priori length estimation techniques that are not directly related to clustering but are used in the proposed clustering algorithm are described and relevant prior works in these areas are discussed.

Chapter 3

AMG-Based Clustering Techniques

3.1 Introduction

In this chapter, two AMG-based clustering techniques are proposed and implemented[1]. In the first proposed technique, the AMG coarsening step is used to identify the most connected cells based on the connectivity matrix of a circuit and cluster them. In the second technique, the first proposed AMG-based clustering technique is improved by using estimated lengths of the nets instead of the connectivity matrix for defining which cells should be clustered together. The final contribution of this chapter is to implement a length-driven unclustering technique.

This chapter is organized as follows: In Section 3.2, the proposed connectivity-based AMG clustering is discussed. In Section 3.4, the proposed length-based AMG clustering is extended to a length-based clustering technique. The length-driven unclustering technique is discussed in Section 3.4.2. Numerical results and analysis of experiments using the proposed techniques are given in Section 3.3. A summary of the chapter is given in Section 3.6.

3.2 Connectivity-based AMG Clustering

In this section, an AMG-based clustering algorithm is proposed. The algorithm is designed to use the AMG coarsening step to identify cells that will perform best as seed cells. The interpolation matrix weights are used as a score for measuring the

[1]Portions of the work presented in this chapter are taken from [36, 37]. S. Martin and J. Aguado contributed to the work by bringing multigrid to attention. A. Farshidi contributed by generating pre-placement net length estimates.

connectivity of each seed cell and its non-seed neighbours. Clusters are formed by joining non-seed cells to their seed cells which they most strongly depend on. The main features of the proposed algorithm are as follows:

- Defining seed cell status based on circuit connectivity,
- Using AMG interpolation matrix for weighing connections,
- Providing a global perspective for clustering,
- Linear-time complexity,
- High flexibility to adapt to clustering requirements.

3.2.1 The Proposed AMGC Algorithm

The inputs to the proposed clustering algorithm are the circuit netlist and AMG parameters that control how much a circuit needs to be clustered. The output of the algorithm is the clustered circuit which is then used for placement. The proposed AMGC algorithm consists of four main steps, which are presented in Figure 3.1. In the

Algorithm AMGC: AMG-based clustering
Input: Circuit netlist, AMG parameters Output: Clustered circuit
1. Matrix construction using the circuit netlist 2. AMG coarsening on the constructed matrix 3. Cluster seed cell and score assignment 4. Final cluster formation

Figure 3.1: The high-level flow of the AMGC algorithm.

following sections, each step is described in detail. In addition, implementation details and discussion of the proposed algorithm are presented.

Step 1: Matrix Construction Using the Circuit Netlist

The first step of the algorithm is to find an efficient representation of a netlist using a matrix. It is shown in [7] that for best AMG performance the matrix should be a symmetric, positive-definite *M-matrix*. An M-matrix is a symmetric, positive-definite matrix with positive diagonal and non-positive off-diagonal elements. In the proposed AMG-based clustering technique, to produce such a matrix, a modified connectivity matrix, $\mathbf{A}^0$, is constructed. The elements of $\mathbf{A}^0$ are determined as follows: Consider two cells, i and j, which are treated as points in AMG, and let $H_{i,j}$ be the set of nets, h_k, that contain both cells i and j. Then, element $a^0_{i,j}$ is calculated as:

$$a^0_{i,j} = \begin{cases} \sum\limits_{h_k \in H_{i,j}} -c\left(i,j,h_k\right), & i \neq j,\ H_{i,j} \neq \emptyset \\ \sum\limits_{h_k \in H_{i,i}} \sum\limits_{\substack{l \in h_k, \\ l \neq i}} c\left(i,l,h_k\right), & i = j,\ H_{i,i} \neq \emptyset \\ 0, & H_{i,j} = \emptyset, \end{cases} \tag{3.1}$$

where $c(\cdot,\cdot,\cdot)$ represents the weight of the connection between two cells of a net. In the context of clustering, the weight is normally set to be equal to:

$$c(i,j,h_k) = \frac{1}{|h_k|}$$

where $|h_k|$ denotes the degree of net h_k. Using the formulation in (3.1), any off-diagonal element of the modified connectivity matrix is equal to the negative sum of the connection weights between cells i and j and any diagonal element is equal to the sum of all the connection weights with i. This means that $\mathbf{A}^0$ is a diagonally dominant matrix with positive diagonal elements and negative off diagonal elements. Although an M-matrix is highly desirable for AMG, $\mathbf{A}^0$ is positive semidefinite. The vector of all ones is in the null space of $\mathbf{A}^0$. In the context of circuit clustering, where no system of equations is being solved, this is not an issue. Other desired properties of the connectivity matrix construction proposed in (3.1) are that it handles multi-

terminal nets and multiple connections between cells without having to change a netlist hypergraph into a graph model which can introduce further inaccuracies.

Step 2: AMG Coarsening on the Constructed Matrix

Once the connectivity matrix $\mathbf{A}^0$ has been constructed, it is used in the context of AMG to find seed cells and potential clusters. It is proposed to use the C-points determined by the AMG coarsening process as seed cells and form clusters based on the connectivity of F-points with different C-points.

A customized version of AMG coarsening is developed to find C-points and F-points. The developed method is based on the classical method by Stüben [20], but it is customized to consider cell areas in C-point assignment. The inputs of the algorithm are the set of cells, P^0, and the connectivity matrix, $\mathbf{A}^0$.

Each step is briefly described as follows:

- Step 1: The C-point set, C^0, and F-point set, F^0, are initialized to empty sets and λ_i, a measure of desirability of p_i^0 to be selected as a C-point, is calculated for each point. λ_i is equal to the number of cells that strongly depend on p_i^0.

- Step 2: The smallest cell with the largest λ is chosen as a C-point based on selection criterion 2 mentioned in Section 2.3.

- Step 3a: All of the cells that strongly depend on cell i become F-points.

- Step 3b: The cells that the new F-points strongly depend on are promoted by incrementing their λ values in accordance with C-point selection criterion 1 in Section 2.3.

- Step 4: The cells that the newest C-point strongly depends on are demoted by reducing their λ values.

- Step 5: The algorithm is finished when all cells have been assigned to set C^0 or F^0.

The algorithm is multilevel in nature, so superscript 0 can be replaced by l since it is applied at each level in Algorithm AMGC. The high-level description of the C-point selection algorithm, called Algorithm CP, is given in Figure 3.2.

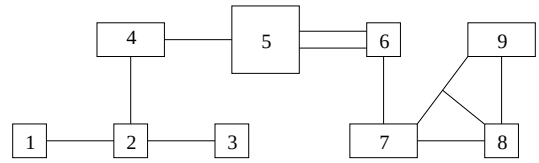
Algorithm CP: C-point selection
Input: A set of points P^l, connectivity matrix $\mathbf{A}^l$ Output: Sets C^l of C-points and F^l of F-points such that $C^l \cap F^l = \emptyset$ and $C^l \cup F^l = P^l$
1. $C^l = \emptyset$, $F^l = \emptyset$, and calculate $\lambda_i, i = 1, ..., \lvert P^l \rvert$ 2. Choose the minimum area point $p_i^l \in P^l$ with maximal λ, set $C^l = C^l \cup \{p_i^l\}$, and $P^l = P^l - \{p_i^l\}$ 3. $\forall p_j^l \in P^l$ that strongly depend on p_i^l 3a. $F^l = F^l \cup \{p_j^l\}$, and $P^l = P^l - \{p_j^l\}$ 3b. $\forall p_k^l \in P^l$ that p_j^l strongly depends on, $\lambda_k = \lambda_k + 1$ 4. $\forall p_j^l \in P^l$ that p_i^l strongly depends on, $\lambda_j = \lambda_j - 1$ 5. Exit if $P^l == \emptyset$ or else go to 2.

Figure 3.2: C-point selection algorithm.

By using the area information to select the smallest cell, which is a modification of the algorithm in [21], clusters of smaller area are more likely to be formed. The effect of this will be discussed more in Section 3.2.1, when more clustering details are given.

The circuit in Figure 3.3(a) is presented to illustrate how Algorithm CP, Figure 3.2, works.

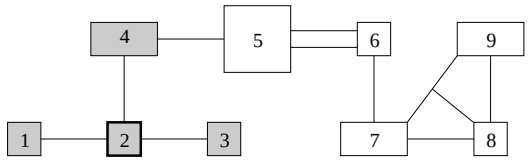
The table in Figure 3.3(b) contains the initial λ value for each cell using $\theta = 0$. The minimum area cell with maximal λ, cell 2 in this example, is the first selected C-cell in Step 2 of Algorithm CP to become a C-point. The cells which strongly depend on cell 2: cells 1, 3, and 4, are considered in Step 3 to perform Steps 3a and 3b. These cells become F-points in Step 3a and the λ value for cells which any of them



(a) Initial circuit

cell	1	2	3	4	5	6	7	8	9
λ	1	3	1	2	2	2	3	2	2

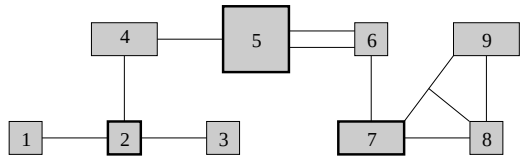
(b) λ values for (a)



(c) Circuit after selecting the first C-point

cell	1	2	3	4	5	6	7	8	9
λ	-	-	-	-	3	2	3	2	2

(d) λ values for (c)



(e) Final C-point selection

Figure 3.3: C-point selection example using Algorithm CP. Shaded cells have been considered by the algorithm and cells with thick borders are selected C-points. In this example $\theta = 0$.

strongly depends on, cell 5 in the example, is incremented by one. The result of the first iteration is shown in Figure 3.3(c) and the new λ values are given in Figure 3.3(d). Cell 7 is the next selected C-point and finally cell 5 is selected in the third iteration. The resulting circuit with C-points highlighted is shown in Figure 3.3(e).

Step 3: Cluster Seed Cell and Score Assignment

After the connectivity matrix has been reduced and the C-points have been identified, all C-points become seed cells to be used for clustering.

The F-points of the circuit are considered to be clustered with the C-points. It is proposed to model the strength of dependence of an F-point on a C-point for this scheme using the AMG interpolation matrix $(\mathbf{W}_l^{l+1})^T$. C-points interpolate directly from themselves, so their row has only one non-zero entry; a one in the column of the C-point. The interpolation coefficients for a point $p_i^l \in F^l$, $\omega_{i,j}^l$, is calculated using the

following equation from [21]:

$$\omega_{i,j}^{l} = -\frac{a_{i,j}^{l} + \sum\limits_{p_m^l \in F_i^{l^s}} \left(\frac{a_{i,m}^{l} a_{m,j}^{l}}{\sum\limits_{p_k^l \in C_i^l} a_{m,k}^{l}} \right)}{a_{i,i}^{l} + \sum\limits_{p_n^l \in F_i^{l^w}} a_{i,n}^{l}}$$

where $a_{i,j}^{l}$ is the (i,j)th element in $\mathbf{A}^l$ and the following sets classify the connections of p_i^l:

- C_i^l is the set of C-points p_i^l strongly depends on,
- $F_i^{l^s}$ is the set of F-points p_i^l strongly depends on, and
- $F_i^{l^w}$ is the set of points p_i^l connects to but does not strongly depend on.

This equation is the commonly used equation for calculating interpolation weights in AMG and is especially efficient for M-matrices. Note that no divisions by zero can occur because of the adherence to C-point selection criterion 1 in Section 2.3, and the construction of matrix $\mathbf{A}^l$.

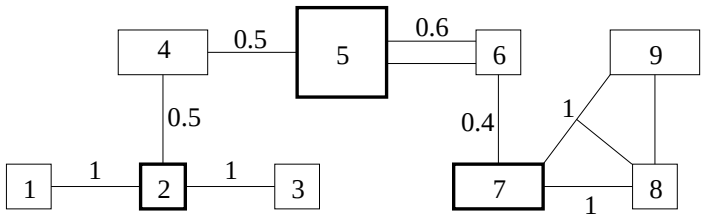
An example of an interpolation matrix and a graph showing the interpolation weights of each F-point to neighboring C-points is given in Figure 3.4. In this example, F-points 1 and 3 only connect to C-point 2 and therefore have an interpolation weight of one. C-point 2 interpolates from itself and also has an interpolation weight of one shown in the interpolation matrix. F-point 4 connects to C-points 2 and 5 and has equal interpolation weight to each. Similarly, F-point 6 connects to two C-points but more interpolates more strongly from 5.

Step 4: Final Cluster Formation

To finalize the cluster formation all F-points are visited. Each F-point is tentatively clustered to the C-point that has the highest interpolation weight coefficient, ω. Using this technique, the order in which the F-points are visited is not important and no

$$(\mathbf{W}_0^1)^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.6 & 0.4 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) Interpolation matrix $(\mathbf{W}_0^1)^T$



(b) Pictorial representation

Figure 3.4: Example showing the interpolation weights of F-points to neighboring C-points in matrix and graph form with interpolation weights annotating edges.

priority queue is needed. The number of C-points that an F-point connects to is guaranteed to be at least one by the construction of Algorithm CP. However, if the maximum weight is small, the cluster may not be desirable. Because of this, if the maximum interpolation weight is not greater than a threshold, $\omega_{\min}$ the F-point is not clustered. Choosing $\omega_{\min}$ near one will limit the number of cells that are clustered, but only the best quality clusters are formed.

After the tentative clusters have been formed, the total area of each cluster is checked to be lower than a maximum allowable area for clusters (1% of the circuit area), or else the cluster is not formed. As will be explained, this is to avoid a problem with white space allocation, i.e. space not occupied by cells, during placement of the clustered circuit. Macro cells are usually the reason for a cluster to exceed the maximum

allowable area. When a standard cell is clustered with a macro, it is physically placed alongside the macro. During the placement stage the resulting cell must be rectangular and this leaves a large amount of whitespace within the clustered cell because of the large differences in the height of the macro and the standard cell. The placer is unaware of the internal organization of the cluster and is forced to remove overlaps that are artificial in a design that has inflated density which can lead to suboptimal results.

Performing the aforementioned cluster formation algorithm on the circuit given in Figure 3.4, results in the clusters shown in Figure 3.5. For example, consider F-point 4 which is connected to C-points 2 and 5 with interpolation weight of 0.5. This tie is broken by choosing the minimum area seed, which is 2.

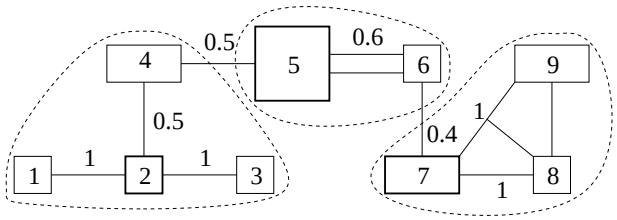


Figure 3.5: Example of the cluster formation showing the seed cells that non-seeds cluster with.

3.2.2 Complexity Analysis

Theorem 3.2.1. *The AMGC algorithm has $O(N)$ (linear) time complexity, where N is the number of pins in the circuit.*

Proof. Since each step in Algorithm AMG is completed apart from other steps, if each step is proven to have linear-time complexity it follows that the entire algorithm has linear-time complexity.

1. The matrix construction algorithm in Section 3.2.1 involves calculating a weight between each pair of cells. However, the number of pairs of cells that are directly connected is far fewer than the total number of pairs of cells in the circuit. Using

a sparse matrix representation, only those directly connected pairs affect the asymptotic complexity. Observe that the number of directly connected pairs of cells is at most equal to:

$$\sum_{h_k \in H} \binom{|h_k|}{2} \leq \sum_{h_k \in H} \binom{|h_{max}|}{2} \tag{3.2a}$$

$$= |H| \binom{|h_{max}|}{2} \in O(N) \tag{3.2b}$$

where H is the set of all nets and $|h_{max}| = \max_{h_k \in H} \{|h_k|\}$ is the maximum degree of a net. The right-hand side of (3.2a) follows from the definition of maximum. In (3.2b), the sum is converted into a product which is in $O(N)$ because the number of nets is on the order of the number of cells for practical designs and $\binom{|h_{max}|}{2} \in O(1)$ because $|h_{max}|$ is independent of the number of cells in the design.

2. Algorithm CP is of linear-time complexity with an efficient implementation. The strong dependence sets for each cell i can be generated in $O(N)$ time. The most connections that a cell can have is $(|h_{max}| - 1) \times |c_{max}|$, where $|c_{max}|$ is the maximum degree of a cell. $(|h_{max}| - 1) \times |c_{max}| \in O(1)$ with the practical assumption that $|c_{max}|$ is independent of the number of cells in the design, the construction of all sets is $O(N)$. The steps of the algorithm can be performed in $O(N)$ [38]. The addition of selecting the minimum area point in Step 2 of Algorithm CP does not affect the asymptotic complexity.

3. The number of F-points is trivially $O(N)$. The number of connections any F-point may have is at most $(|h_{max}| - 1) \times |c_{max}| \in O(1)$. The scoring process is thusly $O(N) \times O(1) \in O(N)$.

4. The maximum interpolation weight of an F-point can be found in $O((|h_{max}| - 1) \times |c_{max}|) \in O(1)$ operations. Therefore, finding the maximum for each F-point is

$O(N)$. The tentative clusters can be stored in a single pass of all cells. Because the number of clusters is $O(N)$ and summing the areas of cells in a cluster is $O((|h_{max}| - 1) \times |c_{max}|) \in O(1)$, the total complexity is $O(N)$.

As each step in the algorithm is $O(N)$ the algorithm is itself $O(N)$. □

Experimental results demonstrating the effectiveness of the AMGC algorithm are presented in Section 3.3. In particular, analysis of cluster structure is given in Section 3.3.1. The performance of AMGC using various measures is given in Section 3.3.2.

3.3 AMGC Experimental Results and Analysis

In this section, several experiments are carried out to verify the efficacy of the proposed AMGC algorithm. These experiments are performed using the ICCAD04 benchmark circuits [39] released by IBM. The detailed statistics of these circuits are given in Table 3.1.

In Columns 2 and 3 of this table the number of nets and cells of each circuit are given, respectively. In Columns 4 and 5, maximum net degree, $|h_{max}|$ and maximum cell degree, $|c_{max}|$ are given. From Columns 4 and 5, it can be observed, as suggested in Section 3.2.2, that the values for $|h_{max}|$ and $|c_{max}|$ do not increase as the number of nets and cells increase.

The rest of this section is organized as follows: the effects of the circuit structure on the proposed AMGC algorithm are analyzed in detail. The analysis of the structure of clusters and the effectiveness of the proposed clustering using various evaluation techniques is shown in Section 3.3.1. Results of experiments measuring the performance of AMGC in terms of clustering metrics, runtime complexity, and placement performance are presented in Section 3.3.2

Table 3.1: Statistics of ICCAD04 Benchmarks.

Circuit	# nets	# cells	$\|h_{max}\|$	$\|c_{max}\|$
ibm01	14,111	12,752	42	39
ibm02	19,584	19,601	134	69
ibm03	27,401	23,136	55	100
ibm04	31,970	27,507	46	425
ibm05	28,446	29,347	17	9
ibm06	34,826	32,498	35	91
ibm07	48,117	45,926	25	98
ibm08	50,513	51,309	75	1165
ibm09	60,962	53,395	39	173
ibm10	75,196	69,429	41	137
ibm11	81,454	70,558	24	174
ibm12	77,240	71,076	28	473
ibm13	99,666	84,199	24	180
ibm14	152,772	147,605	33	270
ibm15	186,608	161,570	33	306
ibm16	190,048	183,484	40	177
ibm17	189,581	185,495	36	81
ibm18	201,920	210,613	66	97

3.3.1 Cluster Structure Analysis

In this section, the results of a set of experiments are presented that show the relationship between AMGC and the structure of the benchmark circuits, and the proposed AMG-based clustering technique is used to gain insight into the structure of the circuits.

In Table 3.2, some of the characteristics of AMGC when applied to the ICCAD04 benchmarks are given and are discussed to gain insight into the structure of the circuits. In the first set of columns, Columns 2 to 5, of this table specific AMGC statistics, when $\theta = 0.8$ are given. In Column 2 of this table, the percentage of C-points after one level of AMG-based reduction in each circuit is given. On average, around 43% of the cells in a circuit are identified as C-points. The circuits with the highest percentage of C-points are ibm02, ibm08, and the circuit with the lowest percentage of C-points is ibm05. Both ibm02 and ibm08 circuits have a large number of cells and nets with

Table 3.2: Results of AMGC for ICCAD04 circuits using $\theta = 0.8$.

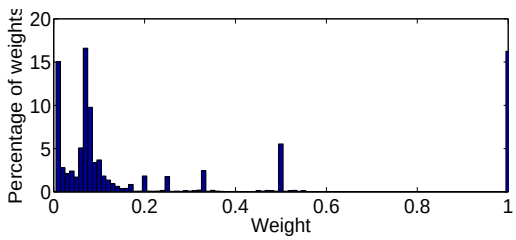
Circuit	Clustering stats				F-point Interpolants			C-point Interpolations			Cluster Stats $\omega_{\min} = 0$		Cluster Stats $\omega_{\min} = 1$	
	% C-p	% CCR	% NCR	% NNZ	Ave	Max	Std	Ave	Max	Std	Ave	Max	Ave	Max
ibm01	39	39.8	61.9	0.03	1.4	36	1.5	3.6	30	1.8	3	21	2.8	10
ibm02	46	46.6	66	0.03	2.5	91	6.2	5.5	56	3.6	2.9	33	2.7	13
ibm03	43	43.9	63.4	0.02	1.6	52	2	3.8	58	2.7	3.1	21	2.9	22
ibm04	43	43.7	66	0.01	1.5	63	1.5	3.5	213	3.1	3.1	19	3.2	20
ibm05	33	34.5	44.2	0.01	1.1	34	0.7	3.4	15	1.3	3.3	15	3.1	17
ibm06	42	42.8	63	0.01	1.4	39	1.5	3.4	60	1.8	3.1	21	2.8	12
ibm07	45	45.2	67.5	0.01	1.7	36	2.1	3.9	70	2.3	3	15	2.8	11
ibm08	49	49.3	68.8	0.01	2.6	80	4.1	5.2	316	4.3	3	31	2.6	12
ibm09	43	43.3	65.3	0.01	1.6	44	1.8	3.7	87	2.7	3.1	22	3	19
ibm10	45	45.2	65.2	0.01	1.7	40	2.2	3.7	112	2.8	2.8	44	2.7	42
ibm11	41	41.5	66.5	0.01	1.6	32	1.6	3.8	127	2.8	3.3	25	3.1	25
ibm12	45	46.2	68.3	0.01	1.8	41	2.2	4	403	4.9	3	18	2.8	14
ibm13	41	41.8	65.8	<0.01	1.7	34	2	4	140	3.1	3.2	40	3	39
ibm14	44	44	65.1	<0.01	1.6	33	1.7	3.6	186	2.9	3	185	2.8	103
ibm15	44	43.9	68.6	<0.01	1.7	43	2.2	3.9	288	3	3.1	141	2.9	102
ibm16	44	44.6	65.9	<0.01	1.7	39	2.2	3.9	160	2.6	3	76	2.8	65
ibm17	45	45.2	68.5	<0.01	1.9	46	2.6	4.3	74	2.3	3.1	56	2.8	45
ibm18	43	42.6	65.3	<0.01	2.1	60	3.3	5	51	3.5	3.1	28	2.8	17
Overall	43.1	43.6	64.7	0.01	1.74	91	2.3	4.02	403	2.8	3.1	185	2.9	103

high degree, but on the other hand, ibm05 is a very homogeneous circuit and has no macro-cells or cells with degree higher than 10. Therefore, there are far fewer points in ibm05 that can be used as C-points.

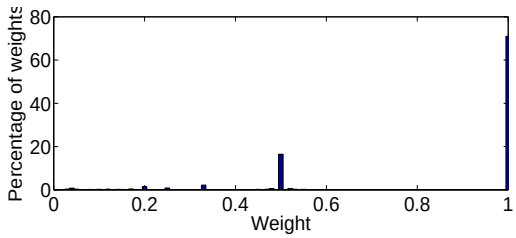
In Columns 3 and 4 of the table, the cell clustering ratio (CCR) and the net clustering ratio (NCR) are given, when the minimum interpolation weight required for an F-point to be considered for clustering, $\omega_{\min}$, is zero. CCR and NCR show the reduction in the number of cells and nets in the clustered circuit. From these columns it can be seen that even though NCR is on average 21.1% higher than CCR, the clustering algorithm is still successful at reducing the number of nets in the circuits. In Column 5 of the table, the percentages of non-zero elements in the interpolation matrix are given. From this table, it can be seen that over 99.97% of the elements of the interpolation matrix are zero. This result is important as it shows how sparse the matrices used for AMG are.

In the second set of columns of Table 3.2, Columns 6 to 8, statistics related to F-

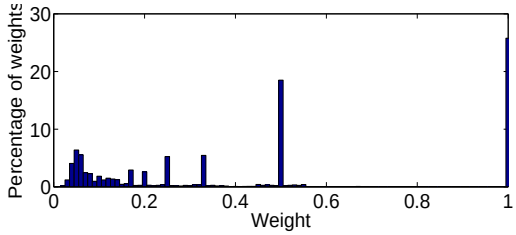
points and the number of their interpolants are given. An interpolant for an F-point is a C-point that the F-point connects to, i.e. a non-zero entry in the interpolation matrix corresponding to the connection between the F-point and C-point. For all circuits the minimum number of interpolants for an F-point, i.e. strong C-point dependencies, is equal to one and hence the minimum was not shown in this table. On average for all circuits, each F-point is interpolated from less than two, 1.74, C-points. Circuits ibm02 and ibm08 have the highest average interpolants and circuit ibm05 has the lowest average. The maximum number of interpolants for an F-point is 91 for ibm02. In Figure 3.6 the histograms for the weights for F-points in the interpolation matrix for circuits, ibm02, ibm05 and ibm10 are shown. In these figures, the abscissa is the value



(a) ibm02



(b) ibm05



(c) ibm10

Figure 3.6: Histograms of F-point weights (non-zeros only) in the interpolation matrix.

of the interpolant in the matrix, and the ordinate is the percentage of the occurrence of

the value. Note that the range of the ordinates are different to show the details of each graph. In ibm02, there are a large number of nets with high degree, therefore, there are many F-points with loose connections to C-points, as shown in Figure 3.6(a). In comparison, in Figure 3.6(b), the histogram of the values of interpolants for the F-points in ibm05 is shown. As it can be seen, more than 70% of the F-points strongly depend on only one C-point, and most of the rest of the F-points have equal connection weights with two C-points. In Figure 3.6(c), a more typical circuit, ibm10, is shown where around 25% of the F-points are strongly depend on only one C-point and another 25% depend equally on two C-points. This information on the typical values of interpolants can help the clustering algorithm decide which cells should be given priority to be clustered and how much the circuit can be reduced while maintaining the structure of the circuit. For cases where F-points depend equally to multiple C-points, three tie-breaking methods were investigated: minimum area, minimum degree, and maximum degree of the corresponding C-points. These methods were chosen because they all preserve the linear runtime complexity. To evaluate them, the number of unresolved ties, i.e. F-points that remained tied after attempting to break the tie using one of the methods, and placement solution quality are considered. All techniques can break the ties more than 80% of the time on average, with minimum area breaking the most ties in 12 of the 18 benchmarks. In the placement results tie-breaking using minimum area performs best on average and hence is chosen for the proposed technique. This result matches intuition, which would suggest that keeping clusters smaller allows more solution space to be explored while placing the clustered circuit.

In the third set of columns of Table 3.2, Columns 9 to 11, statistics related to C-point interpolations are given. In all circuits, the minimum number of F-points for a C-point is one. On average for all circuits, each C-point has four F-point interpolations, with again ibm02 and ibm08 having the highest average C-point interpolations and ibm05

having the lowest average, maximum and standard deviation. This again shows that ibm05 is a very homogeneous circuit, with not much difference between C-points and F-points or any strong point of connection. However, ibm02 and ibm08 are circuits with a lot more variety and C-points that are very distinct and are strong points of connections between different cells.

In last set of columns in Table 3.2, statistics on the clusters, average number of cells in a cluster and maximum number of cells in a cluster, are given. The two values $\omega_{\min} = 0$ and $\omega_{\min} = 1$ are the extreme cases for the minimum required interpolation weight for an F-point to interpolate from a strongly depended upon C-point. The average number of cells in a cluster is high for ibm05 since most of the F-points in ibm05 belong to a C-point and there are very few F-points that have loose connection to C-points. Also, since there are no macro blocks in the circuit, all clusters are acceptable since they do not pass the maximum cluster area limit, which is 1% of the total cell area. It can be seen that when $\omega_{\min}$ is set to one, the average number of cells in a cluster decreases because F-points without a unity interpolant are not considered for clustering. However, the clustering is still effective in this extreme case because many points do possess only a single interpolant.

To be able to better show the cluster sizes, a histogram of the number of cells in clusters for ibm01 is shown in Figure 3.7. From this figure, it can be seen that most of the clusters formed have one to three cells, where one cell means the cell is not clustered. This means that only a few C-points have maximal interpolation weights for more than two F-points and confirms the idea that most clusters should be made of only a few cells.

In Figure 3.8, the effects on CCR and clustering time as θ varies for the circuit ibm01 are illustrated. It can be seen from Figure 3.8(a) that the method is effective at coarsening the circuit for all values of θ. For this circuit the minimum CCR is

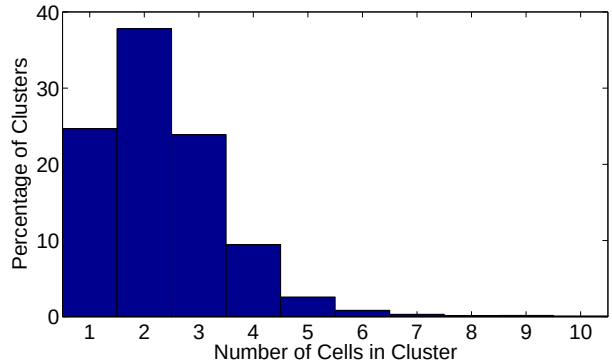


Figure 3.7: A histogram showing the number of cells in each cluster for ibm01 with θ=0.8 and $\omega_{\min} = 0$.

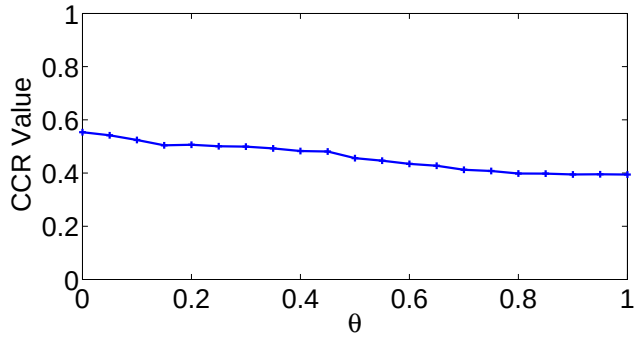
close to 0.4 which means that the circuit is reduced to almost 40% of its original size. Furthermore, the time generally decreases as θ increases for all connection weights, as shown in Figure 3.8(b). Considering that clustering should be performed quickly and reduce circuit sizes significantly, a value of θ greater than 0.6 is most appropriate. These trends are consistent across other benchmarks. In the implementation, θ was set to 0.8 to require near maximal connection weight for a point to be considered as a strong dependence and to reduce runtime.

Finally, more insights are obtained by observing the properties of the interpolation matrix at each level in a six-level coarsening process. In Table 3.3, an example for ibm01 is given. The number of rows, Column 2, shows the number of points in a level

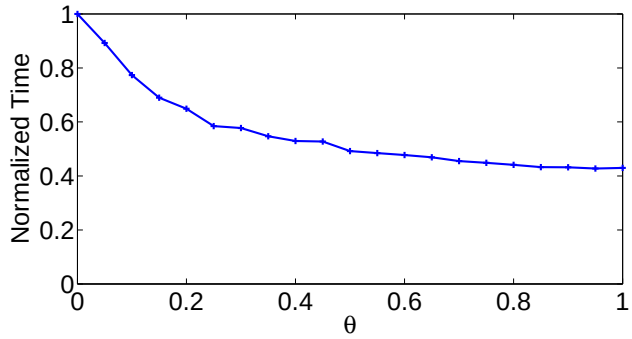
Table 3.3: Properties of the interpolation matrix, ${\mathbf{W}_{l-1}^{l}}^{T}$, with θ=0.8 for ibm01 in a multilevel coarsening.

Level	Number of rows	Number of cols	NNZ	Density	Average NNZ/row
1	12506	4910	17793	<0.001	1.4
2	4910	2092	6525	0.001	1.3
3	2092	827	2636	0.002	1.3
4	827	299	943	0.004	1.1
5	299	121	339	0.009	1.1
6	121	48	127	0.022	1

and the number of C-points is equal to the number of columns and is given in Column 3.



(a) CCR vs. θ



(b) Time vs. θ

Figure 3.8: The effect on CCR and time when θ varies for ibm01 using different weights.

The coarsening reduces the matrix by approximately 60% in each level, which provides much smaller problems after only a few levels. The number of non-zeros (NNZ) show a similar significant reduction. The density shown in Column 5, which equals NNZ divided by the number of elements in the matrix, increases with each level but the average number of NNZ per row, Column 6, remains relatively constant.

3.3.2 Performance of AMGC

To be able to evaluate the effectiveness of the proposed AMGC algorithm, a comparison of cluster performance metrics between the AMGC technique and existing clustering methods is given in Section 3.3.2. In Section 3.3.2, the linear-time complexity is verified. In addition, the effectiveness of the proposed AMGC technique in the context of circuit placement is illustrated in Section 3.3.2.

Clustering Performance Metrics Results

In this section, the performance of the proposed AMGC technique compared to existing techniques is evaluated using the direct clustering metrics discussed in Section 2.2.3. First, a comparison between the proposed clustering method using $\theta = 0.8$ and heavy-edge matching (HEM) [11] and best-choice (BC) [5] is presented in Table 3.4. Both techniques are cell-based, as is the proposed clustering algorithm. BC is used since it is a very high quality clustering algorithm and has been used extensively during placement. HEM randomly selects seed cells, distinguishing it from BC.

Table 3.4: Comparison between the clustering metrics obtained with AMGC and state-of-the-art methods: best-choice (BC) and Heavy edge matching (HEM).

		AMGC			HEM Comparison			BC Comparison		
Circuit	CCR (%)	NCR (%)	AB	RE	NCR (%Δ)	AB (%Δ)	RE (%Δ)	NCR (%Δ)	AB (%Δ)	RE (%Δ)
ibm01	40	62	6622	0.26	**24**	**65**	**2**	**8**	**8**	**12**
ibm02	47	66	8162	0.22	**16**	**35**	**15**	**2**	**0**	**20**
ibm03	44	63	11791	0.25	**21**	**65**	**12**	**10**	**18**	**20**
ibm04	44	66	13032	0.25	**20**	**57**	**14**	**8**	**12**	**21**
ibm05	35	44	17257	0.38	**42**	**88**	-51	**18**	**11**	-37
ibm06	43	63	15548	0.31	**22**	**56**	-6	**6**	**5**	**3**
ibm07	45	67	19418	0.26	**16**	**38**	**10**	**1**	**0**	**17**
ibm08	49	69	19400	0.25	**13**	**25**	**4**	-2	-7	**7**
ibm09	43	65	25917	0.29	**21**	**62**	**1**	**10**	**17**	**11**
ibm10	45	65	32187	0.23	**18**	**41**	**12**	**5**	**4**	**20**
ibm11	42	67	35646	0.28	**22**	**77**	**7**	**11**	**22**	**16**
ibm12	46	68	29783	0.25	**16**	**37**	**11**	**5**	**2**	**19**
ibm13	42	66	41928	0.29	**22**	**68**	**3**	**10**	**16**	**11**
ibm14	44	65	65814	0.27	**18**	**43**	**6**	**3**	**2**	**13**
ibm15	44	69	74845	0.28	**18**	**59**	**3**	**7**	**11**	**12**
ibm16	45	66	81022	0.26	**18**	**41**	**8**	**3**	**0**	**17**
ibm17	45	68	74618	0.27	**16**	**38**	**3**	**4**	**1**	**11**
ibm18	43	65	91105	0.25	**20**	**42**	**5**	**3**	-1	**13**
Avg.	43	65	-	0.27	**20**	**52**	**3**	**6**	**7**	**11**

In Table 3.4, Columns 2 to 4 show the values for NCR, AB and RE obtained using the proposed AMGC technique. In Columns 5 to 10, comparisons of the above values when HEM and BC are used are given. The columns are labeled with %Δ, which are the percentage improvement calculated by subtracting the value using HEM or BC from

the AMGC value and dividing by the value obtained used HEM or BC. Values in bold signify that AMGC outperformed the existing method. Positive numbers represent improvements with this system. The average values are given in the final row of the table. It can be seen that the proposed method improves all scores, on average and in most cases, when compared to the existing methods.

Linear-Time Complexity Verification

In this section, it is verified that the runtime of the clustering algorithm is linear in the number of pins for circuits in the ICCAD04 benchmark suite. A plot of the runtime, which is the time to perform all of the steps in Algorithm AMG, versus the number of pins in each circuit is shown in Figure 3.9. The data is well-fitted with a

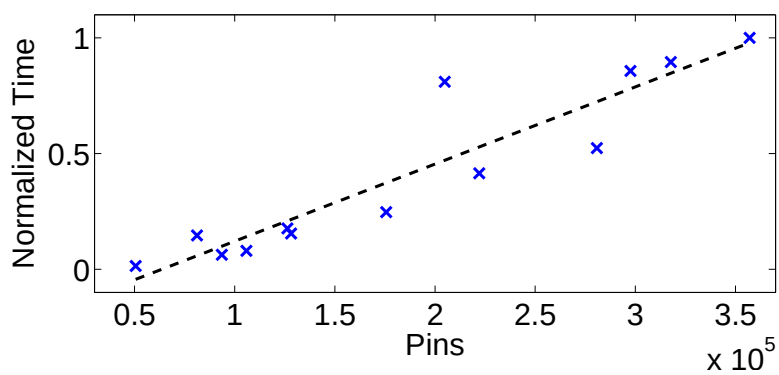


Figure 3.9: Normalized runtime of Algorithm AMGC versus number of pins in each circuit.

line generated using ordinary least-squares fitting. Deviations can be seen due to the varying maximum cell and net degrees for each circuit, which are important parameters in proving the linear-time complexity. The maximum deviation is for circuit ibm08 which has just over 2×10^5 pins. ibm08 is also the circuit with the largest maximum cell degree, as seen in Table 3.1.

Placement Performance

In this section, the results of an experiment using the state-of-the-art placer mPL6 [40] and HEM, BC, and the proposed AMGC algorithm are given. In the experiment, first the placer mPL6 is used to perform placement without clustering on the ICCAD04 benchmark circuits in order to obtain a baseline result. Then, each clustering technique is used to cluster the circuits, and the clustered circuit is placed using mPL6. Finally, mPL6 is used to place the unclustered circuit which is mapped from the placement of the clustered circuit. The results are presented in Table 3.5. In Columns 2 and 3 the total half-perimeter wire length (WL) and runtime of the placement using mPL6 without clustering are given. In Columns 4 to 9 the results using mPL6 with HEM, BC, and the proposed clustering algorithm are compared to the baseline in percentage improvement. On average, the largest improvement in wire length is 1.97% using the proposed clustering algorithm, which is a significant improvement considering that the results without clustering are good to begin with. The number of circuits with improvement in wire length are 11, 9, and 12, for HEM, BC, and the proposed AMGC algorithm, respectively. In addition, the proposed clustering algorithm improves all of the six largest circuits, where clustering is required the most. The circuits that had poorest results in terms of wire length include ibm02, ibm05, and ibm08 which have been identified as circuits with specific properties in the previous sections.

The performance of Net Cluster [15] was also compared to AMG. In these experiments AMGC showed an average improvement of 1.97% over Net Cluster's 1.53%. The reason for not including the details of Net Cluster is as follows: In Net Cluster the Cell Clustering Ratio (CCR) could not be set. The algorithm stops when it cannot find any more "natural clusters" or clusters with more nets inside the cluster than cut by it. The number of natural clusters is far less than the number of clusters formed using the AMG-based technique. Including Net Cluster would have meant that the CCR for

Table 3.5: Placement wire length and runtime results for AMGC and comparison with other clustering techniques. In all these experiments mPL6 was used as the default placer.

(a) HPWL

Circuit	No Clustering ($\times 10^5$)	HEM % Comp	BC % Comp	**AMGC** : $\theta = \mathbf{0.8}, \omega_{\min} = \mathbf{0}$ % Comp
ibm01	24	5.9	7.1	**5.7**
ibm02	52	-6.3	-1.6	-9.9
ibm03	82	2.2	6.1	**5.8**
ibm04	109	0.4	-1.3	-1.9
ibm05	93	-5.4	-5.3	-5.4
ibm06	88	10	12.5	**11.9**
ibm07	124	1.4	-0.1	**2.2**
ibm08	211	-2.7	-2.9	-2.9
ibm09	189	5.1	-2	**3.7**
ibm10	363	1.9	1.7	**2.1**
ibm11	243	-2.6	-4.6	-1.8
ibm12	461	-5	-5.9	-4.2
ibm13	324	-1.2	0.3	**1.7**
ibm14	824	6.9	7.4	**7.4**
ibm15	1001	-7.5	-3.3	**1.7**
ibm16	931	2.5	2.4	**2.6**
ibm17	1144	5.7	5.9	**4.9**
ibm18	885	11.6	12	**12**
Average	-	1.27	1.57	**1.97**

(b) Runtime

Circuit	No Clustering (s)	HEM % Comp	BC % Comp	**AMGC** : $\theta = \mathbf{0.8}, \omega_{\min} = \mathbf{0}$ % Comp
ibm01	150	-20	-20	-14
ibm02	324	-39	-41	-38
ibm03	281	-67	-52	-85
ibm04	344	-40	-28	-31
ibm05	232	-45	-29	-22
ibm06	475	-62	-32	-49
ibm07	488	-51	-43	-49
ibm08	1468	-29	-44	-26
ibm09	1134	-26	-26	-24
ibm10	1762	-45	-49	-45
ibm11	1501	-15	-1	-4
ibm12	2009	-41	-57	-36
ibm13	1456	-28	-26	-23
ibm14	2547	-39	-33	-29
ibm15	4208	-48	-26	-23
ibm16	5821	-33	-35	-26
ibm17	3025	-52	-53	-59
ibm18	3672	-47	-49	-49
Average	-	-40.4	-35.8	-35.1

each circuit would have to be set to match the CCR for Net Cluster, hence, placement results obtained from Net Cluster were omitted from Table 3.5.

It should also be mentioned that the structure of the circuits in the ICCAD04 suite are very different. The sizes and number of cells vary greatly in these circuits, for example, in ibm05 all cells have area less than 0.02% of the total area, but in ibm04 which is very similar in size to ibm05, close to 14% of cells have area more than 0.02% of the total area. There are great differences in the degree of nets of each circuit. For example, the highest net degree in ibm02 is 134 where in ibm05 it is 17. These vast differences greatly affect the placement and hence the clustering results. Better results for each circuit can be obtained by tuning the parameters for AMG. The tuning can be done in two ways, one is based on trial and error to see what parameters works best for each circuit. Even though this technique can show improvement in the final results, it will not enhance the knowledge of how clustering should be performed and will make comparisons with other techniques, such as best-choice, unfair. Future research in this area will include the study of the global and local properties of a circuit that can be obtained through the AMG process and how these properties affect placement. Then, use the results of the study to devise an adaptive AMGC technique.

Using the proposed framework of clustering, placement and legalization, unclustering and further placement and legalization, results in an increase of 35% to 40% in total runtime for all three clustering methods as shown in columns 5, 7 and 9. This runtime increase is because the placement problem has to be solved twice: once for the clustered circuit and then again when the circuit is unclustered. The runtime will be reduced when the clustering technique is fully integrated with a placer. It should also be mentioned that the proposed AMGC algorithm finishes marginally faster than the other techniques.

3.4 Length-Driven Multilevel AMG Clustering Framework

The main objective of the placement phase in the physical design of circuits is to reduce the total wire length of the circuit. Clustering algorithms are normally used during placement to improve the total wire length and runtime. However, clustering is performed based on circuit connectivity and not the wire length. In this section, the proposed AMGC algorithm in Section 3.2 is modified to perform clustering based on estimated wire lengths. The novelty of the algorithm is in using the estimated lengths instead of connectivities in determining the best cells to be clustered.

3.4.1 Length Estimation-Based AMG Clustering Algorithm

The proposed algorithm has four stages similar to the four stages as the algorithm AMGC given in Figure 3.1: matrix construction, AMG-based coarsening, cluster seed cell and score assignment, and final cluster formation. In addition, Step 0 is added to the algorithm during which an individual pre-placement length estimation is performed. The steps of Algorithm AMGC-LE are summarized in Figure 3.10, and further explained in the following sections.

Algorithm AMGC-LE: AMG-based clustering with length estimation
Input: Circuit netlist, AMG parameters Output: Clustered circuit
0. Pre-placement individual net length estimation 1. Proximity matrix construction using the netlist and estimated lengths 2. AMG-based coarsening on the proximity matrix 3. Cluster seed cell and score assignment 4. Final cluster formation

Figure 3.10: The high-level flow of the AMGC-LE clustering algorithm.

Step 0: Pre-Placement Length Estimation

One of the main contributions of this thesis is to use the pre-placement length estimation for clustering. In an estimation technique, first a set of model parameters needs to be calculated. In this work, the parameters used or developed in [26] are employed as these have been shown to be a comprehensive set of parameters that are well-suited to the mixed-size circuits used today. These parameters include local net characteristics, such as the half perimeter of the cells of each net, and global characteristics, such as the number of degree-two nets in the design.

Once the model parameters have been selected, an estimation technique should be used to fit the parameters into a model. The model parameters in [26] are used to fit a quadratic model. Any terms of the resulting model which have small coefficients, ineffective terms, are pruned from the model. The remaining terms, effective terms, are used to make the estimation model and calculate the length estimates.

It is worth noting that any other individual net length estimation technique could be used in this step. The model in [26] is selected as it includes several factors such as macro cells but is independent of the placer to be used allowing for broader application of this work.

To better illustrate the algorithm flow, a small example circuit is presented in Figure 3.11. In this figure, the nets are annotated with the estimated lengths. The estimated lengths are found using a simplified version of the technique in [26]. Due to the small size of the example circuit, only the variables associated with the cell dimensions and net degree are used. These lengths will be used in the following sections to perform clustering using the proposed AMGC-LE algorithm.

Step 1: Proximity Matrix Construction Using Circuit Netlist and Estimated Lengths

To perform AMG-based clustering, a circuit's netlist should be represented using a matrix. In Algorithm AMGC, this matrix is the connectivity matrix of the circuit

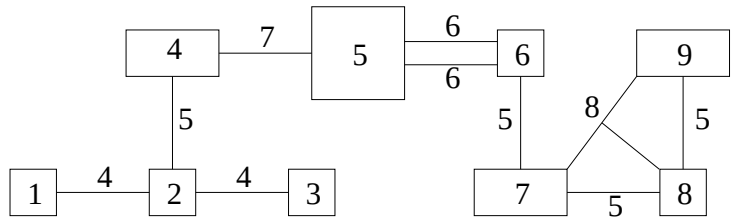


Figure 3.11: An example circuit with nets annotated with estimated lengths. The dimensions of all cells are 2×2 except cells 4, 7 and 9 that are 4×2 and cell 5 which is 4×4.

which shows the total number of weighted connections between two cells. In this section, a new matrix, called the proximity matrix, $\mathbf{P}^0$, is designed which shows how close two cells are estimated to be in the final placement. This proximity matrix is then used for the AMGC-LE clustering instead of the connectivity matrix.

The rows and columns of $\mathbf{P}^0$ represent the cells of the circuit. The element $p^0_{i,j}$ shows the proximity between cell i and cell j. Each $p^0_{i,j}$ is determined as follows: consider two cells, i and j, which are treated as points in AMG, and let $H_{i,j}$ be the set of nets, h_k, that contain both cells i and j. Then, element $p^0_{i,j}$ is calculated as

$$p^0_{i,j} = \begin{cases} \sum\limits_{h_k \in H_{i,j}} -il\left(h_k\right), & i \neq j,\ H_{i,j} \neq \emptyset, \\ \sum\limits_{h_k \in H_{i,i}} il\left(h_k\right), & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{3.3}$$

where $il(\cdot)$ represents the inverse of the estimated length of a net and is defined as

$$il\left(h_k\right) = \frac{1}{l_{\text{est}}\left(h_k\right)}$$

where $l_{\text{est}}\left(h_k\right)$ denotes the estimated length of a net. The inverse of the estimate is used because nets whose estimated lengths are small have a high proximity and vice versa. Using the formulation in (3.3), any non-zero off-diagonal element of the proximity matrix is equal to the negative sum of the inverses of the estimated lengths of nets between i and j and any diagonal element is equal to the sum of all the inverses of

nets connected to i. Multi-terminal nets and multiple connections between cells are handled by (3.3).

As an example, the proximity matrix for the example circuit in Figure 3.11 is given in Figure 3.12. For any two cells i and j that are not connected, the $p^0_{i,j}$ entry is equal to zero. For any two cells that are connected, the $p^0_{i,j}$ entry is equal to the negative sum of the inverse of estimated lengths of the nets connecting them. As an example, cells 5 and 6 are connected by two nets each with estimated length of six. Therefore, $p^0_{5,6} = -\left(\frac{1}{6} + \frac{1}{6}\right) = -\frac{1}{3}$. Finally, the diagonal elements are the negative sum of the off-diagonal elements in the corresponding row of the matrix. For example, $p^0_{5,5} = -\left(p^0_{5,4} + p^0_{5,6}\right) = -\left(-\frac{1}{7} - \frac{1}{3}\right) = \frac{10}{21}$.

$$\mathbf{P}^0 = \begin{pmatrix} \frac{1}{4} & -\frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{4} & \frac{7}{10} & -\frac{1}{4} & -\frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{5} & 0 & \frac{12}{35} & -\frac{1}{7} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{7} & \frac{10}{21} & -\frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{3} & \frac{8}{15} & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{5} & \frac{13}{20} & -\frac{13}{40} & -\frac{1}{8} \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{13}{40} & \frac{13}{20} & -\frac{13}{40} \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{8} & -\frac{13}{40} & \frac{9}{20} \end{pmatrix}$$

Figure 3.12: Proximity matrix of the example circuit in Figure 3.11.

Step 2: AMG-Based Coarsening on the Proximity Matrix

Once the proximity matrix $\mathbf{P}^0$ has been constructed, the same coarsening Algorithm AMGC described in Section 3.2.1 is used to reduce $\mathbf{P}^0$ and construct $\mathbf{P}^1$ and the associated restriction matrix $\mathbf{W}^1_0$. The coarsening can be loosely thought of as a heuristic for selecting a maximally independent subset of cells which are the most significant in $\mathbf{P}^0$. The significance of each cell is measured by the number of cells that *strongly*

connect to it. A parameter, $\theta \in (0, 1]$, is used in the definition of the strength of a connection. Cell i is strongly connected to cell j if

$$\left|p_{i,j}^{l}\right| \geq \theta \times \max_{k \neq i} \left\{\left|p_{i,k}^{l}\right|\right\}, \; k = 1, \ldots, C^{l}, \tag{3.4}$$

where $p_{i,j}^{l}$ is the $(i, j)^{th}$ element of $\mathbf{P}^{l}$, and C^{l} is the number of cells in level l. The relationship between the parameter θ and the amount coarsening performed is complex. If θ is close to one, more aggressive coarsening will be performed. However, more aggressive coarsening can overly simplify the reduced matrix and the associated reduced circuit. An interpretation of the strength of connection criterion in the context of the proposed proximity matrix is that if a cell is in several nets with small estimated length it is likely to be a part of the reduced circuit.

Step 3: Cluster Seed Cell and Score Assignment

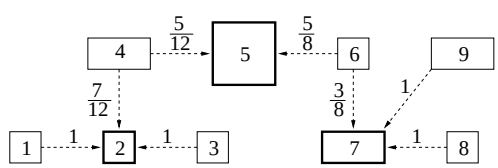
This step is similar to Step 3 of Algorithm AMGC. After coarsening the proximity matrix, the cells that are selected to form the reduced circuit become seed cells used for clustering. The cells which are not selected to form the reduced circuit are considered to be clustered with the selected seed cells. The entries of the AMG interpolation matrix $(\mathbf{W}_{l}^{l+1})^{T}$ are used to rank each seed cell that a non-selected cell connects to. A large entry in the interpolation matrix means the seed cell is representative of the non-selected cell in the reduced circuit.

The selected seed cells for the example in Figure 3.11 are cells $2, 5$, and 7. Therefore, the interpolation matrix has three columns, one for each seed cell with the first column corresponding to cell 2, etc. The number of rows is nine, which is the total number of cells and row 1 corresponds to cell 1, etc. The interpolation matrix is shown in Figure 3.13(a). Each seed cell interpolates from itself directly, so entries $w_{2,1}$, $w_{5,2}$ and $w_{7,3}$ are all one, where $w_{i,j}$ is the $(i, j)^{th}$ element of $\left(\mathbf{W}_{0}^{1}\right)^{T}$. The other non-zero entries represent the interpolation weight of non-selected cells to seed cells. As an example,

cell 4 is connected to seed cells 2 and 5. The interpolation weights of cell 4 with seed cells 2 and 5 are given by entries $w_{4,1} = \frac{7}{12}$ and $w_{4,2} = \frac{5}{12}$, respectively. To better visualize the seed cell and cluster score assignment using the interpolation matrix, a pictorial representation of $(\mathbf{W}_0^1)^T$ is presented in Figure 3.13(b). In this figure, the seed cells are shown with a bold border. Each non-selected cell has an interpolation weight to the seed cells that they connect to. These interpolation weights annotate arrows pointing from non-selected cells to seed cells in Figure 3.13(b).

$$(\mathbf{W}_0^1)^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{7}{12} & \frac{5}{12} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{5}{8} & \frac{3}{8} \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) Interpolation Matrix $(\mathbf{W}_0^1)^T$



(b) Pictorial representation

Figure 3.13: The interpolation matrix shown in matrix form and pictorial form for the example circuit in Figure 3.11. In the pictorial representation, the selected seed cells are shown in bold and arrows are annotated with the interpolation weights of non-selected cells.

Note that the interpolation weights are different from the weights in the example in Section 3.2. For example, the weight between cell 4 and seed cells 2 and 5 were equal in the previous example, but in AMGC-LE, it is found out that cell 4 has more proximity to seed cell 2.

Step 4: Final Cluster Formation

The final step of Algorithm AMGC-LE is to cluster non-selected cells to the seed cell which they interpolate from most strongly. In Step 4 of AMGC-LE, a cluster is finalized as long as the area of the cluster is not greater than five times the average standard cell area and its interpolation weight is more than 0.9. The cluster area is restricted to prevent the formation of macro-sized clusters which require special treatment in placement algorithms. The interpolation weight threshold is set high to ensure that only the best quality clusters are formed. Each cluster meeting the constraints is given a physical shape in the clustered circuit with a height equal to the maximum height of all of its cells and a width equal to the sum of the widths of its cells. There are differences in the maximum area and interpolation weight thresholds compared to Step 4 of Algorithm AMGC. The main reason for the changes is to support the length-driven unclustering proposed in the following section that complements AMGC-LE.

The final clusters for the example in Figure 3.11 are shown with dashed lines in Figure 3.14. The clusters are formed by clustering each non-selected cell with the seed cell with which it has the maximum interpolation weight.

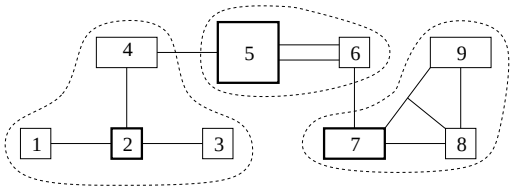


Figure 3.14: The final clusters for the example circuit in Figure 3.11 shown with dashed lines.

3.4.2 Length-Driven Unclustering Technique

A clustering algorithm alone is not enough to ensure success of a clustering algorithm used for placement. Just as important is the algorithm for unclustering, or determining

the locations of cells within a cluster after the cluster's location has been determined. The description of this step is rarely mentioned in the literature of clustering algorithms.

In this work, a length-driven unclustering algorithm is proposed. The main benefit of using the proposed technique is that a legal solution is preserved after unclustering. Global placement algorithms often produce legal solutions before performing detailed placement. A new unclustering technique which preserves the legality of the globally-placed solution improves the runtime in multilevel placement. With the legality restriction, the problem of unclustering reduces to ordering the cluster's cells. The proposed technique finds the ordering by solving a relaxed length-driven minimization and using the result to determine the ordering of the cells. In addition, the combination of preserving legality and restricting cluster areas means that the row in which the unclustered cells will be placed is also preserved. Consequently, the vertical components of the lengths can be ignored. This helps to reduce the disruption in the wire length before and after unclustering which is a significant problem for multilevel placement as discussed in [41].

The locations of cells, $\mathbf{f}_{C_i}$, in cluster C_i are determined by minimizing the quadratic matrix length objective

$$\mathbf{f}_{C_i}^T \mathbf{U} \mathbf{f}_{C_i} + \mathbf{t}^T \mathbf{f}_{C_i} + v, \tag{3.5}$$

where $\mathbf{U}$ is a weighted connectivity matrix between the cells of the cluster, i.e. $i, j \in C_i$, and is defined as

$$u_{i,j} = \begin{cases} \sum\limits_{h_k \in H_{i,j}} -\frac{1}{|h_k|}, & i \neq j,\ H_{i,j} \neq \emptyset, \\ \sum\limits_{h_k \in H_{i,i}} \frac{1}{|h_k|}, & i = j, \\ 0, & \text{otherwise}, \end{cases} \tag{3.6}$$

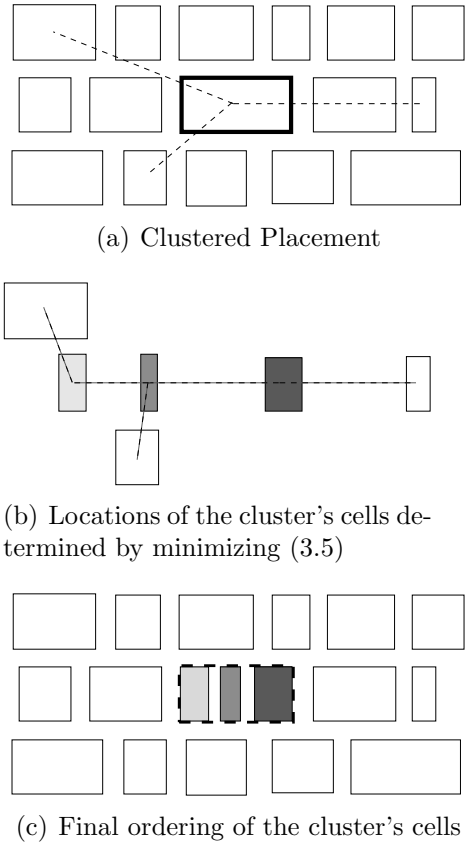
where $|h_k|$ is the degree of net h_k, i.e. the number of cells in h_k. The vector $\mathbf{t}$ contains the weighted horizontal cell locations of cells outside of the cluster that each clustered

cell connects to, i.e. $i \in C_i$ and $j \notin C_i$ defined as

$$t_i = \begin{cases} \sum\limits_{h_k \in H_{i,j}} -\frac{f_j}{|h_k|}, & i \neq j,\ H_{i,j} \neq \emptyset, \\ 0, & \text{otherwise}, \end{cases} \tag{3.7}$$

where f_j is the location of the cell or the cluster that a cell belongs to. The scalar v represents the length of all the nets not involving the cells in C_i and can be ignored. The resulting system of linear equations is solved using a direct Gaussian elimination method. Because the number of cells in a cluster is small, usually three or less, methods which are more efficient for solving large linear systems are not required. After minimizing (3.5), the cells are inserted into the area occupied by the cluster in the order of their locations in $\mathbf{f}_{C_i}$. An illustration of the length-driven unclustering is given in Figure 3.15. In Figure 3.15(a), a clustered placement is given. The highlighted cluster is the focus of the example and contains three cells (not shown in 3.15(a)). The cluster is connected to three other cells via degree-two nets, shown by dashed lines in the illustration. The cells contained in the cluster are shown in shades of grey in Figure 3.15(b). The three external connections as well as the two internal connections are used to determine the non-zero values in the matrix $\mathbf{U}$. In this example all of the values for $|h_k|$ are equal to 2, the degree of each net. The locations of the cells not in the cluster, shown in white, are used in forming the vector $\mathbf{t}$. The locations of the cluster's cells in Figure 3.15(b) are determined by minimizing (3.5). These locations are used to determine the order of the cells inside of the cluster's area to complete the unclustering, illustrated in Figure 3.15(c). Because the unclustered cells remain inside of the cluster's area, the placement remains legal.

This technique does not take advantage of the locations of already unclustered cells as it iterates through the clusters. However, this also means that the technique is not sensitive to the order in which the clusters are visited. Furthermore, because the clusters are restricted to have an area of less than five times the average standard cell



(a) Clustered Placement

(b) Locations of the cluster's cells determined by minimizing (3.5)

(c) Final ordering of the cluster's cells

Figure 3.15: An illustration of the length-driven unclustering technique.

area, the additional benefit of using the locations of already unclustered cells and a good ordering of the clusters is expected to be small.

3.5 AMGC-LE Experimental Results and Analysis

This section benchmarks the performance of the two proposed length-driven algorithms, AMGC-LE and length-driven unclustering. The AMGC-LE algorithm is designed to improve upon the placement performance of AMGC. The length-driven unclustering proposed in Section 3.4.2 is also designed to accomplish improved placement results. Therefore, all of the experiments performed in this section are comparing against several state-of-the-art academic placers. In comparison to the experimental placement

setup of the last section, some of the underlying parameters of AMGC-LE are configured differently. The choices made are explained before each experiment is performed. This is to allow a more aggressive clustering which can translate into better placement results in terms of wire length and runtime.

The ICCAD04 benchmarks are used for performing placement experiments, and in additiona, the ISPD05 benchmark suite [42] is used to show the scalability of the AMGC-LE algorithm and the proposed length-driven unclustering. The combination of the two length-driven techniques and the legality-preserving nature of the unclustering make a combination capable of dealing with such formidable benchmarks. The statistics of these benchmark circuits are presented in Table 3.6 and in the same format as Table 3.1. The benchmark circuit bigblue4 is not included in the experiments due to the memory limitations of the testing platform.

Table 3.6: Statistics of ISPD05 Benchmarks.

Circuit	# nets	# cells	$\|h_{max}\|$	$\|c_{max}\|$
adaptec1	221,142	211,447	2,271	448
adaptec2	266,009	255,023	1,935	620
adaptec3	466,758	451,650	3,713	1224
adaptec4	515,951	496,045	3,974	416
bigblue1	284,479	278,164	2,621	388
bigblue2	577,235	557,866	11,869	119
bigblue3	1,123,170	1,096,812	7,623	1692

This section is organized as follows: placement experiments using AMGC-LE are first presented in Section 3.5.1. In Section 3.5.2, results of the same placement experiments are given using the proposed length-driven unclustering. Finally, a comparison of AMGC-LE with and without using length-driven unclustering to existing clustering techniques is presented in Section 3.5.3.

3.5.1 Placement Performance of AMGC-LE

To evaluate the proposed AMGC-LE algorithm, its effectiveness in the context of multilevel circuit placement using multiple placers is illustrated. The experiments are performed using three high-quality academic placers: Capo 10.5 [43], Fastplace 3.0 [44], and mPL6 [40]. Fastplace and mPL are analytical placers and have clustering algorithms embedded inside their placement algorithms. To better evaluate the performance of the proposed technique, the internal clustering of these placers are disabled in the clustering experiments. If the internal clustering is enabled, it will not be clear if the proposed clustering technique or the internal clustering is the cause of any benefits. Comparisons of the proposed technique with existing clustering techniques, including the technique used internally by Fastplace 3.0 and mPL6, are presented in Section 3.5.3. Capo is a partitioning-based placer and does not use internal clustering. Capo is, however, non-deterministic so the average of ten runs of each experiment are reported when results for Capo are given. The experiments are performed on a 64-bit dual core AMD Opteron system running Linux with 8GB of RAM. In addition, the parameter θ in (3.4) is set to 0.1 in the experiments as well as $\omega_{\min}$ being set to 0.9. This combination allows many strong connections to be exposed, low θ, while only the strongest clusters are formed, high $\omega_{\min}$.

Wire length and runtime results of the experiment are presented in Table 3.7(a) and (b), respectively. In both tables, Column 1 identifies the circuit that is placed. Columns 2-4, 5-7, and 8-10 show the results for Capo, Fastplace, and mPL, respectively. Each placer is used in three modes: baseline without clustering, one-level AMGC-LE clustering with length-driven unclustering, and two-level AMGC-LE clustering with length-driven unclustering. The columns showing one- and two-level AMGC-LE are given in terms of the percentage improvement over the baseline. In this representation, positive percentages represent improvements.

Table 3.7: Placement wire length and runtime results comparing one- and two-level AMGC-LE clustering with length-driven unclustering to baseline placements without using clustering with three placers on the ICCAD04 benchmark suite.

(a) HPWL

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^5$)	(%)	(%)	($\times 10^5$)	(%)	(%)	($\times 10^5$)	(%)	(%)
ibm01	25	**3.1**	**6.4**	24	-1.8	**1.3**	24	**4.6**	**4.2**
ibm02	51	**4.4**	**5.4**	54	**3.1**	**4.6**	52	**1.3**	**1.1**
ibm03	77	**1.8**	**9.8**	80	**5.1**	**6.8**	82	-7.0	-3.6
ibm04	93	**14.8**	**15.7**	86	**7.9**	**7.7**	109	-5.7	-4.4
ibm05	103	**1.1**	**2.1**	101	**0.7**	**0.5**	93	-5.2	-5.1
ibm06	67	**0.6**	**3.8**	99	**38.1**	**33.9**	88	**5.8**	**2.7**
ibm07	126	**9.3**	**11.9**	123	**7.4**	**12.2**	124	-3.1	-4.0
ibm08	137	**7.9**	**8.1**	147	-4.1	**10.1**	211	**4.7**	**4.2**
ibm09	145	**4.4**	**6.6**	155	**8.6**	**8.0**	189	**2.9**	**1.7**
ibm10	318	**4.0**	**4.9**	362	**10.8**	**12.2**	363	**2.0**	**1.8**
ibm11	210	**4.1**	**7.0**	225	**11.0**	**10.1**	243	-0.1	-2.6
ibm12	413	**10.9**	**14.5**	410	**11.4**	**14.4**	461	-1.1	**1.1**
ibm13	266	**3.4**	**7.6**	273	**12.1**	**11.8**	324	-0.4	**1.1**
ibm14	392	**2.6**	**4.0**	478	**14.5**	**21.7**	824	**6.5**	**6.6**
ibm15	544	**6.1**	**5.6**	577	**8.6**	**12.2**	1001	-2.9	-5.0
ibm16	629	**5.6**	**6.4**	680	**11.5**	**11.5**	931	**6.7**	**5.7**
ibm17	737	**2.7**	**3.8**	810	**9.3**	**12.2**	1144	**7.5**	**7.2**
ibm18	458	**2.1**	**3.3**	574	**19.0**	**19.8**	885	**11.0**	**10.7**
Average	-	**4.9**	**7.1**	-	**9.6**	**11.7**	-	**1.5**	**1.3**

(b) Runtime

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	(s)	(%)	(%)	(s)	(%)	(%)	(s)	(%)	(%)
ibm01	193	-3	-75	26	-5	-62	109	-69	-152
ibm02	329	-24	-112	46	-14	-72	236	-91	-171
ibm03	487	-21	-105	46	-20	-82	221	-108	-210
ibm04	512	-23	-110	51	-17	-90	250	-87	-195
ibm05	411	-25	-124	36	-35	-81	174	-118	-183
ibm06	580	-19	-115	96	29	-6	390	-83	-197
ibm07	920	-21	-105	96	-9	-82	387	-100	-198
ibm08	941	-20	-121	99	-31	-158	1109	-105	-197
ibm09	1198	-29	-119	122	-6	-66	898	-136	-221
ibm10	1868	-12	-120	286	4	-62	1450	-126	-210
ibm11	1834	-30	-121	135	-66	-139	1084	-84	-193
ibm12	1996	-22	-124	282	-12	-52	1517	-115	-222
ibm13	2387	-16	-120	239	-43	-118	1235	-110	-218
ibm14	3327	-29	-129	530	-42	-121	2189	-93	-213
ibm15	5781	-30	-142	624	-57	-197	3969	-101	-247
ibm16	4917	-35	-132	722	-75	-217	5457	-57	-118
ibm17	5286	-31	-135	1133	-33	-101	2788	-131	-228
ibm18	4215	-39	-142	1589	3	-57	3106	-125	-257
Average	-	-24	-119	-	-24	-98	-	-102	-202

The results in Table 3.7(a) show that AMGC-LE is effective at improving wire length on average for all placers using one or two levels of clustering. A maximum improvement of 38.1% is achieved using one-level AMGC-LE and Fastplace. Using one-level AMGC-LE, every circuit for Capo, 16 circuits for Fastplace, and 10 circuits for mPL have improved wire length when compared to the baseline. When two-level AMGC-LE is used, every circuit for Capo and Fastplace, and 12 circuits for mPL are improved.

The runtime results show that using the one-level scheme for mPL and the two-level scheme for all placers double the total runtime. For a placer, such as Fastplace, the increase in runtime may be more acceptable because it requires much less time than other placers. However, the increases for Capo and particularly for mPL are more prominent. It should be noted that mPL does not offer an option to perform only detailed placement requiring full global and detailed placement to be performed between each level, which accounts for the significant increases compared to the other two placers. The experiment performed in the following section will illustrate how to improve the runtime results while maintaining, and even improving, the wire length results.

Upon comparing the results using AMGC-LE with mPL to the results using AMGC in section 3.3.2, it can be seen that the wire length for some benchmarks has decreased while some have increased. This can partly be explained by the fact that the length estimates produced by the technique in [26] have the poorest correlation when using mPL [35]. In [35], a predictor-corrector clustering framework based on length estimation is developed. The application of this framework resulted in major improvements in wire length in the final placement results. The same framework can be applied to AMGC-LE as it is a length-based algorithm. Further analysis comparing the two techniques are given in Section 3.5.3.

An interesting result arising from the experiment is that the placements resulting from using the AMGC-LE framework are more correlated with the length estimates than the baseline placements. The correlation improvement is small, with a maximum improvement of 6% for Capo using two-level AMGC-LE, but does suggest that the estimates are directing the placement.

In order to assess the scalability of the proposed clustering and unclustering algorithms, the same experiments are performed on the ISPD05 circuits. These circuits are bigger than the ICCAD04 benchmarks and have significantly larger maximum net degree. The results are tabulated in Table 3.8. The version of Capo used in the ex-

Table 3.8: Placement wire length and runtime results comparing one- and two-level AMGC-LE clustering with length-driven unclustering to baseline placements without using clustering with three placers on the ISPD05 benchmark suite.

(a) HPWL

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^6$)	(%)	(%)	($\times 10^6$)	(%)	(%)	($\times 10^6$)	(%)	(%)
adaptec1	91	**1.5**	**2.3**	87	**8.6**	**8.7**	81	-1.1	-1.1
adaptec2	120	**14.2**	**14.5**	108	**4.0**	**9.6**	97	-0.1	-0.1
adaptec3	254	**6.4**	**5.7**	287	**15.4**	**17.2**	224	**0.6**	**0.6**
adaptec4	-	-	-	230	**8.6**	**12.7**	195	-0.5	-0.5
bigblue1	114	**4.0**	**3.3**	107	**5.5**	**7.8**	101	**0.0**	**0.0**
bigblue2	167	**2.7**	**2.3**	181	**8.5**	**12.3**	152	-0.3	-0.3
bigblue3	439	**3.6**	**6.4**	663	**37.0**	**40.1**	492	**1.7**	**1.7**
Average	-	**5.4**	**5.7**	-	**12.5**	**15.5**	-	**0.0**	**0.0**

(b) Runtime

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^2$s)	(%)	(%)	($\times 10^2$s)	(%)	(%)	($\times 10^2$s)	(%)	(%)
adaptec1	30	-24	-73	5	-79	-109	18	-80	-109
adaptec2	44	2	-49	13	-15	-4	17	-134	-211
adaptec3	81	-32	-107	28	-136	-77	65	-89	-135
adaptec4	-	-	-	21	-208	-184	45	-258	-199
bigblue1	54	11	-36	6	-158	-141	18	-128	-173
bigblue2	92	-64	-100	23	-299	-278	49	-235	-286
bigblue3	254	-24	-49	139	-45	-78	123	-323	-365
Average	-	-22	-69	-	-134	-125	-	-178	-211

periment could not place the adaptec4 benchmark on the test platform so its entries in Table 3.8 are not included. In general, the results resemble those of Table 3.7 with Capo and Fastplace obtaining significant wire length improvement for both one- and two-level AMGC-LE. Meanwhile, only a negligible average improvement is achieved by mPL. The runtime increases for all placers with mPL having the largest increase in runtime, which is not surprising given that mPL must perform global placement in addition to detailed placement at each level. Capo has a more modest increase in runtime compared to the other two placers, on average. In this case, the placements for mPL are nearly identical whether one or two levels of AMGC-LE clustering are performed. This means that the effects of the second level of clustering are almost entirely removed by performing global placement on the unclustered circuit.

3.5.2 Length-Driven Unclustering as a Detailed Placer

In this section, the same multilevel experiments are performed with the difference of not performing any placement apart from placing the bottommost clustered circuit. The circuit is placed at the bottom level using the global and detailed placement functions of each placer (if the two are distinguished). As a result, the placement at the bottom level is legal, i.e. free from overlaps. To improve upon the runtime results given in the previous section, it is proposed to use the length-driven unclustering technique as the only refinement between levels. The placement will preserve its legality using the technique mentioned in Section 3.4.2. The results of this experiment are given in Table 3.9.

The format of the Tables 3.9(a) and (b) is the same as Table 3.7. It can be seen that all placers' wire lengths are improved on average using either one- or two-level AMGC-LE. The best wire length improvement is again for Fastplace with one-level AMGC-LE on ibm06 with a 35.8% improvement. Using one-level AMGC-LE 16, 15, and 14 circuits

Table 3.9: Placement wire length and runtime results comparing one- and two-level AMGC-LE clustering with length-driven unclustering without detailed placement to baseline placements with three placers on the ICCAD04 benchmark suite.

(a) HPWL

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^5$)	(%)	(%)	($\times 10^5$)	(%)	(%)	($\times 10^5$)	(%)	(%)
ibm01	25	-1.0	-1.7	24	-7.7	-4.9	24	-3.2	-3.4
ibm02	51	**1.6**	**1.2**	54	**1.3**	**2.5**	52	-3.8	-3.8
ibm03	77	**4.2**	**4.6**	80	**1.3**	**3.4**	82	**5.0**	**5.1**
ibm04	93	**10.6**	**10.4**	86	**4.0**	**4.7**	109	**11.3**	**15.9**
ibm05	103	**0.3**	-1.1	101	-1.9	-2.7	93	-10.1	-13.4
ibm06	67	**1.3**	**0.3**	99	**35.8**	**31.1**	88	**15.4**	**17.1**
ibm07	126	**10.2**	**9.9**	123	**3.3**	**7.0**	124	-4.4	**1.2**
ibm08	137	**2.6**	**1.9**	147	-3.0	**3.3**	211	**13.0**	**20.0**
ibm09	145	**0.6**	**1.5**	155	**4.3**	**3.9**	189	**11.9**	**15.9**
ibm10	318	-1.0	-0.5	362	**7.4**	**9.0**	363	**3.2**	**5.7**
ibm11	210	**1.9**	**1.9**	225	**7.6**	**6.3**	243	**4.3**	**4.1**
ibm12	413	**9.6**	**9.6**	410	**7.4**	**10.4**	461	**6.4**	**6.9**
ibm13	266	**3.3**	**3.4**	273	**7.7**	**7.3**	324	**9.7**	**14.3**
ibm14	392	**2.4**	**1.0**	478	**11.5**	**18.4**	824	**30.0**	**35.2**
ibm15	544	**2.6**	-0.1	577	**5.0**	**8.6**	1001	**23.3**	**23.4**
ibm16	629	**5.1**	**2.5**	680	**8.5**	**7.9**	931	**16.8**	**21.8**
ibm17	737	**1.9**	**1.2**	810	**7.0**	**9.0**	1144	**13.0**	**21.5**
ibm18	458	**0.9**	**0.0**	574	**15.3**	**15.8**	885	**18.8**	**28.9**
Average	-	**3.2**	**2.6**	-	**6.4**	**7.8**	-	**8.9**	**12.0**

(b) Runtime

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	(s)	(%)	(%)	(s)	(%)	(%)	(s)	(%)	(%)
ibm01	193	16	12	26	-12	-22	109	47	54
ibm02	329	-8	2	46	1	-28	236	27	38
ibm03	487	-5	5	46	-16	-36	221	10	32
ibm04	512	-1	-4	51	-13	-42	250	39	40
ibm05	411	-1	-1	36	-29	-63	174	-2	32
ibm06	580	12	5	96	34	15	390	39	51
ibm07	920	7	8	96	-8	-57	387	28	35
ibm08	941	6	-11	99	-22	-110	1109	62	67
ibm09	1198	-8	1	122	-3	-42	898	55	62
ibm10	1868	3	-3	286	8	-11	1450	8	38
ibm11	1834	-1	-5	135	-64	-94	1084	56	56
ibm12	1996	-1	-7	282	-3	-17	1517	29	39
ibm13	2387	-7	-6	239	-32	-81	1235	54	56
ibm14	3327	-2	-14	530	-51	-125	2189	27	49
ibm15	5781	0	-22	624	-38	-188	3969	32	43
ibm16	4917	-4	-11	722	-58	-167	5457	50	53
ibm17	5286	-4	-26	1133	-40	-84	2788	3	12
ibm18	4215	-8	-36	1589	2	-43	3106	2	14
Average	-	0	-6	-	-19	-66	-	32	43

improve for Capo, Fastplace, and mPL, respectively. Improvement is observed in 14, 16, and 15 circuits for Capo, Fastplace, and mPL, respectively, when using two-level AMGC-LE. When compared to the results in Section 3.5, the average wire length for Capo and Fastplace degrades while mPL significantly improves. This improvement is because mPL has no option to perform just detailed placement. It performs global and detailed placement using the clustered placement as the starting point. During this process, the clusters become dispersed. Therefore, when the proposed length-driven unclustering is used as a detailed placer, the benefits of clustering and length-driven unclustering are preserved.

The runtime results show significant improvements over the results in Section 3.5. When compared to the baseline, Capo is not significantly changed while Fastplace is degraded by 19% and 66% for one- and two-level AMGC-LE. The increases in runtime for Fastplace may be tolerated because of its comparatively low runtimes. However, mPL runtimes are improved significantly with more improvement seen with two-level AMGC-LE, nearly cutting the average runtime by half.

In order to assess the scalability of the proposed length-driven unclustering as a detailed placer, the same experiments are performed on the ISPD05 circuits. The results are tabulated in Table 3.10. The results follow similar trends as those of Table 3.9. All placers achieve wire length improvements and significantly improved running times compared to the results of Table 3.8 which are obtained by using each placers' detailed placement algorithm in between clustering levels. In this case, Capo achieves an overall runtime improvement compared to the baseline with 3.6% and 2.4% average improvements in wire length when performing one and two levels of AMGC-LE clustering, respectively. mPL achieves substantial improvements in wire length with modest increases in average running time. Finally, for Fastplace, significant average wire length improvements are observed for both one and two levels of AMGC-LE clus-

Table 3.10: Placement wire length and runtime results comparing one- and two-level AMGC-LE clustering with length-driven unclustering without detailed placement to baseline placements with three placers on the ISPD05 benchmark suite.

(a) HPWL

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^6$)	(%)	(%)	($\times 10^6$)	(%)	(%)	($\times 10^6$)	(%)	(%)
adaptec1	91	-1.4	-2.7	87	**1.8**	-1.6	81	**5.3**	**3.1**
adaptec2	120	**12.6**	**10.8**	108	-0.2	**3.6**	97	**13.7**	**12.9**
adaptec3	254	**4.0**	**2.0**	287	**12.0**	**10.8**	224	**16.4**	**15.3**
adaptec4	-	-	-	230	**4.9**	**7.4**	195	**8.0**	**7.6**
bigblue1	114	**1.1**	**0.1**	107	-1.7	-2.6	101	-0.4	-2.3
bigblue2	167	**1.0**	-0.4	181	**0.8**	**2.0**	152	**3.2**	**1.6**
bigblue3	439	**4.3**	**4.6**	663	**35.7**	**37.6**	492	**30.4**	**29.1**
Average	-	**3.6**	**2.4**	-	**7.6**	**8.2**	-	**10.9**	**9.6**

(b) Runtime

Circuit	Capo			Fastplace			mPL		
		AMGC-LE			AMGC-LE			AMGC-LE	
	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl	Baseline	1Lvl	2Lvl
	($\times 10^2$s)	(%)	(%)	($\times 10^2$s)	(%)	(%)	($\times 10^2$s)	(%)	(%)
adaptec1	30	21	17	5	-61	-65	18	32	3
adaptec2	44	34	41	13	-14	10	17	-34	-59
adaptec3	81	19	29	28	-47	-73	65	-22	16
adaptec4	-	-	-	21	-203	-195	45	-59	-52
bigblue1	54	39	42	6	-95	-74	18	-2	-36
bigblue2	92	-2	-16	23	-128	-114	49	-71	-91
bigblue3	254	11	-4	139	-29	-68	123	-90	-75
Average	-	20	18	-	-82	-83	-	-35	-42

tering, while the runtime has degraded which is acceptable considering the relatively low runtimes of Fastplace.

In the end, the circuit designers can decide which of the placement flows presented in Sections 3.5 and 3.5.2 is preferred for their application. Both improve wire length but to varying degrees depending on the choice of placer. When using a placer like Capo or Fastplace, the best wire length is obtained by an increase in runtime, while mPL achieves the best results with the fast detailed placement flow. If runtime is the most urgent concern, the proposed scheme of using length-driven unclustering as a detailed placer is the best option.

3.5.3 Comparison to Existing Clustering Algorithms

The proposed AMGC-LE clustering algorithm with length-driven unclustering is compared to other existing clustering algorithms in this section. Each technique is evaluated in terms of after placement wire length and total runtime using the mPL6 placer [40]. The clustering algorithms compared to are heavy-edge matching (HEM) [11], best-choice (BC) [5], and the AMG clustering (AMGC) algorithm in Section 3.2. It is worth mentioning that best-choice is the clustering algorithm used internally by Fastplace 3.0 and mPL6. Placement is first performed without any clustering to establish a baseline. Then, each of the algorithms is used to produce a clustered circuit which is placed by mPL. The clustered placement is then unclustered and detailed placement is performed by mPL or by using length-driven unclustering as a detailed placer. Results of the experiment are given in Table 3.11.

The percentage improvement for AMGC-LE in a regular clustered placement flow, as described in Section 3.5, is given under the subheading Regular. The column labeled Fast Detailed refers to the results using only length-driven unclustering as a detailed placer, described in Section 3.5.2. In terms of wire length, the regular AMGC-LE

Table 3.11: Placement wire length and runtime results comparing one-level of existing clustering algorithms and AMGC-LE with and without length-driven unclustering to baseline placements with mPL6 on the ICCAD04 benchmark suite.

(a) HPWL

Circuit	Baseline ($\times 10^6$)	HEM (%)	BC (%)	AMGC (%)	AMGC-LE 1Lvl Regular (%)	AMGC-LE 1Lvl Fast Detailed (%)
ibm01	2.4	5.9	7.1	5.7	4.6	-3.2
ibm02	5.2	-6.3	-1.6	-9.9	1.3	-3.8
ibm03	8.2	2.2	6.1	5.8	-7	5
ibm04	10.9	0.4	-1.3	-1.9	-5.7	11.3
ibm05	9.3	-5.4	-5.3	-5.4	-5.2	-10.1
ibm06	8.8	10	12.5	11.9	5.8	15.4
ibm07	12.4	1.4	-0.1	2.2	-3.1	-4.4
ibm08	21.1	-2.7	-2.9	-2.9	4.7	13
ibm09	18.9	5.1	-2	3.7	2.9	11.9
ibm10	36.3	1.9	1.7	2.1	2	3.2
ibm11	24.3	-2.6	-4.6	-1.8	-0.1	4.3
ibm12	46.1	-5	-5.9	-4.2	-1.1	6.4
ibm13	32.4	-1.2	0.3	1.7	-0.4	9.7
ibm14	82.4	6.9	7.4	7.4	6.5	30
ibm15	100.1	-7.5	-3.3	1.7	-2.9	23.3
ibm16	93.1	2.5	2.4	2.6	6.7	16.8
ibm17	114.4	5.7	5.9	4.9	7.5	13
ibm18	88.5	11.6	12	12	11	18.8
Average	-	1.2	1.5	1.9	1.5	8.9

(b) Runtime

Circuit	Baseline (s)	HEM (%)	BC (%)	AMGC (%)	AMGC-LE 1Lvl Regular (%)	AMGC-LE 1Lvl Fast Detailed (%)
ibm01	150	-20	-20	-14	-69	47
ibm02	324	-39	-41	-38	-91	27
ibm03	281	-67	-52	-85	-108	10
ibm04	344	-40	-28	-31	-87	39
ibm05	232	-45	-29	-22	-118	-2
ibm06	475	-62	-32	-49	-83	39
ibm07	488	-51	-43	-49	-100	28
ibm08	1468	-29	-44	-26	-105	62
ibm09	1134	-26	-26	-24	-136	55
ibm10	1762	-45	-49	-45	-126	8
ibm11	1501	-15	-1	-4	-84	56
ibm12	2009	-41	-57	-36	-115	29
ibm13	1456	-28	-26	-23	-110	54
ibm14	2547	-39	-33	-29	-93	27
ibm15	4208	-48	-26	-23	-101	32
ibm16	5821	-33	-35	-26	-57	50
ibm17	3025	-52	-53	-59	-131	3
ibm18	3672	-47	-49	-49	-125	2
Average	-	-40	-36	-35	-102	31

scheme is competitive with the existing algorithms performing better in some circuits and worse in others. On the other hand, the runtime for regular AMGC-LE degrades more than the other clustering techniques. This is because the runtime includes the time to perform pre-placement length estimation and the algorithm is implemented in the MATLAB environment [45]. Therefore, the runtime could be more competitive if the procedure were implemented in a more efficient way. However, the fast detailed variant of AMGC-LE is a clear winner in terms of average wire length and runtime improvement.

3.6 Summary

In this chapter, two clustering algorithms based on AMG are proposed. The first proposed algorithm uses AMG and circuit connectivity matrix for cluster formation. Furthermore, the complexity of the proposed algorithm is analyzed and proven to have linear-time complexity. The new algorithm is tested to illustrate its efficiency, scalability, quality, and empirical verification of the linear-time complexity.

The second AMG-based clustering technique uses length estimates instead of circuit connectivity to produce clustering solutions. The results obtained from this clustering are shown to directly reduce the wire length of a circuit.

Finally, a physical unclustering technique is proposed with the benefits of reducing wire length and preserving legality. Because of its legality-preserving nature, the technique eliminates the need to use detailed placement between levels in the framework. These techniques are shown to be effective in reducing wire length and can also be used to benefit runtime.

Part III

Clock Tree Synthesis Problems

Chapter 4

Background Material for Part III

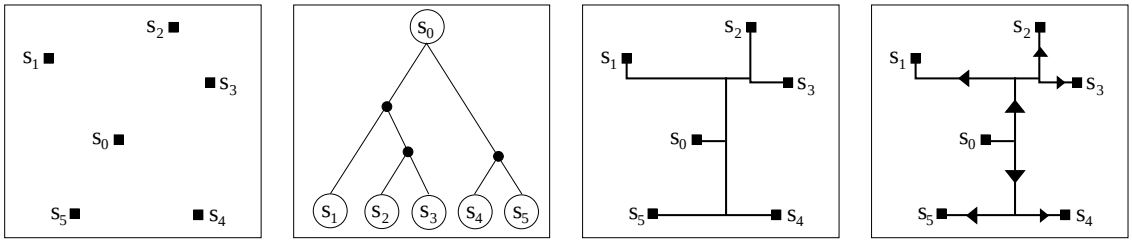
4.1 Introduction

High-performance circuits make use of a signal to synchronize the flow of data through the circuit. This signal is called a clock. One of the most important components of a high-performance circuit is the clock tree. The clock tree is the network responsible for propagating the clock signal from the clock source, generated by an oscillator, to the clock input of all of the synchronous elements in a circuit, e.g. flip-flops, referred to as clock sinks. The clock tree affects the performance of the circuit in several ways. First, the maximum frequency of the circuit is limited by the clock skew, the maximum difference in arrival times of the clock signal at the clock sinks. Second, the clock tree is a major consumer of power in a circuit, up to 50% [46], because it is a large network that charges and discharges every clock period. The design of the clock tree is complicated in modern designs by the increasing scale of the number of sinks in designs. Furthermore, as manufacturing processes continue to scale down the feature sizes of fabricated circuit elements, the effect of process variations become critical.

This chapter presents material relevant to the contributions proposed in Chapters 5 and 6. Both chapters have the design of clock trees, or clock tree synthesis (CTS), in common. The main stages of CTS are described in Section 4.2. In Section 4.3, extra attention is given to the topic of buffer sizing for CTS, which is the problem dealt with in Chapter 6. The work in Chapter 5 requires knowledge on the topic of parallel computing. Relevant material on the topic of parallel computing is given in Section 4.4. Optimization techniques used in Chapter 6 are presented in Sections 4.5, namely, geometric programming and robust optimization.

4.2 Clock Tree Synthesis

The input to a CTS flow is a clock problem instance which defines the locations of the clock source, s_0, and clock sinks, $s_1, \ldots, s_N$, along with the input capacitance of each sink. The output is a network which can distribute the clock signal from the source to the sinks. Clock tree synthesis is typically performed in three steps: topology generation, tree embedding, and buffer insertion [47]. In topology generation, a topology tree is generated with the clock sinks as leaves and clock source as the root. It is common for the topology tree to be binary and intermediate nodes, known as *Steiner nodes*, are created during this step. Minimizing the expected amount of wire needed to turn the topology into a physical route is usually the goal of topology generation. In tree embedding, the topology tree is routed on the layout area. The route aims to reduce skew and wire length in the process. In buffer insertion, buffers are added into the route to ensure that the clock signal has desirably short transition times. Excessive buffering causes the power to be unacceptable, so a compromise is required. The location of the buffers is an important consideration, but so too is the size of the buffers being placed. The topic of buffer sizing is explained in further detail in Section 4.3. Each step is represented pictorially in Figure 4.1.



(a) Clock problem instance (b) Clock topology tree (c) Embedded clock tree (d) Buffered clock tree

Figure 4.1: Steps in performing CTS on a given problem instance.

4.2.1 Topology Generation

Methods for topology generation can be classified as top-down, i.e. partitioning, or bottom-up, i.e. clustering. The method of means and medians (MMM) creates a topology tree by recursively partitioning the sinks based on their median location, alternating between x- and y-coordinates [48]. MMM is used in many algorithms for performing clock tree synthesis of 3D circuits, e.g. [49, 50]. This balances the number of sinks between partitions with the hope of reducing wire length, but the balanced bipartition (BB) technique proposed in [51] attempts to balance the capacitance in each partition. The effect of balancing the capacitance is that the delay to the sinks in each level of the topology tree is more balanced as well. This leads to lower clock skew. A commonly used clustering method (CL) proposed in [52] forms a nearest-neighbor graph on the sinks and forms clusters of pairs of sinks with minimum distance between them. This process is repeated with the newly-formed set of clusters until a single cluster contains all of the sinks and the topology tree is complete.

4.2.2 Tree Embedding

The second half of MMM defines how to perform embedding in a bottom-up fashion by connecting siblings in the topology tree with a straight wire. At the next level up in the tree, new wires connect pairs of existing wires at the mean, or the middle, of the existing wires. By far the most common technique for embedding is an approach which balances the estimated delays of subtrees being merged, e.g. [51, 53]. These techniques are collectively called zero skew techniques. The techniques also work in a bottom-up fashion but instead of connecting to the middle of wires a connection is made where the delays to the sinks of each subtree are equal. The idea can be intuitively understood by considering two sibling subtrees with different delays to their sinks. When connecting the two subtrees, the signal should enter closer to the slower

of the two subtrees. By performing a top-down step before the zero skew merging, the techniques can achieve minimum wire length with respect to the delay model being considered. If this additional planning step is taken, the method is called a deferred-merge-embed (DME) method.

4.2.3 Buffer Insertion

After embedding the tree, buffers need to be added to satisfy transition time requirements. One method is to determine how much capacitive load a buffer can drive until a constraint is violated, which can be calculated off-line by simulation, and insert buffers whenever that limit is reached [54]. This technique can be performed in bottom-up or top-down fashion. So long as the tree is balanced, the number of buffers along each path will be balanced maintaining low skew.

Another method requires a set of candidate buffer locations and uses dynamic programming to obtain a buffered tree with minimum delay, e.g. [55, 56]. This tends to minimize skew as a side effect [54].

4.3 Clock Tree Buffer Sizing

Buffer insertion is required in clock trees to prevent signal degradation and satisfy slew constraints. The performance of the network can be further optimized by sizing the buffers in the clock tree. The process technology provides a library of available buffers with varying properties, such as inverting or non-inverting, minimum buffer sizes, resistance and capacitance. In a typical buffer sizing problem, the number and locations of buffers are set, and the widths and lengths of buffers are determined while minimizing maximum delay, skew, or power.

The problem of buffer sizing has been studied for general signal nets [57] and closed form solutions for optimal buffer and wire sizing are available for a single wire segment

[58]. However, these solutions are aimed at minimizing maximum delay, not skew or power. In addition, the solutions can be only applied to a single segment and not an entire tree.

As mentioned in Section 4.2.3, a dynamic programming method, known as van Ginneken's algorithm has been proposed in [56], which for a given routing tree and set of candidate buffer locations, will find the optimal buffer sizes to minimize the maximum delay over the tree. Extensions of the algorithm have included slew and power constraints while maintaining optimality [59]. These methods can only hope to maintain an acceptable skew value but offer no direct control. Works presented in [60, 61] attempt to perform buffer insertion in the presence of blockages, regions where buffers are not allowed.

As power consumption in a clock tree is the most important factor, a buffer sizing formulation that minimizes the power, subject to clock skew constraints is discussed. Accurate calculations of total power of a network is very complicated and requires specialized programs, such as SPICE [62]. However, it is known that buffer area is well correlated with the power consumption [57]. Hence, buffer sizing to minimize power while maintaining skew can be expressed mathematically as:

$$\begin{aligned} \min.\ \ & \text{Area}(\mathbf{x}) = \sum_{b^{\triangleright} \in B} x_{w_{b^{\triangleright}}} x_{l_{b^{\triangleright}}} \\ s.t.\ \ & \max\{d_i(\mathbf{x}) - d_j(\mathbf{x})\} \leq t_{\text{skew}},\ \forall i, j \in S,\ i \neq j \\ & x_{l_{b^{\triangleright}}} \geq l_{\min},\ b^{\triangleright} \in B \\ & x_{w_{b^{\triangleright}}} \geq w_{\min},\ b^{\triangleright} \in B \end{aligned}$$

where, $[x_{w_{b^{\triangleright}}}, x_{l_{b^{\triangleright}}}]$ are the width and length of buffer $b^{\triangleright}$ belonging to the set of all buffers, B, in the clock tree, $\mathbf{x}$ is the vector representing the buffer widths and lengths of all the buffers in the clock tree and Area($\mathbf{x}$) is the total area of the buffers. $d_i(\mathbf{x})$ is the delay from source to clock sink i in the set of all sinks S. Target clock skew is represented by t_{skew}, and $l_{\min}$ and $w_{\min}$ are the minimum length and width of a

buffer. This formulation is not convex because of the skew constraints because of the subtraction and the form of the delay terms. Hence, the quality of the obtained solution is very susceptible to the choice of the initial solution and the methodology used for solving the problem.

4.3.1 Variation-Aware Techniques

When circuits are fabricated, the manufacturing process introduces defects. Such defects are referred to as process variations. Process variations are unavoidable and undeniable at current technology nodes. Fabricating circuits considering only nominal values can lead to a large percentage of dies failing to meet timing specifications. A sequential linear programming technique for sizing buffers and wires considering skew and power supply variation using first order Taylor expansion is proposed in [63]. In [64], process variations are considered in the buffer and wire sizing problem which is formulated as a power-constrained skew minimization problem. Sequential programming is then employed to solve the robust optimization problem. However, the buffer and wire sizes are continuous variables and the results after discretization are not reported. In addition, only an elementary model is used for delay calculations and accurate SPICE simulation results are not available.

In [65], the effects of supply voltage variations are considered by treating the delay of each path as a random variable. The proposed technique achieves a fine trade-off between robustness and power efficiency where the capacitance of the resulting clock network increases by not more than 60%, and the clock tree is still twice as efficient in power compared to a clock mesh structure. In [66], a technique is proposed to produce robust clock trees by budgeting the power consumption between the top and bottom level trees. The empirical results show that this technique is more efficient in decreasing power consumption and runtime when compared to the existing cross link techniques.

However, this technique only considers the effects of variations in supply voltage and does not account for the variations in sizes of buffers and wires.

4.4 Parallel Computing

4.4.1 Introduction

The landscape in computing has moved from uniprocessor machines to workstations with two or more *cores*. This shift is a result of pressure on the computer hardware industry to sustain the exponential performance increases predicted by Moore's Law [3]. The traditional techniques of scaling feature sizes, clock frequencies, and supply voltages are no longer practical for generating the required performance gains [1]. This is because they introduce new problems that had negligible effects before but prove to be debilitating now. A sampling of the main problems being encountered are leakage and total power dissipation, diminishing supply-to-threshold voltage of transistors, and inadequate cooling [67].

The most common configuration of multi-core processors is the symmetric multiprocessor (SMP) system. SMP refers to an architecture where the operating system controls multiple identical processors and a shared main memory [68]. In order to harness the computational power of SMP systems, work must be performed in parallel. Often this involves rewriting existing code so that several tasks can be performed at the same time or can involve redesigning algorithms altogether when there is no clear way to parallelize what exists. The prospect of rewriting a large base of code creates inertia in adopting parallel computing practices. Programmers must shoulder the burden with the limited success of compilers capable of automatically parallelizing programs [69].

The success of a parallel algorithm depends upon being able to divide the work into pieces which can be performed simultaneously [68]. Some problems do not lend to parallel solutions but if a parallel solution is available there are several aspects which

affect the resulting performance. The next section covers main issues in parallelization that are relevant to the contributions made in this thesis.

4.4.2 Implementation Considerations

Allocating a unit of work to be assigned to a processor is referred to as *spawning a thread*. Having threads run in parallel creates new issues when compared to sequential, or single thread, programs. Key issues that arise are as follows [70]:

- **Race condition:** When it is possible for two threads to write or read-and-write from the same data location a *race condition exists*. Left unchecked, race conditions result in unpredictable program behaviour because results of computation are order-dependent.

- **Critical sections:** Lines of code which could potentially cause race conditions are called critical sections. Identifying them is key to preventing races.

- **Synchronization primitives:** Mechanisms for ensuring only a single thread enters a critical section at any given time are known as synchronization primitives. An example is a *lock* which enforces *mutual exclusion* by requiring a thread to obtain the lock's only key before entering and returning the key upon exiting the critical section. However, threads can be synchronized for reasons other than accessing data, for example if all threads must wait for the slowest one to complete. The primitive used to enforce this behaviour is referred to as a *barrier*. Synchronization primitives restrict the amount of work that threads can perform. They should be used only when necessary for that reason.

- **Load balance:** Another aspect that affects performance is the relative balance of the work performed by each thread. If one thread has much more work than

the others, the benefit of working in parallel is limited. Ideally each thread would be working throughout the entire process. How evenly the work is distributed is called the *load balance*.

- **Scheduling:** It is possible for more threads to exist than their are processors to compute with. In such situations, the order that threads are executed can affect performance. For example, if the work is not balanced, allowing the thread with the most work to execute last, is a poor work schedule. Furthermore the problem of scheduling work onto P processors is $\mathcal{NP}$-complete [71].

As these issues illustrate, care must be taken in designing and implementing parallel algorithms.

4.4.3 Analysis of Parallel Algorithms

Several metrics exist for measuring the performance of parallel algorithms. The one emphasized in this chapter is *speedup*. The speedup measures the ratio of a serial execution compared to a parallel one. In this work, the number of processors is considered fixed and between 2 and 16, which is common in workstations nowadays. Speedups which increase in direct proportion to the number of processors are said to be *linear*. The best possible speedup that can be achieved is P, the number of processors, ignoring the effects of memory hierarchies that may produce speedups in excess of P. Such speedup is said to be *ideal*. Linear speedup may be achieved but not ideal linear speedup. For example, if doubling the number of processors results in running time decreases less than a factor of two.

There are theoretical limits on speedups that can be achieved. Amdahl's Law [72] bounds the maximum speedup that can be achieved solving a problem of fixed size

with some number of processors. It can be stated as

$$\text{Speedup} = \frac{1}{(\text{Fraction of Serial Work}) + \frac{\text{Fraction of Parallel Work}}{P}}.$$

Roughly speaking, the speedup that can be achieved is limited by the amount of work that cannot be parallelized. In the modern era of CTS, problem sizes are very large and the serial work is small compared to the parallel work. For the modest amount of processors considered in this work, the maximum theoretical speedup is not a limitation. Another famous theoretical limit is Gustafson's Law [73]. Gustafson points out that if the fraction of serial work reduces as problem size increases, the speedup can increase. The law can be expressed as

$$\text{Speedup} = P - (\text{Fraction of Serial Work})(P - 1).$$

Another insight is that increasing the total amount of work may result in greater speedup so long as the serial part is reduced.

The traditional methods of analyzing the complexity of algorithms involve bounding the function for the work performed by an algorithm with a simpler function. The same technique can be used for analyzing parallel algorithms by introducing a variable for the number of processors. Although these bounds can be used in analyzing space complexity, they are used in analyzing time complexity in this work. The bound can be an upper bound, lower bound, or one which simultaneously bounds from above and below asymptotically. Three useful notations for describing complexity are [74]:

- Big O notation is used to express an upper bound on the complexity of an algorithm. Formally, a function $f(N)$ is Big O of $g(N)$, expressed $f(N) \in O(g(N))$, if

$$\exists c \in \mathbb{R}, N_0 \in \mathbb{Z} \text{ with } c > 0 \text{ such that } f(N) \leq cg(N), \forall N > N_0.$$

- Big Omega notation is used to express a lower bound. A function $f(N)$ is said to be Big Omega of $g(N)$, expressed $f(N) \in \Omega(g(N))$, if

$$\exists c \in \mathbb{R}, N_0 \in \mathbb{Z} \text{ with } c > 0 \text{ such that } f(N) \geq cg(N), \forall N > N_0.$$

- Big Theta notation expresses a bound from above and below. A function $f(N)$ is said to be Big Theta of $g(N)$, expressed $f(N) \in \Theta(g(N))$, if

$$\exists c_1, c_2, \in \mathbb{R}, N_0 \in \mathbb{Z} \text{ with } c_1, c_2 > 0 \text{ such that } c_1 g(N) \leq f(N) \leq c_2 g(N), \forall N > N_0.$$

Some examples are $2N\lg(N) + N \in O(N^2)$, $N\lg^2(N) \in \Omega(N\lg(N))$, and $2N^4 + 1000N^3 + 10 \in \Theta(N^4)$. As an example of the complexity of a parallel algorithm, consider an algorithm that is linear in problem size but has linear speedup in the number of processors, P. The complexity of such an algorithm can be expressed as $\Theta\left(\frac{N}{P}\right)$ and the speedup can be expressed as $\Theta(P)$.

In the context of CTS, many of the algorithms are recursive in nature. That is, an algorithm which depends on the solution of smaller versions of the same problem. When the problem size is small enough the solution can be obtained directly; these are the so-called *base cases*. For CTS algorithms, recursive algorithms are binary because the current solution depends upon the solution of two smaller problems. Conceptually, the work performed by a recursive algorithm can be drawn as a *recursion tree*. The root of the recursion tree is the initial call to the recursive function. Each level in the tree after level zero, shows the recursive calls from the level before. The leaves represent when the base cases are reached. Each node in the tree has a cost associated with it, which represents the work of the associated function call. An illustration of a recursion tree for an algorithm that performs N^2 work and sends problems of half the original size to two children is shown in Figure 4.2. The initial problem size is eight in the illustration.

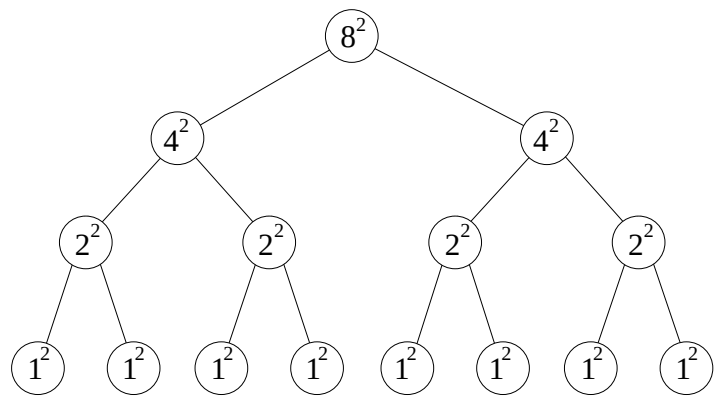


Figure 4.2: Illustration of a N^2 recursion tree generated by passing subproblems of half the original size to two children with initial problem size of eight.

Usually, it is easier to analyze the complexity of a single function call versus the overall complexity of the recursion tree. A useful theorem for obtaining the overall complexity of certain classes of recursive functions is called the *Master Theorem* [8]. The Master Theorem is applicable for solving recurrences of the form:

$$T(N) = aT\left(\frac{N}{b}\right) + w(N),$$

where $T(\cdot)$ is a recursive function defined on non-negative integers, $a \geq 1$, $b > 1$, and $w(\cdot)$ is expresses the work done at each node. As is conventionally done, the base cases are not shown in the recurrence and are assumed to be $O(1)$ complexity. The floor of $\frac{N}{b}$ is implicitly taken to ensure integer problem size. The Master Theorem has three cases:

1. If $w(N) \in O(N^{\log_b(a)-\epsilon})$ for some $\epsilon > 0$, then $T(N) \in \Theta\left(N^{\log_b(a)}\right)$,

2. If $w(N) \in \Theta(N^{\log_b(a)})$, then $T(N) \in \Theta\left(N^{\log_b(a)} \lg(N)\right)$,

3. If $w(N) \in \Omega(N^{\log_b(a)+\epsilon})$ for some $\epsilon > 0$ and if $a \times w\left(\frac{N}{b}\right) \leq k \times w(N)$ for some constant $k < 1$ and N sufficiently large, then $T(N) \in \Theta(w(N))$.

Even when the recurrence being analyzed satisfies the earlier mentioned restrictions,

the Master Theorem cannot always be applied. The additional restrictions in Cases 1 and 3 must be respected. For example, in Case 1, the condition on ϵ ensures that $w(N)$ is not just smaller than $N^{\log_b(a)}$ but polynomially smaller, i.e. smaller by a factor N^{ϵ}.

As an example of applying the Master Theorem, the work performed by the algorithm illustrated in Figure 4.2 satisfies the recurrence

$$T(N) = 2T\left(\frac{N}{2}\right) + N^2.$$

In this case $N^{\log_b(a)}$ is $N^{\log_2(2)} = N^1$ and $w(N) = N^2 \in \Omega(N)$. Since the restrictions in case 3 are met with $\epsilon < 1$, and $k > 0.5$, the recurrence $T(N) \in \Theta\left(N^2\right)$.

4.5 Optimization Techniques used for CTS

The contributions made in Chapter 6 utilize two concepts from mathematical optimization: Geometric programming and robust optimization. An overview of each is respectively presented in Sections 4.5.1 and 4.5.2.

4.5.1 Geometric Programming

Geometric programming (GP) involves optimizing a specific kind of optimization problems which are not initially convex but can be transformed into convex problems. A GP is an optimization problem in the general form:

$$\begin{aligned} \min. \quad & h_0(\mathbf{z}) \\ \text{s.t.} \quad & h_i(\mathbf{z}) \leq 1, \ i = 1, ..., p \\ & m_i(\mathbf{z}) = 1, \ i = 1, ..., g \end{aligned} \tag{4.1}$$

where $\mathbf{z} \in \mathbb{R}^q$ is the nominal value of the variables, $m_1(\mathbf{z}), \ldots, m_g(\mathbf{z})$ are monomials, and $h_0(\mathbf{z}), h_1(\mathbf{z}), \ldots, h_p(\mathbf{z})$ are posynomials. A monomial is a generic function $m(\cdot) : \mathbb{R}^q \rightarrow \mathbb{R}$ of the form of $m(\mathbf{z}) = \alpha z_1^{a_1} z_2^{a_2} \cdots z_q^{a_q}$ where, $\alpha > 0$ and the exponents, $a_1, \ldots, a_q \in \mathbb{R}$,

are real. A posynomial is a summation of monomials where all the coefficients of the summation are positive. An example of a GP problem is given below:

$$\begin{aligned} \min. \quad & \frac{z_1}{\sqrt{z_2}} + z_3^{1.5} \\ \text{s.t.} \quad & z_1 + 4.2 z_2^5 \leq z_3^{0.3} \\ & z_1 z_2 = z_3^4 \end{aligned}$$

Which in GP format is written as:

$$\begin{aligned} \min. \quad & z_1 z_2^{-0.5} + z_3^{1.5} \\ \text{s.t.} \quad & z_1 z_3^{-0.3} + 4.2 z_2^5 z_3^{-0.3} \leq 1 \\ & z_1 z_2 z_3^{-4} = 1 \end{aligned}$$

The problem formulated in (4.1) is not convex in general, but can be transformed into a convex problem. By defining a change of variables $y_i = log(z_i)$ and $\beta = \log \alpha$, the generic monomial is rewritten as: $m(\mathbf{y}) = e^{\beta} e^{a_1 y_1} \cdots e^{a_n y_n} = e^{\mathbf{a}^T \mathbf{y} + \beta}$. Similarly, for a posynomial $h_i(\cdot)$, the transformed function has the form $\sum_{j=1}^{J_i} e^{\mathbf{a}_{ij}^T \mathbf{y} + \beta_{ij}}$, where J_i is the number of monomials in posynomial $h_i(\cdot)$, and $\mathbf{a}_{ij} \in \mathbb{R}^q, i = 1, \ldots, m$ contain the posynomial exponents. The optimization problem associated with the transformation has the form:

$$\begin{aligned} \min. \quad & \textstyle\sum_{j=1}^{J_0} e^{\mathbf{a}_{0j}^T \mathbf{y} + \beta_{0j}} \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{J_i} e^{\mathbf{a}_{ij}^T \mathbf{y} + \beta_{ij}} \leq 1, \ i = 1, \ldots, p, \\ & e^{\mathbf{m}_i^T \mathbf{y} + \beta_i} = 1, \ i = 1, \ldots, g, \end{aligned}$$

where, $\mathbf{m_i} \in \mathbb{R}^q$ contain the equality constraint coefficients. This form of optimization can be transformed into a convex problem by taking the logarithm resulting in:

$$\begin{aligned} \min. \quad & \text{lse}(\mathbf{A}_0 \mathbf{y} + \boldsymbol{\beta}_0) \\ \text{s.t.} \quad & \text{lse}(\mathbf{A}_i \mathbf{y} + \boldsymbol{\beta}_i) \leq 0, \ i = 1, \ldots, p, \\ & \mathbf{G}\mathbf{y} + \boldsymbol{\beta} = 0 \end{aligned} \tag{4.2}$$

where, lse($\cdot$) is the log-sum-exp function: lse($\mathbf{y}$) $= log(e^{y_1} + ... + e^{y_J})$. GPs in convex form can be solved using non-linear convex optimization techniques [75].

It has been shown that the problem of minimizing logic gate area while meeting maximum delay with process variation affecting the width and length of gates can be formulated as a geometric program [57]. However, the same technique is not applicable to clock network design where clock skew is an additional concern.

4.5.2 Robust Optimization

In many stages of physical design, an optimization problem is solved, where a certain objective such as area, power, wire length or skew is minimized. The parameters and the variables of the optimization models are normally considered to be deterministic. However, there can be inherent uncertainties in the parameters or variables. For example, the length of a gate might be nominally assigned 45nm, but, during the fabrication, the actual length can measure 20% lower or higher (between 36 to 54nm). Major sources of uncertainty include process variations, measurement errors, uncertainties about future decisions, and partial access to information. Examples of typical uncertainties in physical design are supply voltage variations, manufacturing process variation and effects of thermal noise. In traditional optimization techniques, these uncertainties are ignored and only nominal solutions are obtained. These nominally optimal solutions can be very sensitive to variation in parameters and can even cause infeasibility in practice [76].

Robust optimization (RO) techniques [76, 77] try to obtain a balanced solution considering all uncertainties. A robust solution is an optimized solution when a set of uncertainties is considered. In the simplest form, a robust solution can be thought of as resistant to uncertainties. Hence, a robust solution can look inferior if directly compared to an individual nominally optimal solution obtained under the assumption

of perfect conditions. However, when simulations that include uncertainty such as Monte Carlo simulations or measurements of fabricated designs are performed, the robust solution will show its superiority.

The first step in producing a robust solution to a problem using robust optimization is to identify and model the uncertainties in the system. The source of uncertainty gives rise to different uncertainty sets. For example, if the uncertainties are due to variations in the supply voltage, they may be unknown but bounded. In other cases, such as uncertainty arising from manufacturing defects, a distribution may be known.

The second step is to integrate uncertainties into the optimization problem. An optimization problem that does not include uncertainty can be written as:

$$\begin{aligned} \min. \quad & f_0(\mathbf{z}) \\ \text{s.t.} \quad & f_i(\mathbf{z}) \leq 0, \ i = 1, ..., p \\ & g_i(\mathbf{z}) = 0, \ i = 1, ..., g \end{aligned}$$

where $\mathbf{z}$ represents the decision variables. A robust problem with uncertainties in the constraints can be written as:

$$\begin{aligned} \min. \quad & f_0(\mathbf{z}) \\ \text{s.t.} \quad & \tilde{f}_i(\mathbf{z}, \mathbf{u}) \leq 0, \ \mathbf{u} \in U, \ i = 1, ..., p \\ & g_i(\mathbf{z}) = 0, \ i = 1, ..., g \end{aligned}$$

where $\mathbf{u}$ is an uncertainty vector contained in the uncertainty set U and the inequality constraint functions $\tilde{f}_i$ include the uncertainty vector as input. Because the constraints must be satisfied for every u in the uncertainty set, the problem is optimized for the worst-case of uncertainties.

The formulation of certain objective and uncertain inequality constraints is most common in robust optimization [77]. However, the case of uncertainty in the objective

can be written as:

$$\begin{aligned} \min . \quad & \max_{\mathbf{u} \in U} \tilde{f}_0(\mathbf{z}, \mathbf{u}) \\ \text{s.t.} \quad & \tilde{f}_i(\mathbf{z}, \mathbf{u}) \leq 0, \ \mathbf{u} \in U, \ i = 1, ..., p \\ & g_i(\mathbf{z}) = 0, \ i = 1, ..., g \end{aligned}$$

The solution of this problem minimizes the maximum of the objective $\tilde{f}_0(\mathbf{z}, \mathbf{u})$ over all possible values of uncertainty $\mathbf{u}$.

The final step in robust optimization is finding a solution. Although the uncertainty set being considered may produce a problem that is not computationally tractable, there are many practical cases that produce optimization problems that can be efficiently solved. In these tractable cases, solvers implementing state-of-the-art algorithms are available to find the robust solution to a given problem [78]. In some other cases where the problem is not tractable, it may be possible to efficiently compute a robust feasible, but not necessarily optimal solution [77].

Stochastic optimization techniques have been used in applications where uncertainty exists. However, a requirement to use stochastic optimization is to confine the variations to a probabilistic model. In robust optimization, the uncertainty model is a set, and the solution is optimal considering all members of the set. Hence, robust optimization can be easier to solve and more effective if a design must perform well under all realizations of uncertainty. Robust optimization can also be used when the nature of uncertainty is not probabilistic, or a distribution is not available, not accurately known, or known but leading to intractable computational models.

Robust optimization has been used in the context of gate sizing in the physical design field. A method using robust optimization to perform gate sizing over multiple design corner cases is presented in [79]. For this problem, the uncertainty is modeled by a set of corners. Each corner assigns values to environmental parameters and process parameters, such as temperature and threshold voltage, respectively. Equations

representing the delays of the circuit are derived for each corner. By replicating the constraints for each corner in the optimization problem, the uncertainty is accounted for. The resulting gate sizing solution is the best that can be achieved for all corners.

Another gate sizing application of robust optimization is in [57]. In this work, the uncertainty is in the size of gates after manufacturing. The uncertainty is modeled using ellipsoidal uncertainty sets and can include intra-die correlations in manufacturing defects, which affect the resulting gate sizes.

A technique for yield optimization under process variation is discussed in [80]. In this work, a joint design-time and post-silicon minimization on parametric yield loss using adjustable robust optimization is discussed. Uncertainties in gate sizes and threshold voltages are dealt with while minimizing leakage power.

4.6 Summary

In this chapter, background material related to the proposed clock tree synthesis algorithms are discussed. An overview of the importance of the clock tree in a synchronous circuit and the main stages of CTS are discussed. A review of buffer sizing methods to improve the performance of clock trees is also given.

The use of parallel computing in solving large-scale problems is highly desirable. Several important aspects of parallel computing are presented.

Finally, geometric programming and robust optimization are explained as powerful mathematical tools in combating problems of scale.

Chapter 5

Parallel Clock Network Synthesis

5.1 Introduction

Circuit sizes have continued to grow at exponential rates over the past decades [3]. This growth is the cause of many of the challenges encountered during the process of laying out circuits today [81]. A less explored way of overcoming the challenges is to harness the proportionately growing computation power by working in parallel. Although not all problems involved in the physical design of circuits lend themselves to efficient parallel solutions, there are important problems that do. As will be shown in this chapter, several steps in CTS can be parallelized.[1]

CTS has been singled out as a step of utmost importance in recent years, as evidenced by the 2009 and 2010 industry-sponsored contests on their design during the ACM/IEEE International Symposium on Physical Design [82, 83]. As today's circuit sizes are exceeding millions of elements, working in parallel is becoming a necessity [46].

There have been a myriad of algorithms developed for CTS [46, 47]. However, the notion of parallel algorithms for CTS has been almost entirely overlooked. The work in [84] explores the problem of tree embedding in parallel, but does not consider the requirement of adding buffers to the network, which is essential in modern designs. It is also not a generic approach useful for different algorithms as the framework proposed in this chapter.

The main contributions of this chapter are as follows:

- Proposing a generic parallelization framework applicable to many CTS algo-

[1] With the assumption that the target machine has a modest number, P, of homogeneous processors and shared memory, as is common in workstations.

rithms,

- Proving asymptotic optimality of the framework,
- Developing parallel versions of common CTS algorithms,
- Discussing implementation details that improve speedup.

The remainder of this chapter is organized as follows: In Section 5.2, the generic framework that can be used to parallelize CTS algorithms is proposed. For each main step of CTS, commonly used algorithms are parallelized using the proposed framework and experimental validation are given in Sections 5.3.1-5.3.3. Finally, conclusions and future directions for the work are discussed in Section 5.4.

5.2 The Proposed Parallelization Framework

Many algorithms used for performing clock tree topology generation, embedding, and buffer insertion are recursive. Usually, the work performed at each step of the recursion involves only the attributes of a single node, and possibly its immediate children. Because of this, siblings in the recursion tree are independent. The independence of subtrees generated by the recursion is the key in the development of the proposed parallelization framework for CTS algorithms.

5.2.1 Overview of the Proposed Framework

To gain the intuition behind the parallelization framework, consider a recursion tree which is balanced, i.e. a binary tree with N nodes and height $\lceil \lg(N) \rceil$. Assuming that the work performed at each node in the tree involves only the node's attributes and its children's, any two nodes at the same depth in the tree can be computed in parallel. Furthermore, any subtrees rooted at nodes which are at the same depth in the tree can

be computed in parallel. The proposed framework consists of two phases:

1. Performing work serially until the target depth is reached,

2. Performing the work of subtrees rooted at nodes at and below the target depth in parallel.

The target depth is chosen as $\lfloor \lg(P) \rfloor$, where P is the number of processors available for computation. Setting the target depth at this level allows each processor to work on an independent subtree.

An illustration of the work performed in each phase is given in Figure 5.1. The

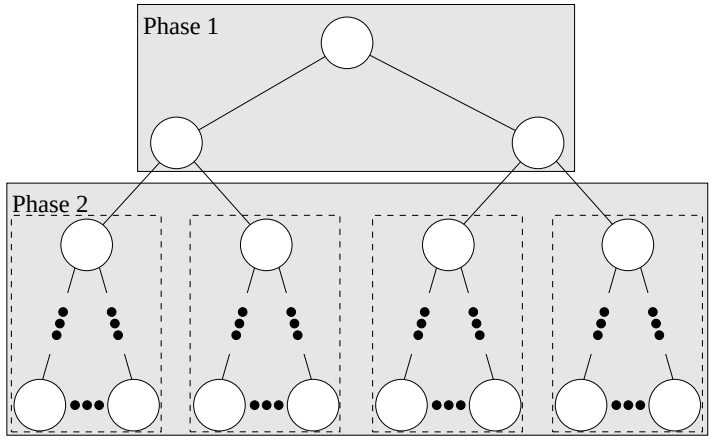


Figure 5.1: Illustration of the parallelism used in the proposed framework with a target depth of two.

target depth is two in the figure, allowing four processors to work in parallel. In Phase 1, the work at depth zero and one are dealt with serially. In Phase 2, the work at depth two and below is dealt with in parallel by different processors. The work of each processor is highlighted by a dashed box. Similarly, if the target depth is chosen to be one, Phase 1 work would be a single node and 2 processors could work in parallel on work of depth greater than zero.

```
Function(node,{arguments})
1. work before recursing
2. Function(node->lChild,{left child arguments})
3. Function(node->rChild,{right child arguments})
```

Figure 5.2: Generic top-down, binary recursive function pseudocode.

```
PFunction(node,{arguments},depth)
1. work before recursing
2. depth=depth+1
3. if depth==⌊lg (P)⌋ // Start Phase 2
4.   spawn Function(node->lChild,{left child arguments})
5.   spawn Function(node->rChild,{right child arguments})
6. else // Continue Phase 1
7.   PFunction(node->lChild,{left child arguments},depth)
8.   PFunction(node->rChild,{right child arguments},depth)
```

Figure 5.3: Pseudocode of the proposed CTS parallelization framework applied to the generic function in Figure 5.2.

The proposed framework is applied to recursive functions that perform work on the attributes of a node and possibly attributes of its immediate children. If only the node's attributes are required, a top-down algorithm is most appropriate. If the attributes of the children are required, a bottom-up algorithm is necessary. A generic top-down, binary recursive function is given in Figure 5.2 with base cases excluded. The `Function` takes a `node` and a set of `arguments` as inputs. Work done in Step 1 involves the `node` and the `arguments`. The last two steps recurse on the node's left and right children, `lChild` and `rChild`, and their associated arguments, which may come from the current `arguments` or have been computed in Step 1. In the algorithms considered in this thesis, the arguments are not shared and as such do not require the use of locks around critical sections in the code. Pseudocode for the proposed parallelization framework applied to the generic recursive function in Figure 5.2 is presented in Figure 5.3. The depth in the recursion tree is now passed as an argument. In Step 3, a test of

whether the target depth has been reached is performed. If the target depth has been reached, Phase 2 begins by spawning new threads with the serial `Function` accepting `node`'s children as arguments in Steps 4 and 5. Otherwise, the Phase 1 work continues with calls made in Steps 7 and 8. A thread barrier synchronization statement needs to be executed after all instances of `PFunction` finish to ensure all work is complete before the program proceeds. For bottom-up algorithms, the same synchronization statement needs to be executed and then the work in Phase 1 can be completed.

5.2.2 Complexity Analysis of the Proposed Framework

In this section it is shown that linear speedup can be achieved by applying the proposed parallelization framework to algorithms meeting certain requirements. The requirements to ensure asymptotic linear speedup are:

- The algorithm to which the proposed framework is applied is $\mathrm{O}\big(N \lg^2 N\big)$,
- The recursion tree is balanced.

The requirements are met by many algorithms in the CTS domain, so the applicability of the proposed framework is broad.

Due to the large problem sizes, N, encountered in modern CTS, common algorithms perform constant-time $\Theta(1)$ or linear $\Theta(N)$ work at each node. Algorithms that perform $\Theta\left(N \lg\left(N\right)\right)$ work and are also used occasionally. Assuming the recursion tree is balanced, the Master Theorem, explained in Section 4.4.3, can be applied with $a = 2, b = 2$ to determine the overall complexity of such algorithms by assigning $w(N)$ appropriately. For the case of $w(N) = \Theta(1)$ the overall complexity is $\Theta(N)$. For the case of $w(N) = \Theta(N)$ the overall complexity is $\Theta(N \lg(N))$. In fact, these results hold even when the tree is not balanced. The only requirement is that the larger of the two problem sizes reduces geometrically. This can be seen by applying

the Master Theorem with $1 < b < 2$. The Master Theorem cannot be applied in this case when $w(N) = \Theta(N \lg(N))$ because $N \lg(N)$ is not polynomially larger than $N^{\log_b(a)} = N^{\log_2(2)} = N$. However, the overall complexity is derived in Appendix A.1. The result of the derivation is that the overall complexity is $\Theta\left(N \lg^2(N)\right)$.

The following theorem proves that the proposed parallelization framework is asymptotically optimal in the speedup achieved for algorithms of complexity $\Theta(N \lg^2(N))$ or lower.

Theorem 5.2.1. *The asymptotic speedup achieved by using the proposed parallelization framework is optimal, i.e. linear in P, for balanced recursion trees when applied to $O\left(N \lg^2(N)\right)$ algorithms.*

Proof. Without loss of generality assume that N and P are powers of two. The assumption on P can be made because the proposed framework uses only powers of two processors for Phase 2 work. The speedup can be calculated as the ratio of the total work to the sum of the work performed during Phase 1 and the work of a single Phase 2 subtree. From the proof of Theorem A.1.1, the total amount of work without any parallel processing is bounded by $\frac{cN}{2} \lg(N)(\lg(N) + 1)$ for some $c \in \mathbb{R}$. The result from Theorem A.1.1 permits the assumption that N be a power of two as explained therein Appendix A.1. The amount of work in a Phase 2 subtree is $\frac{cN}{2P} \lg\left(\frac{N}{P}\right)\left(\lg\left(\frac{N}{P}\right) + 1\right)$, by the proof of Theorem A.1.1. The amount of work in Phase 1 is $cN \lg(P)\left(\lg(N) - \frac{\lg(P)-1}{2}\right)$ which is obtained by truncating the series in the proof

of Theorem A.1.1 at $i = \lg(P) - 1$. The speedup is then

$$
\begin{aligned}
&\frac{\frac{cN}{2}\lg(N)(\lg(N)+1)}{cN\lg(P)\left(\lg(N) - \frac{\lg(P)-1}{2}\right) + \frac{cN}{2P}\lg\left(\frac{N}{P}\right)\left(\lg\left(\frac{N}{P}\right)+1\right)} \\
&= \frac{P\lg(N)(\lg(N)+1)}{2P\lg(P)\left(\lg(N) - \frac{\lg(P)-1}{2}\right) + \lg\left(\frac{N}{P}\right)\left(\lg\left(\frac{N}{P}\right)+1\right)} \\
&= \frac{P\left(\lg^2(N)+\lg(N)\right)}{2P\lg(P)\lg(N) - P\lg^2(P) + P\lg(P) + (\lg(N)-\lg(P))(\lg(N)-\lg(P)+1)} \\
&= \frac{P\left(\lg^2(N)+\lg(N)\right)}{2(P-1)\lg(P)\lg(N) - (P-1)\lg^2(P) + (P-1)\lg(P) + \lg^2(N) + \lg(N)} \\
&= \frac{P\left(\lg^2(N)+\lg(N)\right)}{(P-1)\lg(P)(2\lg(N) - \lg(P) + 1) + \lg^2(N) + \lg(N)} \\
&= \frac{P\left(\lg^2(N)+\lg(N)\right)}{\lg^2(N) + \lg(N)(2(P-1)\lg(P)+1) - (P-1)\lg(P)(\lg(P)-1)} \\
&= O(P),
\end{aligned}
$$

where the last step follows because the $\lg^2(N)$ terms asymptotically dominate the numerator and denominator, keeping in mind that P is fixed. The constant hidden by the Big O notation can be chosen arbitrarily close to one implying the speedup approaches the ideal value of P. □

5.2.3 Practical Considerations

Highly Unbalanced Recursion Trees

The restriction that the recursion tree be balanced in Theorem 5.2.1 is a mathematical convenience. Although CTS topology generation algorithms tend to generate well-balanced trees, it is still possible to achieve optimal asymptotic speedup without a strictly balanced tree. In fact, the tree can be near degenerate. Consider applying the proposed framework to the recursion tree depicted in Figure 5.4 with target depth of one, i.e. $P = 2$. In this case, although the tree is extremely unbalanced because the depth of two leaves can be arbitrarily lower than the depth of the other leaves, the total work of each Phase 2 subtree is equal. As a result, the asymptotic speedup approaches

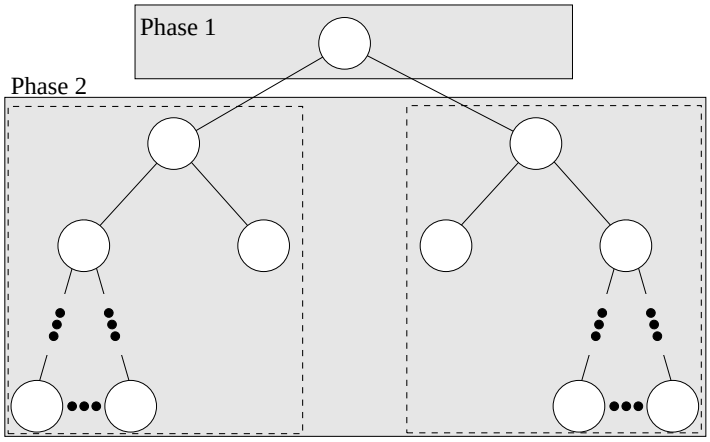


Figure 5.4: A degenerate recursion tree in which optimal asymptotic speedup is achieved by applying the the proposed framework with a target depth of one.

the ideal of P. It is with this looser interpretation of balance, i.e. balanced total work of Phase 2 subtrees, that asymptotically optimal speedup can be achieved.

The recursion tree in Figure 5.4 can also be used to illustrate a limitation of the proposed framework for trees that are highly unbalanced. Consider applying the proposed framework with target depth of two, i.e. four processors. In this case two subtrees have only a single node of work to perform. The speedup that could be achieved would approach two, rather than an ideal speedup of four. This behaviour of flat or decreasing speedup is unlikely to occur in the target application where the size of the trees is large and are designed with balance in mind. However, this phenomenon can explain inconsistent speedup scaling as processors are added when solving the same problem.

The asymptotic result of Theorem 5.2.1 does require an assumption on the value of P. However, for it to be relevant to problems of practical sizes it is useful to assume P is modest, between 2 and 16. The larger the problem, the closer the speedup will be to approaching the ideal value of P. Furthermore, the lower the complexity of the work performed at each node in the recursion tree, the sooner the speedup will achieve an ideal value. Roughly speaking, if the work at each node is $O(1)$ it will achieve

a speedup higher than if the work were $O(N)$. Lemmas A.3.1 and A.2.1 are special cases of Theorem 5.2.1 when the work performed at each node is $\Theta(N)$ and $\Theta(1)$, respectively. Their proofs include equations for speedup, as the proof of Theorem 5.2.1 does, which support both the ideas of approaching ideal speedup if problem sizes are increased and if the complexity of the work is decreased.

In Section 5.2.2 it is proven that the proposed framework is successful in achieving linear speedup in some cases. A natural question is when it is not successful in achieving linear speedup. Lemma A.4.1 proves that when the work at each node is $\Omega\left(N^2\right)$ the speedup achieved by the proposed framework is bounded above by a constant. By using the Master Theorem, it can be shown that recursive algorithms performing $\Omega\left(N^2\right)$ work at each node are $\Omega\left(N^2\right)$ complexity overall.

Parallelization of Phase 1

One extension of the proposed framework is to allow nodes in Phase 1 to be scheduled in parallel. This has no effect for $P < 4$. Asymptotically, ideal linear scaling in P has been proven for algorithms of complexity $O\left(N \lg^2(N)\right)$. However, when $P \geq 4$, there is potential for improving speedup of problems with sizes not large enough to achieve ideal scaling. The strategy of allowing parallel execution during Phase 1 introduces scheduling conflicts and extra overheads which can derail attempts at increasing speedup [85]. By not allowing parallel execution in Phase 1, the optimal work schedule of assigning each Phase 2 subtree to unique processors is obvious and processors do not need to perform context switches. In Section 5.3, the effectiveness of the proposed framework on CTS algorithms is shown.

It is also worth mentioning that allowing parallel processing in Phase 1 can not achieve linear speedup for $\Theta\left(N^2\right)$ algorithms. To improve the speedup for $\Omega\left(N^2\right)$ algorithms, a non-generic technique must be used, if such a technique is possible for a particular algorithm. However, $\Omega\left(N^2\right)$ algorithms are out of favor because their

```
PMMMTopo(set sinks, direction coord, int depth)
1. find median of sinks in coord
2. partition sinks into upperSinks and lowerSinks by the median
3. if depth==⌊lg(P)⌋
4.   spawn MMMTopo(lowerSinks,!coord)
5.   spawn MMMTopo(upperSinks,!coord)
6. else
7.   PMMMTopo(lowerSinks,!coord,depth+1)
8.   PMMMTopo(upperSinks,!coord,depth+1)
```

Figure 5.5: Pseudocode for performing parallel MMM topology generation.

running times become prohibitive for large problems. Great strengths of the proposed framework are that it is generic in application, it does not require modifying the work performed by the underlying algorithm, and it can achieve ideal speedup for recursive algorithms with $O(N \lg^2(N))$ complexity.

5.3 Validation of the Proposed Framework

5.3.1 Proposed Parallel Topology Generation

Consider MMM partitioning explained in Section 4.2.1 which, given N sinks, recursively selects the median of each partition in alternating coordinates. In a partitioning algorithm, parallelism is easy to come by as each generated partition is independent of the others. The proposed approach to capture the parallelism is to find the median using an optimal $\Theta(N)$, or linear-time, algorithm [86] and work in parallel once there is a partition for every processor.[2] This results in an algorithm which is $\Theta\left(\frac{N}{P}\log(\frac{N}{P})\right)$, compared to $\Theta(N\log(N))$ originally. Asymptotic linear speedup is guaranteed by Lemma A.3.1, which states that linear speedup is achieved for recursive $\Theta(N)$ work, because partitioning by the median ensures a balanced tree. A simplified pseudocode listing of the parallel algorithm ignoring base cases is provided in Figure 5.5. In the pseudocode,

[2] A linear-time median is provided in the C++ Standard Template Library.

a variable of type `direction` can represent two possible values, horizontal and vertical. The operation `!` on a `direction` alternates the direction. The median of the current set of `sinks` is found in Step `1` and the sinks are divided into two partitions based upon the median in Step `2`. Once the target depth of $\lfloor \lg(P) \rfloor$ has been reached in Step `3`, Phase 2 threads are spawned with a non-parallel version called `MMMTopo` to avoid the conditionals. Otherwise, the depth is incremented and Phase 1 work continues in Steps `7-8`. In practical implementations, the topology node needs to be instantiated and left and right children assigned. Furthermore, a synchronization command needs to be executed before performing further CTS steps as explained in Section 5.2. Applying the framework only affects execution time and does not change the output of the algorithm.

The parallelization framework can also be used to parallelize the BB algorithm explained in Section 4.2.1. The resulting time-complexity is $\Theta\left(\frac{N}{P}\lg^2\left(\frac{N}{P}\right)\right)$ compared to $\Theta(N\lg^2(N))$ in the non-parallel version.

In contrast to partitioning, commonly used clustering algorithms do not create independent subproblems. As a result the parallelism that can be exploited in clustering algorithms is at the data-level and not the task-level. Although they can be parallelized, the speedup from complexity analyses do not improve as much as in partitioning [84].

Experimental Validation

In this section the performance of an implementation of the proposed parallel algorithm presented in Figure 5.5 is evaluated. The algorithms proposed in this chapter are implemented in C/C++ using OpenMP tasks [87] and compiled using Intel compiler version 12.0.3. Implementations of the parallel partitioning algorithms only required adding a small number of additional lines of code to the serial versions using OpenMP tasks. In the proposed algorithm, no threads share work, nor do they get assigned work more than once. For OpenMP implementations in particular, using such a deterministic

work schedule avoids crippling overheads and poor task schedules [85].

Experiments are performed on a Linux server with Intel Xeon cores running at 2.67GHz and 74GB of RAM. Circuits used in the experiments are based on the circuits in the ISPD 2010 contest benchmarks [83] and range in size 1000 sinks (C1) to 150000 (C7).[3] The benchmarks have been modified to include more sinks distributed over the circuit area and varying sink capacitances to be more representative of problems seen in industry [46]. The benchmark specifications are given in Table 5.1.

Table 5.1: Specification of Benchmarks used in Validating Parallel CTS Algorithms.

Name	Height (mm)	Width (mm)	Number of sinks	Total sink capacitance (nF)	Varying sink capacitance
C1	8.0	8.0	1000	14	✓
C2	13.0	7.0	2000	27	✓
C3	3.1	0.5	4000	53	✓
C4	2.1	2.7	8000	105	✓
C5	2.3	2.5	15000	180	✓
C6	1.9	0.9	15000	180	✓
C7	2.5	1.4	150000	1624	✓

Parallel Method of Means and Medians The running times obtained by executing `PMMMTopo` using two, four, and eight processors and a single processor running a serial version ($P = 1$) are presented in Table 5.2. To better interpret the results, the speedups

Table 5.2: Running times with varying number of processors using `PMMMTopo` in Figure 5.5.

	Running Time (s×10^{-3})			
Circuit	$P = 1$	$P = 2$	$P = 4$	$P = 8$
C1	6.8	4.7	3.5	1.9
C2	11.4	8.4	5.8	3.6
C3	23.6	15.4	11.3	6.7
C4	41.7	31.2	22.1	9.8
C5	72.4	39.2	24.1	16.9
C6	71.6	51.0	31.3	19.3
C7	789.4	460.5	314.4	191.3

obtained by executing `PMMMTopo` using two, four, and eight processors compared to a

[3]Circuits available at `http://people.ucalgary.ca/~lmrakai/parallelBenchmarks.tgz`

single processor running a serial version are shown in Figure 5.6. Linear speedup can

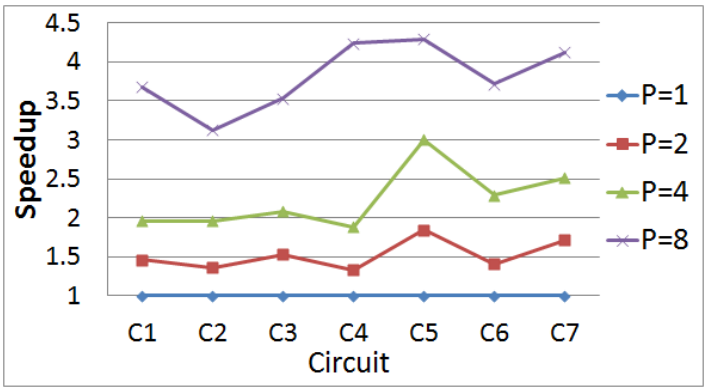


Figure 5.6: Speedups obtained by using `PMMMTopo` in Figure 5.5.

be observed as eight threads roughly doubles the speedup of four, and four roughly doubles the speedup of two threads. Ideal linear speedup, i.e. speedup equal to P, is not achieved in part because the Phase 1 work done before generating parallel tasks involves the largest partitions. As the problem sizes grow, the proportion of work performed in Phase 2 overtakes that of Phase 1 allowing asymptotic ideal speedup. Several other uncontrollable factors also restrict the potential speedup, such as the time taken to dispatch threads, processor communication delays, the CPU utilization of the machine during the experiment, and hierarchical memory effects, i.e. caching and paging.

Parallel Balanced Bipartition The same experiment as partitioning with parallel MMM is performed using a parallel version of BB. The running times are presented in Table 5.3 and associated speedups are illustrated in Figure 5.7. The complexity of the BB algorithm is on the cusp of what algorithms can achieve linear speedup, i.e. $\Theta(N \lg^2(N))$, and the speedups obtained are less than what is achieved the lower complexity MMM algorithm. Appreciable speedup is achieved with $P = 2$, but the benefit of using four or eight is less appreciable except for the largest test case, C7. Although there is no clear indication of linear speedup in this experiment, the upward

Table 5.3: Running times with varying number of processors using a parallel BB algorithm.

	Running Time (s×10^{-3})			
Circuit	$P = 1$	$P = 2$	$P = 4$	$P = 8$
C1	275.9	188.2	145.8	140.4
C2	833.9	556.0	499.4	466.1
C3	3204.6	2121.2	1910.1	1790.5
C4	11264.8	7393.8	6665.9	6227.6
C5	44309.1	29489.1	26117.8	23352.3
C6	44341.9	29548.0	26035.6	23122.2
C7	516908.3	290483.1	212415.4	157530.7

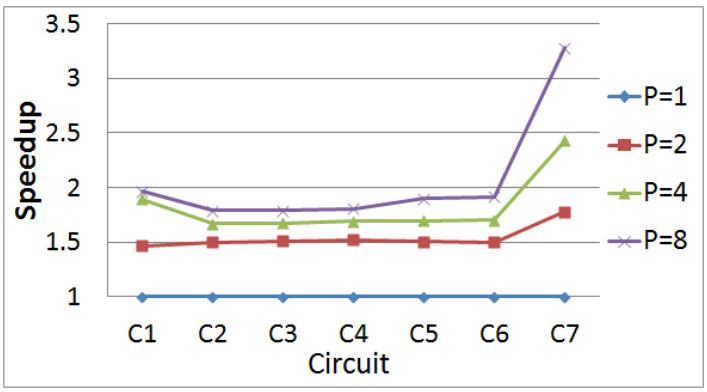


Figure 5.7: Speedups obtained by using parallel BB.

```
PMMMEmbed(topologyNode* node, int depth)
1. if depth==⌊lg(P)⌋
2.   spawn MMMEmbed(node->left)
3.   spawn MMMEmbed(node->right)
4.   synch
5. else
6.   PMMMEmbed(node->left, depth+1)
7.   PMMMEmbed(node->right, depth+1)
8. set node's location as mean of its children's
```

Figure 5.8: Simplified pseudocode listing for performing parallel MMM tree embedding.

trend in speedup suggests the following conclusion. As problem sizes continue to grow, so too will the benefit of applying the proposed framework to $\Theta(N \lg^2(N))$ algorithms. The trees generated by BB are also not as well-balanced as with MMM, but the problem size is the main limiting factor in this experiment.

5.3.2 Proposed Parallel Tree Embedding

With a given topology tree, the routing can be performed in parallel across independent subtrees in the topology tree. For MMM, the parallel version of the algorithm requires minimal modifications and results in linear speedup of the algorithm, i.e. $\Theta\left(\frac{N}{P}\right)$. A simplified listing of the pseudocode is given in Figure 5.8. The statements regarding practical implementations mentioned in Section 5.3.1 also hold for this algorithm. The embedding phase of MMM is bottom-up, so a `synch`ronization statement is required within the function because both subtrees must be completed before their parent can proceed. When more than two processors are available, all but two processors are idle at any time. This can be remedied by having one function to dispatch all the Phase 2 parallel jobs, lines `1-3` and `5-7`, a `synch` over all threads, and another function to work on the Phase 1 work.

Zero skew embedding algorithms can be parallelized to achieve linear speedup using

the same method by performing a zero skew merge instead of a mean computation in line 8.

The method of assigning subtrees to processors is most useful when the topology tree is balanced, which is guaranteed for MMM and pair-wise clustering algorithms. The BB algorithm is not ideally balanced but under realistic assumptions [51] is balanced enough for the linear speedup worst-case bound. For pathologic cases where the trees are of $\Theta(N)$ height, there is little parallelism that can be exploited.

Experimental Validation

Parallel Method of Means and Medians The running times of embedding using `PMMMEmbed` with input topology trees generated by MMM on the circuits presented in Table 5.1 are shown in Table 5.4. The associated speedups are shown in Figure

Table 5.4: Running times with varying number of processors using `PMMMEmbed` in Figure 5.8.

	Running Time (s$\times 10^{-3}$)			
Circuit	$P=1$	$P=2$	$P=4$	$P=8$
C1	0.8	0.5	0.3	0.1
C2	1.5	0.9	0.5	0.3
C3	3.0	1.6	1.0	0.5
C4	3.2	1.7	0.9	0.6
C5	6.3	3.4	2.0	1.1
C6	6.3	3.1	1.8	0.9
C7	104.8	46.7	27.8	15.9

5.9. Linear speedup can be observed. Since the overall complexity of MMM for tree embedding is $\Theta(N)$, near ideal speedups can be observed even for the smallest problems in the benchmark suite. Super linear speedup is achieved with $P = 2$ for benchmark C7. A speedup of greater than two can be attributed to memory hierarchy/caching effects. When more processors are utilized the total amount of cache memory available increases which allows for super linear speedups to be observed.

Parallel Deferred-Merge-Embed The same experiment is performed using a par-

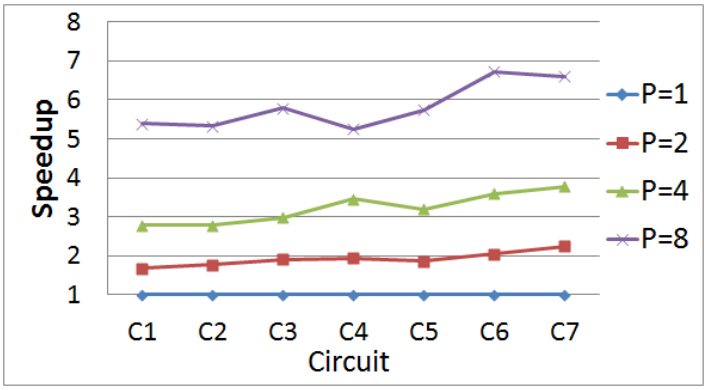


Figure 5.9: Speedups obtained by using `PMMMEmbed` in Figure 5.8.

allel version of a DME algorithm with input topology trees generated by the BB algorithm. The running times are presented in Table 5.5 and associated speedups are illustrated in Figure 5.10. In this experiment, the speedups achieved are closer to ideal

Table 5.5: Running times with varying number of processors using a parallel DME algorithm.

	Running Time (s$\times 10^{-3}$)			
Circuit	$P=1$	$P=2$	$P=4$	$P=8$
C1	0.4	0.2	0.1	0.1
C2	0.5	0.4	0.2	0.1
C3	1.3	0.7	0.5	0.3
C4	3.7	1.8	1.0	0.6
C5	5.3	3.6	1.5	0.9
C6	5.7	3.0	1.8	1.1
C7	119.5	65.5	32.7	17.5

than in Section 5.3.1 because work at each tree node is constant-time, $\Theta(1)$, compared to $\Theta(N)$. The speedup obtained is generally less than the speedup obtained with the parallel MMM embedding. This can be attributed to the fact that the topology trees generated by BB are not as well-balanced as those generated by MMM.

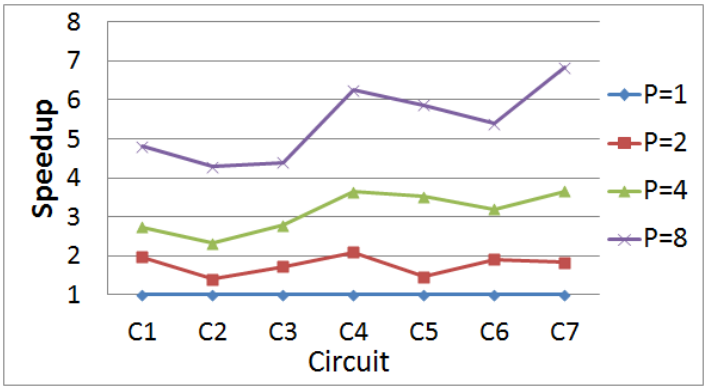


Figure 5.10: Speedups obtained by using parallel DME.

```
PGreedyBuff(embeddedNode* node, int depth)
1. if depth==⌊lg(P)⌋
2.   spawn GreedyBuff(node->left)
3.   spawn GreedyBuff(node->right)
4.   synch
4. else
5.   PGreedyBuff(node->left,depth+1)
6.   PGreedyBuff(node->right,depth+1)
7. node->load=node->left.load+node->right.load
8. buffer node's wire segment
9. node->load=remaining unbuffered capacitance
```

Figure 5.11: Simplified pseudocode listing for performing parallel greedy, bottom-up buffer insertion.

5.3.3 Proposed Parallel Buffer Insertion

For greedy bottom-up or top-down procedures for buffer insertion, since the tree is given at this step, both procedures can be parallelized in similar fashion. The difference is in whether recursive calls are made at the top or bottom of the function, respectively. A simplified listing of the pseudocode is given in Figure 5.11. Lines 7-8 insert a buffer whenever the capacitive limit is reached along a wire segment. The remaining wire capacitance that is unbuffered plus the last buffer's input capacitance is set as the node's remaining load in line 9. This algorithm leads to a linear speedup, i.e. $\Theta\left(\frac{N}{P}\right)$,

so long as the tree is not degenerate.

Dynamic programming approaches for buffer insertion can also be parallelized. The approaches consist of two phases: one bottom-up where candidate buffering solutions are propagated up the tree. The other is top-down where the best solution is known and the candidate that generated it is chosen at each node. Because of the independence of subtrees, assigning each subtree to a processor in each phase, it is possible to achieve the asymptotic linear speedup bound for non-degenerate trees.

Experimental Validation

Parallel Greedy Buffering The running times of performing buffer insertion with `PGreedyBuff` on a tree generated and embedded using MMM on the circuits presented in Table 5.1 are presented in Table 5.6. The associated speedups are presented in Figure 5.12. Speedups observed are linear but fall short of ideal values. This is because the amount of work done at each node in the tree, i.e. the number of buffers inserted, can vary. The number of buffers is a function of the length of wire, but is bounded above by a constant maintaining $\Theta(1)$ complexity. Nevertheless, the amount of work for each thread can be unbalanced.

Table 5.6: Running times with varying number of processors using `PGreedyBuff` in Figure 5.11.

	Running Time (s$\times 10^{-3}$)			
Circuit	$P = 1$	$P = 2$	$P = 4$	$P = 8$
C1	3.5	2.0	1.4	0.8
C2	6.1	3.4	2.0	1.0
C3	7.7	5.3	3.8	2.0
C4	12.5	6.9	3.9	2.4
C5	20.6	11.2	6.8	4.0
C6	20.6	11.4	8.1	4.7
C7	254.0	142.7	84.1	57.5

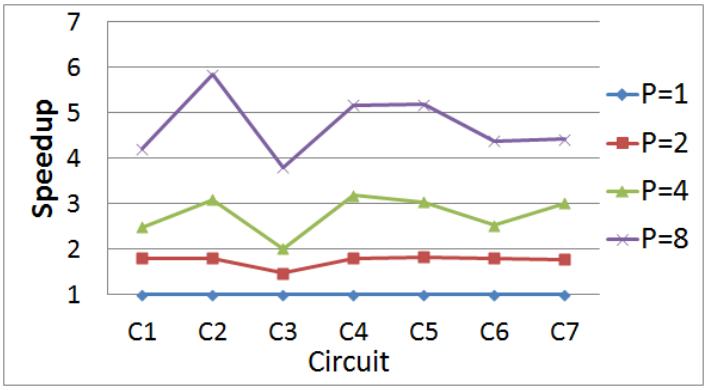


Figure 5.12: Speedups obtained by using `PGreedyBuff` in Figure 5.11.

5.4 Summary

In this chapter, a parallelization framework suitable for speeding up common CTS algorithms was proposed. Several algorithms used in CTS were parallelized and benchmarked. The experiments illustrate that linear speedup can be achieved in practice. Since the algorithms are used in state-of-the-art academic tools, the impact of the work is broadly applicable.

Chapter 6

Variation-Aware Clock Tree Buffer Sizing

6.1 Introduction

Buffers are an integral part of a clock network added to maintain the signal integrity. The goal of the buffer insertion step is to insert buffers to meet the slew constraints with an appropriate safety margin. In this step, it is desirable to keep the increase in power consumption of the network as small as possible, while maintaining low skew.

The buffers are sized to optimize the performance of the network, i.e. power consumption or skew. However, the current buffer sizing formulations result in optimization problems which are hard to solve. In addition, under process variation, the sizes of these buffers can vary greatly, affecting the predictability of the performance and lowering the yield of the network.

In this chapter, a GP formulation for buffer sizing is proposed. Using GP techniques [88], the non-convex and highly non-linear buffer sizing formulation is transformed into a series of convex optimization problems that can be solved quickly and efficiently. Then, robust optimization techniques are used to model variation in the problem formulation and produce the best solution under process variation. In the proposed algorithm, spatial correlation of process variations are considered instead of the worst-case variation scenarios, resulting in more area-efficient designs with minimal degradation in quality. In addition, the proposed framework is flexible and can be extended to handle variations in other parameters, such as the wire widths. The major contributions of this chapter are:

- Developing a novel formulation of the clock network buffer sizing problem that

converts a highly non-linear, non-convex problem into a series of convex problems that can be efficiently solved.

- Proposing to use robust optimization techniques to include process variations in the formulation.

- Applying a more accurate process variation model using spatial correlation instead of the worst-case variation scenario.

- Flexibility to extend the process variation model to other parameters such as wire widths.

- Offering designers the ability to optimally trade area for skew.

The rest of this chapter is organized as follows: In Section 6.2, the proposed GP model and the robust buffer sizing solution are discussed. The proposed model is examined and validated by several experiments in Section 6.3. A summary of the chapter is provided in Section 6.4.

6.2 Buffer Sizing Under Variation

In this section, a new formulation for buffer sizing of clock networks under variation using GP and robust optimization is presented. The advantage of the proposed GP formulation is that the problem can be solved using convex optimization techniques, where all local optima are global optima. Hence, finding a global optimum can be done more efficiently and in lower runtime. Once a nominal solution is obtained using the GP formulation, it is proposed to include process variations and formulate a new GP problem which includes uncertain parameters representing the variations in buffer

sizes. A problem with data uncertainty, does not have a unique optimum solution, but has a range of solutions that can happen based on where the actual parameters measure. An optimal solution to a problem with uncertainty is called a *robust solution* which in essence is a solution that compromises the set of problems created due to uncertainty. In this chapter, a robust formulation based on worst-case ellipsoidal robust optimization for considering variations is solved. The variations are considered to be spatially-correlated to agree with manufacturing data [57]. Other variations can also be modelled in the robust solution. The experimental results show that when the clock network power is calculated using SPICE, there is a dramatic decrease in the total power.

6.2.1 Algorithm Outline

The inputs to the proposed algorithm are t_{skew} and a clock network including buffers, sinks and net segments. The algorithm performs buffer sizing for this clock network with the objective of minimizing the buffer area, or in essence power, while meeting t_{skew} under process variation in lengths and widths of the buffers. The skew and power are calculated using SPICE during network validation. The output of the algorithm is a clock network sized under variation.

The proposed algorithm, as shown in Figure 6.1, includes two major phases. In the first phase, *Optimal Nominal Sizing*, the non-convex buffer sizing problem formulation presented in (6.1) is relaxed into a sequence of convex GP problems. Convex optimization techniques are used to solve large-scale problems in low runtime and obtain a global optimum. The sequence of relaxed GP problems are solved until t_{skew} is reached and nominal values of buffer sizes are obtained.

Once t_{skew} is reached, the covariance matrix, $\mathbf{\Sigma}$, is obtained and Algorithm Network Validation, Figure 6.2, is used to add spatially-correlated defects to buffer sizes and

```
Algorithm: Variation-Aware Buffer Sizing
Input: Clock network, t_skew
Output: Optimal buffer sizes under variation
Phase 1. Optimal Nominal Sizing
   1.a Perform GP relaxation
   1.b While t_skew not reached
          Solve GP
   1.c Calculate covariance matrix
   1.d Calculate average skew using
        Algorithm Network Validation
Phase 2. Variation-Aware Sizing
   2.a Formulate Robust GP (RGP)
   2.b While t_skew not reached
          Solve robust GP
   2.c Calculate average skew of the robust network
        using Algorithm Network Validation
```

Figure 6.1: Outline of the variation-aware buffer sizing algorithm.

recalculate skew and measure power using SPICE. The covariance matrix, $\mathbf{\Sigma}$, represents the intra-die correlation between buffer size variations, e.g. if two buffers are located close to each other, the variation in their sizes will be closer than the buffers that are located at a distance. $\mathbf{\Sigma}$ is calculated using a grid-based model similar to the one described in [89]. As there is significant uncertainty in the variation and defects introduced by the fabrication process, Monte Carlo simulation is used to introduce defects and extract timing information.

```
Algorithm: Network Validation
Input: Clock network, covariance matrix Σ
Output: Skew distribution
while # simulations < max simulations required
   Introduce spatially-correlated defects
   Extract timing information using SPICE
end
```

Figure 6.2: Network validation algorithm

In Phase 2 of the proposed algorithm, *Variation-Aware Sizing*, the nominally-sized network from Phase 1 is used as an initial solution. The covariance matrix is obtained and used in Algorithm Network Validation to represent uncertainty and a GP which considers spatially-correlated process variations is formulated. The resulting problem is called a Robust GP (RGP) formulation which is solved using RO techniques until t_{skew} is reached. Each one of the phases of the algorithm are described in detail in the following sections.

6.2.2 Phase 1: Optimal Nominal Sizing

The formulation in (6.1) is non-convex. In this section, techniques to transform it into a series of convex programming problems in GP format are proposed.

Proposed GP Relaxation

The buffer sizing formulation with the objective of minimizing total power/area under skew constraints given in Section 4.3 is repeated here for convenience:

$$\min.\ \text{Area}(\mathbf{x}) = \sum_{b^{\triangleright} \in B} x_{w_{b^{\triangleright}}} x_{l_{b^{\triangleright}}} \tag{6.1}$$

$$\text{s.t.}\ \max\{d_i(\mathbf{x}) - d_j(\mathbf{x})\} \leq t_{\text{skew}},\ \forall i, j \in S,\ i \neq j \tag{6.2}$$

$$x_{l_{b^{\triangleright}}} \geq l_{\min},\ b^{\triangleright} \in B \tag{6.3}$$

$$x_{w_{b^{\triangleright}}} \geq w_{\min},\ b^{\triangleright} \in B \tag{6.4}$$

where, $[x_{w_{b^{\triangleright}}}, x_{l_{b^{\triangleright}}}]$ are the width and length of buffer $b^{\triangleright}$ belonging to the set of all buffers, B, in the clock tree, $\mathbf{x}$ is the vector representing the buffer widths and lengths of all the buffers in the clock tree and Area($\mathbf{x}$) is the total area of the buffers. $d_i(\mathbf{x})$ is the delay from source to clock sink i in the set of all sinks S. Target clock skew is represented by t_{skew}, and $l_{\min}$ and $w_{\min}$ are the minimum length and width of a buffer. Each one of the objectives and constraint functions need to be verified as posynomials to be able to use GP. If a function is not posynomial, then proper transformations or

relaxations are performed to express it as a posynomial.

In this section, techniques to relax the buffer sizing problem (6.1) to GP format are proposed.

Objective function (6.1): The objective, $\sum_{b^{\triangleright} \in B} x_{w_{b^{\triangleright}}} x_{l_{b^{\triangleright}}}$, is the sum of buffer areas which is already a posynomial.

Minimum length (6.3) and minimum width (6.4): The minimum length, $l_{\min} \leq x_{l_{b^{\triangleright}}}$, and minimum width, $w_{\min} \leq x_{w_{b^{\triangleright}}}$, constraints can be easily transformed by rewriting them as ${l_{\min} x_{l_{b^{\triangleright}}}}^{-1} \leq 1,\ b^{\triangleright} \in B$ and ${w_{\min} x_{w_{b^{\triangleright}}}}^{-1} \leq 1,\ b^{\triangleright} \in B$.

Skew Constraints (6.2): Each skew constraint, $\max\{d_i(\mathbf{x}) - d_j(\mathbf{x})\} \leq t_{\text{skew}}\ \forall s_i \in S$, involves a delay calculation, maximum operator, and negative coefficients. Each one of the mentioned operations must be changed or relaxed to conform (6.2) to a posynomial format. The proposed techniques for changing these operations are explained in the following.

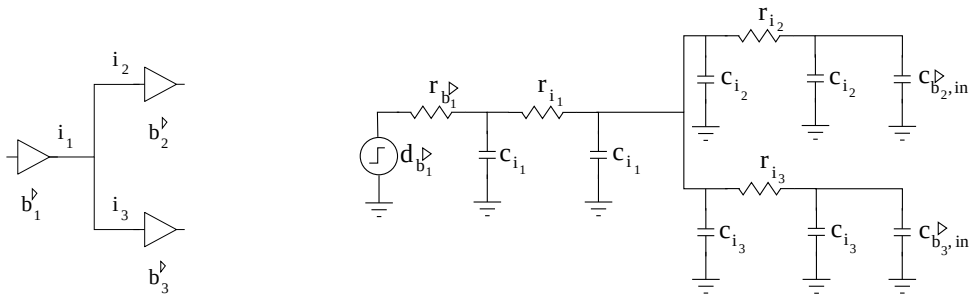
Delay Calculations: In order to accurately calculate delay, SPICE needs to be used. However, using SPICE is not practical when solving large-scale optimization problems because the runtime becomes prohibitive. The Elmore delay model used in [57] which gives a *reasonable approximation* for the delay and is a posynomial is used in this chapter. More accurate delay models can be used as long as they are posynomials, such as [90]. All delay calculations are finally verified using SPICE when experiments are performed.

For an RC tree, the Elmore delay is the summation of the resistance of each segment times the downstream capacitance. When a buffer is inserted in an RC tree, it isolates the upstream and downstream capacitances. A buffer $b_i^{\triangleright}$ is modeled as an input capacitance, $c_{b_i^{\triangleright},\text{in}}$, output resistance, $r_{b_i^{\triangleright}}$, and an intrinsic buffer delay, $d_{b_i^{\triangleright}}$, which is the product of buffer's output resistance and output capacitance, $c_{b_i^{\triangleright},\text{out}}$, i.e. $d_{b_i^{\triangleright}} = r_{b_i^{\triangleright}} c_{b_i^{\triangleright},\text{out}}$. In terms of the length and width of $b_i^{\triangleright}$, $x_{l_{b_i^{\triangleright}}}$ and $x_{w_{b_i^{\triangleright}}}$, the resistance

and capacitances of $b_i^{\triangleright}$ are:

$$r_{b_i^{\triangleright}} = \gamma_1 \frac{x_{l_{b_i^{\triangleright}}}}{x_{w_{b_i^{\triangleright}}}}, \; c_{b_i^{\triangleright}} = \gamma_2 x_{l_{b_i^{\triangleright}}} x_{w_{b_i^{\triangleright}}} + \gamma_3$$

where $\gamma_1, \gamma_2, \gamma_3$ are technology dependent parameters which are always positive. Hence, the expressions for resistance and capacitance are posynomials.



(a) Buffered tree segment (b) Equivalent RC network

Figure 6.3: An example of a buffered tree segment and its equivalent RC network.

In Figure 6.3, a buffered tree segment and its equivalent RC network are shown. As an example, the delay from the input of $b_1^{\triangleright}$ to the input of $b_2^{\triangleright}$ is given by:

$$d_{b_1^{\triangleright}} + r_{b_1^{\triangleright}} D_{b_1^{\triangleright}} + r_{i_1} D_{i_1} + r_{i_2} D_{i_2},$$

where D_i denotes the capacitance downstream of the resistance of element i. Since the resistance and capacitance of buffers and wires are posynomial functions of buffer lengths and widths and the expression for sink delay is formed by multiplication and addition, the Elmore delay is a posynomial. In the rest of this chapter, the Elmore delay approximation is shown with $d(\cdot)$, and refers to a posynomial. It should be noted that the wire width may be treated as a variable and the delay remains a posynomial in the length and width of buffers and width of wires. The addition of minimum wire width constraints and including wire area in the objective creates a formulation to simultaneously size buffers and wires. However, buffer sizing is the focus of this thesis.

Negative coefficients: To eliminate the negative coefficient of (6.2), the skew constraint is first relaxed by replacing $d_j(\mathbf{x})$ with the minimum sink delay, $d_{\min}$. As $d_i(\mathbf{x}) - d_{\min} \geq d_i(\mathbf{x}) - d_j(\mathbf{x})$, this relaxation does not affect feasibility. This relaxation may be large for some sinks. However, as will be discussed in Section 6.2.2, it is used in an iterative scheme and $d_{\min}$ is updated in each iteration. Therefore, the target skew is eventually reached. Both sides of the relaxed constraint are added by $d_{\min}$ that leads to $\max\{d_i(\mathbf{x})\} \leq t_{\text{skew}} + d_{\min}$.

Maximum operator: The maximum operator in the skew constraint is replaced with $\mu \leq t_{\text{skew}} + d_{\min}$, and a set of constraints $d_i(\mathbf{x}) \leq \mu,\ \forall s_i \in S$, where μ is a dummy variable.

All the new constraints are monomial constraints and together act as the maximum operator. The resulting problem is an optimization problem in the standard GP format:

$$
\begin{aligned}
(GP) \quad \min. \quad & \text{Area}(\mathbf{x}) = \textstyle\sum_{b^{\triangleright} \in B} x_{w_{b^{\triangleright}}} x_{l_{b^{\triangleright}}} \\
\text{s.t.} \quad & (t_{\text{skew}} + d_{\min})^{-1} \mu \leq 1 \\
& d_i(\mathbf{x}) \mu^{-1} \leq 1,\ \forall i \in S \\
& l_{\min} x_{l_{b^{\triangleright}}}^{-1} \leq 1,\ b^{\triangleright} \in B \\
& w_{\min} x_{w_{b^{\triangleright}}}^{-1} \leq 1,\ b^{\triangleright} \in B
\end{aligned}
\tag{6.5}
$$

Iterative GP Solution

Since the skew constraints $d_i(\mathbf{x}) - d_j(\mathbf{x}) \leq t_{\text{skew}}$ are relaxed to $d_i(\mathbf{x}) - d_{\min} \leq t_{\text{skew}}$, the calculated minimum delay may not be the optimal solution. To solve this problem, it is proposed to solve the GP problem iteratively by calculating $d_{\min}$ at each iteration and replacing with the new solution if the current skew is more than the target skew.

Algorithm Network Validation is used to validate the solution, once a nominal value has been obtained using the proposed GP model. The delay including process variations of up to ±20% of the nominal values for lengths and ±25% for widths, is obtained using Monte Carlo simulations and the skew is calculated using delays from

SPICE.

If the GP becomes infeasible, zero skew algorithms, such as [51], can be used to initialize the network and any t_{skew} of interest can be achieved. If a zero skew network is made up of only minimum size buffers, GP will not modify the network and Phase 1 can be skipped. The nominally-sized networks are observed to meet the slew constraints of 100ps set out in the ISPD contests [82, 83]. Hence, slew constraints were excluded from the GP formulation in this chapter. Posynomial models for slew constraints, for example [91], can be included in the GP.

6.2.3 Phase 2: Variation-Aware Sizing

The buffer sizing problem in (6.1) or in GP format of (6.5) is a deterministic optimization problem. However, once process variations are included in the formulation, variables and parameters of the problem can become non-deterministic. This means the optimal solution of (6.5) can become suboptimal. RO techniques have been successfully applied to problems where variability in the values of parameters exist [77]. The robust version of (6.5) for the buffer sizing problem is formulated in convex form as:

$$\begin{aligned} \min. \quad & \text{lse}(\mathbf{A}_0\mathbf{x} + \mathbf{b}_0) \\ \text{s.t.} \quad & \text{lse}(\mathbf{A}_i\tilde{\mathbf{x}} + \mathbf{b}_i) \leq 0, \; i = 1, \ldots, m \end{aligned} \tag{6.6}$$

where, $\tilde{\mathbf{x}} \in U$ is the vector of uncertain variables that belongs to the uncertainty set U, representing the uncertainty in the buffer sizes. As robust optimization typically is performed to ensure feasibility under uncertainty [77], the objective function is not modified to consider variations. In this problem in particular, meeting the skew constraints is the most important part of the problem, so the variations are not considered in the area objective. To be able to solve the uncertain problem, a worst-case robust

GP in the following form is solved by using the supremum function sup:

$$\begin{aligned} \min . \quad & \operatorname{lse}(\mathbf{A}_0\mathbf{x} + \mathbf{b}_0) \\ \text{s.t.} \quad & \sup_{\tilde{\mathbf{x}} \in U} \operatorname{lse}(\mathbf{A}_i\tilde{\mathbf{x}} + \mathbf{b}_i) \leq 0, \ i = 1, \ldots, m \end{aligned} \tag{6.7}$$

where, the supremum value of the constraints for all uncertain values is bounded by 0. To solve this problem using existing optimization techniques, an uncertainty model needs to be developed so that the uncertain variables can be replaced in the formulation. The proposed robust GP problem is formulated and solved using the model in (6.7) and the ellipsoid uncertainty model: $U = \{\mathbf{u} + \mathbf{\Sigma}^{1/2}\mathbf{d} \mid ||\mathbf{d}||_2 \leq 1, \quad \mathbf{d} \in \mathbb{R}^{2|B|}\}$, where, $\mathbf{u}$ is the nominal value of the variables, $\mathbf{\Sigma}$ is the covariance matrix of variations in length and width, and $\mathbf{d}$ is constrained to have norm less than one. By capturing the spatially-correlated variations in lengths and widths in the covariance matrix, area can be saved. This is because buffers are not sized according to an unrealistic worst-case scenario where all individual worst variations occur at the same time.

The variation in the lengths and widths of the buffers in the boundary constraints, $l_{\min} x_{l_{b^\triangleright}}^{-1} \leq 1, \ b^\triangleright \in B$ and $w_{\min} x_{w_{b^\triangleright}}^{-1} \leq 1, \ b^\triangleright \in B$, are normally introduced as part of the manufacturing process, so variation does not need to be considered for these constraints. The skew constraints need to be modified to include variations. First, the skew constraints are combined to eliminate the dummy variable μ, then the variable $\tilde{\mathbf{x}}$ is replaced by: $\tilde{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$. The new skew constraints are: $d_i(\mathbf{x} + \delta\mathbf{x}) \leq (t_{\text{skew}} + d_{\min}), \ \forall i \in S$ where, $d_{\min}$ is the infimum of the random variable related to the minimum delay. The delay function $d_i(\mathbf{x}+\delta\mathbf{x})$ is approximated using first order Taylor series: $d_i(\mathbf{x} + \delta\mathbf{x}) = d_i(\mathbf{x}) + \nabla_x^T d_i(\mathbf{x})\delta\mathbf{x}$. This approximation is an affine model with $d_i(\mathbf{x})$ representing the nominal value and $\nabla_x^T d_i(\mathbf{x})\delta\mathbf{x}$ representing the variation. The variational term, $\nabla_x^T d_i(\mathbf{x})\delta\mathbf{x}$, describes how small changes in the lengths and width will affect the delay. Substituting the Taylor series approximation, the constraint becomes:

$$d_i(\mathbf{x}) + \nabla_x^T d_i(\mathbf{x})\delta\mathbf{x} \leq (t_{\text{skew}} + d_{\min}), \ \forall i \in S \tag{6.8}$$

In order to solve the robust GP including the new skew constraints (6.8), the worst-case, i.e. sup, of the constraint should be considered.

The uncertainty vector, $\delta\mathbf{x}$ is bounded within the ellipsoid defined by the covariance matrix: $\delta\mathbf{x} = \mathbf{\Sigma}^{1/2}\mathbf{z}$, $||\mathbf{z}|| \leq 1$. The covariance matrix, $\mathbf{\Sigma}$, represents the intra-die correlation between buffer size variations, e.g. if two buffers are located close to each other, the variation in their sizes will be closer than the buffers that are located at a distance. $\mathbf{\Sigma}$ is calculated using a grid-based model similar to the one described in [89]. The term $\nabla_x^T d_i(\mathbf{x})$ is split into positive terms, $\boldsymbol{\phi}_1$, and negative terms, $\boldsymbol{\phi}_2$. Hence, the variational term, $\nabla_x^T d_i(\mathbf{x})\delta\mathbf{x}$, is equal to:

$$\boldsymbol{\phi}_1^T\mathbf{\Sigma}^{1/2}\mathbf{z} + \boldsymbol{\phi}_2^T\mathbf{\Sigma}^{1/2}\mathbf{z}, ||\mathbf{z}|| \leq 1.$$

Only the maximum variational term is of importance, and $\boldsymbol{\phi}_1^T\mathbf{\Sigma}^{1/2}\mathbf{z} + \boldsymbol{\phi}_2^T\mathbf{\Sigma}^{1/2}\mathbf{z}$, $||\mathbf{z}|| \leq 1$ is replaced by

$$\max_{||\mathbf{z}||\leq 1}(< \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1, \mathbf{z} > + < \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2, \mathbf{z} >), \tag{6.9}$$

where $< a, b >$ is the inner product of a and b. The inequality constraint becomes:

$$d_i(\mathbf{x}) + \max_{||\mathbf{z}||\leq 1}(< \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1, \mathbf{z} > + < \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2, \mathbf{z} >) \leq t_{\text{skew}} + d_{\min}$$

To remove the max operator, robust variables ρ_1 and ρ_2 are defined as:

$$\rho_1 = ||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1||, \quad \text{and} \quad \rho_2 = ||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2||,$$

hence

$$\boldsymbol{\phi}_1^T\mathbf{\Sigma}\boldsymbol{\phi}_1\rho_1^{-2} \leq 1, \quad \text{and} \quad \boldsymbol{\phi}_2^T\mathbf{\Sigma}\boldsymbol{\phi}_2\rho_2^{-2} \leq 1.$$

Using $||\mathbf{z}|| \leq 1$, the constraint will become

$$\begin{aligned}
\max_{||\mathbf{z}||\leq 1}\Big(< \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1, \mathbf{z} > + < \mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2, \mathbf{z} >\Big) &\leq \max_{||\mathbf{z}||\leq 1}\Big(||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1||\ ||\mathbf{z}|| + ||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2||\ ||\mathbf{z}||\Big) \\
&\leq \Big(||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_1|| + ||\mathbf{\Sigma}^{1/2}\boldsymbol{\phi}_2||\Big) \\
&= \rho_1 + \rho_2,
\end{aligned}$$

where the first inequality uses the Cauchy-Schwarz inequality $< \mathbf{c}, \mathbf{d} > \leq ||\mathbf{c}||\ ||\mathbf{d}||$. Using the above equations, the constraints of (6.5) considering the process variations in the buffer sizes can be reformulated using the following three constraints (6.11) to (6.13):

$$(RGP) \quad \min. \quad \text{Area}(\mathbf{x}) = \sum_{b^{\triangleright} \in B} x_{w_{b^{\triangleright}}} x_{l_{b^{\triangleright}}} \tag{6.10}$$

$$\text{s.t.} \quad d_i(\mathbf{x}) - d_{\min} + \rho_{1_i} + \rho_{2_i} \leq t_{\text{skew}}, \forall i \in S \tag{6.11}$$

$$\boldsymbol{\phi}_{1_i}^T \boldsymbol{\Sigma} \boldsymbol{\phi}_{1_i} \rho_{1_i}^{-2} \leq 1, \forall i \in S \tag{6.12}$$

$$\boldsymbol{\phi}_{2_i}^T \boldsymbol{\Sigma} \boldsymbol{\phi}_{2_i} \rho_{2_i}^{-2} \leq 1, \forall i \in S \tag{6.13}$$

$$l_{\min} x_{l_{b^{\triangleright}}}^{-1} \leq 1,\ b^{\triangleright} \in B \tag{6.14}$$

$$w_{\min} x_{w_{b^{\triangleright}}}^{-1} \leq 1,\ b^{\triangleright} \in B \tag{6.15}$$

This problem is still exclusively made of posynomials and monomials and is a standard form GP.

The solutions obtained by RGP and GP need to be discretized in order to be manufactured. This is because the available buffers generally come from a standard cell library. Standard cells cannot be scaled arbitrarily; instead only integer multiples of the smallest buffer are allowed. The discretization technique used in this work is to round the continuous values for buffer sizes down to the nearest integer. This strategy aligns with the objective of minimizing area but can negatively impact skew. The impact will be observed in the results of experiments presented in Section 6.3.

6.3 Experimental Results

To show the validity of the proposed algorithms, networks from the 2009 ISPD clock network synthesis (CNS) contest [82] are used because the networks are publicly available, in contrast to the 2010 ISPD contest. Details of the benchmarks used in the experiments are given in Table 6.1. From left to right, the columns tabulate the circuit

Circuit	Height (mm)	Width (mm)	Sink cap. (pF)	$\|S\|$	$\|B\|$
11	11	11	4.2	121	3536
12	8	13	4.1	117	3472
21	13	12	3.6	117	3568
22	12	5	3.4	91	2112
nb1	3	2	5.9	330	1264
31	17	17	9.6	273	7760
32	17	17	6.7	190	5904

Table 6.1: Details of the benchmarks used for experimentation.

names, height and width of the chip, total clock sink capacitance, number of sinks, and number of buffers in the input network. The number of buffers are calculated as the equivalent number of minimum size buffers. For example, if a buffer is four times wider than a minimum buffer of the corresponding type, it counts as four buffers. The process used is 45nm and the technology files are the same as used in the ISPD CNS contest. All experiments are performed on 2.8GHz Intel Pentium 4 processor with 1GB of memory. The algorithms are implemented in C++ and GPs are solved using Mosek 6.0 [78]. t_{skew} of all the optimization problems is set to be 1ps. All power and skew calculations are measured directly from Ngspice 24 [62]. Ngspice power measurements are SPICE accurate, i.e. include switching, short circuit, and leakage power. Since short circuit power is included, slew also impacts the power. It should be mentioned that in the experiments, the skew constraints in (6.5) are met after solving a single GP.

6.3.1 Performance of GP and RGP

In the first set of experiments, results tabulated in Table 6.2, the area obtained from solving (6.5) and (6.10), average skew of the Monte Carlo simulations, and the power calculations from Ngspice are compared to the results of the input network. The delay is calculated under process variation using a Monte Carlo simulations. The variations in buffer width and length are set to 25% and 20% of the nominal values, respectively. When performing Monte Carlo simulations, the number of simulations is 20 times

the number of buffers in the network. In Table 6.2(a), the buffer area for the input, nominally-sized, GP, in continuous, Cont., and discrete, Disc., buffer sizes and the robust solution, RGP, in continuous, Cont., and discrete, Disc., buffer sizes are given, respectively. In Table 6.2(b), the average skew results are given in the same order as columns of Table 6.2(a). To obtain the discrete results, the values obtained from solving GP and RGP are rounded down to the closest available discrete buffer size, and Ngspice is used to recalculate the skew to ensure feasibility. More sophisticated schemes where additional bounds are placed on buffer sizes in GP at each iteration to obtain final integer values can be developed as part of the future work. The power calculated using Ngspice for the input network and the improvement in the power after using GP and RGP are shown in Table 6.2(c). In Table 6.2(d), the runtime for the GP and RGP are given.

The objective of the buffer sizing, area and therefore power, are substantially reduced for all the test cases and in both continuous and discrete solutions. On average, the area is reduced by 75% for GP and 51% for RGP while the power is reduced by 61% for GP and 37% for RGP. In comparison with the nominally-sized GP network, the robust network has more area and power but lower skew. This means that the nominal GP solution is over-optimizing the area and power because it does not consider the variations.

On average, obtaining a robust network requires nine times more running time compared to the nominal network alone.

The average skew calculated using Ngspice for continuous GP increases slightly compared to the input while continuous RGP beats the input network by 14ps, on average. However, with the discrete sizes, both GP and RGP have higher average skews than the input network, on average. This is because the proposed GP and RGP are able to achieve the target skew using the approximate delay model with much smaller

buffers. This results in reduced area and power for the networks. These smaller buffers are more susceptible to the variations and the skew increases. It can be seen from the results that the circuits with the largest area decrease have the largest skew increase. For example, the most noticeable improvement in area and power happens for circuit nb1 with a decrease of 90% in area and 83% power reduction in GP solution and 75% area and 70% power reduction in RGP. The same happens for the other two circuits with relatively low skew, i.e. 21 and 22. The proposed model has the flexibility to deal with the problem of increase in the average skew by employing better delay models and increasing robustness.

The power of RGP and considering variation in the formulation can also be demonstrated by circuit nb1. When RGP is solved, the skew is reduced from 115 to 45ps. Other interesting results happen for circuits 11, 12, 31 and 32 in the continuous-sized and circuit 31 for the discrete-sized solutions, where using the GP/RGP formulation results in reduction of **all categories:** area, average skew and power. Special attention should be given to circuit 31 which is the hardest circuit to design: Using our proposed RGP has resulted in significant power reduction while there is no change in skew. The results in Table 6.2 suggest a trade-off between area/power and the skew, where reduction in area/power results in higher skew. These results also suggest that considering variations and using the proposed RGP model can result in optimal solutions where area, power and skew are all reduced. This is due to considering variations during the optimization process, as the optimized solution is obtained considering that the gate sizes can in reality be smaller than nominal values, i.e. higher skew. Finally, it should be mentioned that the average skew for RGP in continuous sized circuits is improved over the input circuits, on average. Therefore, better discretization methods will be developed so that this reduction in skew will become sustainable.

6.3.2 Trade-Off Analysis

The curve shown in Figure 6.4 is the Pareto optimal trade-off curve between area and t_{skew} for a small network with two buffers and two sinks, but the ideas apply to the larger circuits. For this circuit, a skew of about 6.15ps is not achievable as the curve

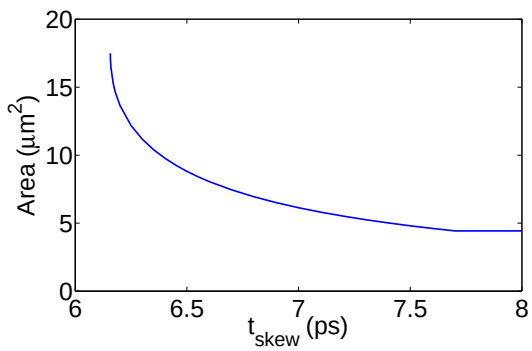


Figure 6.4: An illustration of the optimal area-t_{skew} trade-off for a small network.

grows asymptotically on the left side. Also, setting t_{skew} greater than 7.7ps makes the skew constraint inactive since 7.7ps can be achieved with less area than it takes to achieve a higher skew. A clock network designer can trace out such an optimal trade-off curve and decide upon the most appropriate design.

6.3.3 Robustness of RGP

Finally, to show the power of the robust solution, the distribution of the skew for each Monte Carlo run for the nominal and robust network considering spatial correlations are presented in Figure 6.5. For this comparison, the network for ispd09f11 is sized nominally (black) and sized considering variations (grey). When the nominal network is used, the skew is distributed between 39ps to 56ps, which means that the solution can be infeasible. However, for the robust network, the skew ranges around 24ps to 35ps, which is a much more robust circuit.

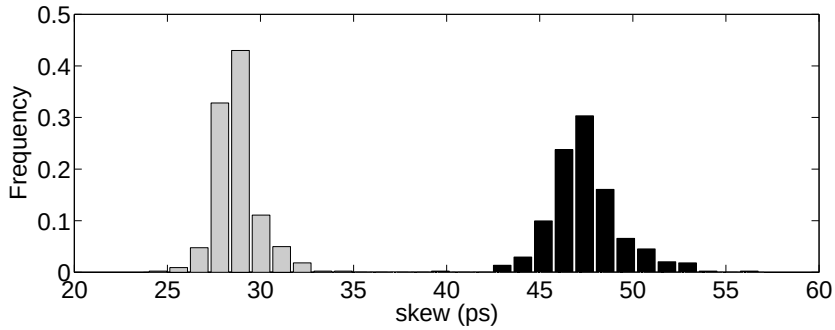


Figure 6.5: The Monte Carlo skew distributions for the nominally-sized network (black) and the robust network (grey).

6.4 Summary

In this chapter, it is shown that by using geometric programming, a globally optimal solution for buffer sizing problem can be obtained in an acceptable amount of runtime. In addition, a robust model is proposed that can consider the variation in the sizes of the buffers. The proposed techniques are based on state-of-the-art optimization techniques and the experimental results obtained from SPICE show improvement. By using the proposed GP formulations, designers can trace out optimal trade-off curves to decide on the most appropriate design.

For future work, a technique will be developed to discretize the buffer sizes with minimal disruption to skew. Also, the robust model can be developed as a multi-objective optimization problem with the objectives of minimizing the skew and the area simultaneously. Finally, the model can be extended to include other variations, such as wire width variations, to allow for even more robust designs.

Table 6.2: Comparison of input networks and the resulting nominally-sized networks and robust networks.

(a) Area

Circuit	Area (μm^2)				
	Input	GP		RGP	
		Cont.	Disc.	Cont.	Disc.
11	536	162	146	341	327
12	527	96	82	174	159
21	541	180	164	288	262
22	320	101	92	191	180
nb1	192	19	17	50	48
31	1177	387	350	705	669
32	895	204	177	421	395
Ave.	598	164	147	310	291

(b) Average Skew

Circuit	Average Skew (ps)				
	Input	GP		RGP	
		Cont.	Disc.	Cont.	Disc.
11	48	47	75	29	65
12	91	83	162	72	121
21	51	60	77	54	86
22	27	46	66	38	60
nb1	25	115	129	45	47
31	228	195	176	179	230
32	143	109	200	101	132
Ave.	88	94	126	74	106

(c) Power

Circuit	Power (μW)	Power **Improvement**			
		GP		RGP	
	Input	Cont.	Disc.	Cont.	Disc.
11	19668	48%	50%	18%	19%
12	19685	70%	71%	42%	43%
21	20631	44%	46%	23%	24%
22	12125	51%	52%	24%	25%
nb1	5234	82%	83%	70%	70%
31	42976	58%	60%	38%	39%
32	33392	64%	65%	37%	38%
Ave.	21959	60%	61%	36%	37%

(d) Runtime

Circuit	Runtime (s)	
	GP	RGP
11	20	167
12	53	372
21	41	289
22	25	116
nb1	33	304
31	204	1924
32	229	2270
Ave.	86	777

Part IV

End Matter

Chapter 7

Conclusions

The importance of solving scale-borne problems in physical design is paramount. This thesis investigates two important problems in the area: clustering for circuit placement and clock network synthesis.

The scale-borne problem in circuit placement is the ever-increasing number of instances which make exploration of vast solution spaces challenging. Clustering is an effective way to deal with this problem. By clustering a netlist, the circuit size is effectively reduced but this alone will not improve the solution quality of placement. The cells in a cluster must be tightly connected implying that they belong near one another in the final placement. The AMGC technique proposed in this thesis is based upon AMG, which has been applied with great success in other contexts where large-scale problems arise. AMGC is shown to provide better cluster solutions than state-of-the-art clustering algorithms. To further improve upon AMGC, a length-driven extension, AMGC-LE, is proposed. AMGC-LE makes use of individual net length estimation techniques to provide guidance during the clustering. A length-driven unclustering technique is also proposed to improve the solution quality. Due to the legality-preserving nature of the unclustering technique, it is possible to avoid placement and legalization at all but the coarsest level. AMGC has several benefits including:

- Proven linear-time complexity,
- Improved performance by several measures over other state-of-the-art clustering algorithms,
- Improvement of several existing placers.

Two of the scale-borne problems in the area of CTS are: the large scale of the number of clock sinks and the variations arising from the small scale of the features manufactured in making modern circuits. To cope with the large-scale problem of the design size, a technique is proposed to perform CTS in parallel. The nature of many common CTS algorithms is to work on one branch at a time. Since each branch as an independent subproblem, there are no conflicts in performing the work on several branches in parallel. A parallelization framework is proposed to exploit this property. The framework is applied to several popular CTS algorithms and linear speedup is observed. Theoretical limits of the technique are also derived. The larger the problem size and the lower the asymptotic complexity of the original algorithm, the larger the observed speedup will be. Some of the advantages of the proposed framework include:

- Proven linear asymptotic speedup,
- Application to existing algorithms for each of the main stages of CTS without change,
- Easy implementation.

To deal with the problem of variations, a novel formulation of the clock tree buffer sizing problem is proposed. By a series of transformations, the original non-convex problem is formulated as a GP that can be solved using convex optimization techniques. The formulation can be extended to simultaneously perform wire sizing. In order to size buffers to be robust to variations, the formulation an ellipsoidal uncertainty set is defined. The uncertainty includes spatial correlations in the process variations which provides better solutions than a similar correlation-unaware technique. The worst-case uncertainty is guarded against in the proposed robust formulation. By solving the robust formulation the solutions obtained are comparable to the input networks in terms of skew, but provide substantial savings in terms of power and area. The

skew distributions obtained from simulating the robust networks not only show lower average skew but also lower deviation. The formulations can be used in design scenarios because it allows area and skew to be optimally traded off.

- Obtaining a globally optimal solution,
- Handling various delay models, and
- Incorporating spatially-correlated variations.

In conclusion, innovation is required to address the problems of scale that continue to complicate the process of physical design. This thesis proposes several methods for solving modern, scale-borne problems in VLSI physical design. The major contributions of this thesis are:

- Development of an AMG-based clustering technique,
- Design of the first length estimation-based clustering technique for placement,
- Development of a length-based unclustering algorithm,
- Development of a parallel framework for CTS algorithms,
- Implementation of clock tree buffer sizing as a GP,
- Design and implementation of a robust solution methodology for the clock tree buffer sizing problem.

Appendix A

Complexity Results

A.1 Asymptotic Complexity of Balanced, Recursive $\Theta(N \lg(N))$ Work

Theorem A.1.1. *The time complexity of balanced recursive functions performing* $\Theta(N \lg(N))$ *work at each node is* $\Theta\left(N \lg^2(N)\right)$.

Proof. Without loss of generality, the problem size, N, can be assumed to be a power of two. By the definition of $\Theta(N \lg(N))$, $\exists c \in \mathbb{R}$ such that $cN \lg(N)$ bounds the work at each node from above $\forall N \geq N_0$, for some $N_0 \in \mathbb{Z}$. The sum of work performed at

each level is

$$
\begin{aligned}
& c\left(N\lg(N) + 2\left(\frac{N}{2}\lg(\frac{N}{2})\right) + 4\left(\frac{N}{4}\lg(\frac{N}{4})\right) + \ldots + 2^{\lg(N)}\left(\frac{N}{2^{\lg(N)}}\lg(\frac{N}{2^{\lg(N)}})\right)\right) \\
&= c\sum_{i=0}^{\lg(N)} 2^i\left(\frac{N}{2^i}\lg(\frac{N}{2^i})\right) \\
&= c\sum_{i=0}^{\lg(N)} N\lg(\frac{N}{2^i}) \\
&= cN\sum_{i=0}^{\lg(N)} \lg(\frac{N}{2^i}) \\
&= cN\sum_{i=0}^{\lg(N)} \lg(N) - cN\sum_{i=0}^{\lg(N)} \lg(2^i) \\
&= cN(\lg(N)+1)\lg(N) - cN\sum_{i=0}^{\lg(N)} i \\
&= cN(\lg(N)+1)\lg(N) - cN\left(\frac{\lg(N)(\lg(N)+1)}{2}\right) \\
&= cN\lg(N)\left((\lg(N)+1) - \left(\frac{(\lg(N)+1)}{2}\right)\right) \\
&= cN\lg(N)\left(\frac{(\lg(N)+1)}{2}\right) \\
&= O\left(N\lg^2(N)\right)
\end{aligned}
$$

Furthermore, because the work at each node is $\Theta(N\lg(N))$, the same analysis follows by selecting a constant that bounds the work from below. Thus, the time complexity of the algorithm is $\Theta\left(N\lg^2(N)\right)$. Asymptotic results using recursion trees where the problem size is reduced by a constant factor at each level, in this case two, hold by taking floors and ceiling [8]. This allows the assumption that N is a power of two. □

A.2 Asymptotic Speedup of Balanced, Recursive $\Theta(1)$ Work

Lemma A.2.1. *The asymptotic speedup using the parallelization framework proposed in Section 5.2 is optimal, i.e. linear in the number of processors P, when the work performed at each node is $\Theta(1)$*

Proof. Without loss of generality assume that N and P are powers of two. The speedup can be calculated as the ratio of the total work to the sum of the work performed during Phase 1 and the work of a single Phase 2 subtree. The total amount of work without any parallel processing is bounded by

$$\begin{aligned}
&c\,(1+2+4+\ldots+\lg(N)) \\
&= c\sum_{i=0}^{\lg(N)} 2^i \\
&= c\left(\frac{2^{\lg(N)+1}-1}{2-1}\right) \\
&= c(2N-1),
\end{aligned}$$

for some $c \in \mathbb{R}$. The amount of work in a Phase 2 subtree is $\frac{2cN}{P} - 1$, by substituting $\frac{N}{P}$ for N in the total work equation. The amount of work in Phase 1 is $c(P-1)$ which is obtained by truncating the total work series at $i = \lg(P) - 1$. The speedup is then

$$\begin{aligned}
&\frac{c(2N-1)}{c(P-1)+\frac{2cN}{P}-1} \\
&\frac{P(2N-1)}{P(P-1)+2N-P} \\
&\frac{P(2N-1)}{2N+P(P-2)} \\
&= O(P),
\end{aligned}$$

where the last step follows because the $2N$ terms asymptotically dominate the numerator and denominator. The constant hidden by the Big O notation can be chosen arbitrarily close to one implying the speedup approaches the ideal value of P. □

A.3 Asymptotic Speedup of Balanced, Recursive $\Theta(N)$ Work

Lemma A.3.1. *The asymptotic speedup using the parallelization framework proposed in Section 5.2 is optimal, i.e. linear in the number of processors P, when the work performed at each node is $\Theta(N)$*

Proof. Without loss of generality assume that N and P are powers of two. The speedup can be calculated as the ratio of the total work to the sum of the work performed during Phase 1 and the work of a single Phase 2 subtree. The total amount of work without any parallel processing is bounded by

$$\begin{aligned}
&c\left(N + 2\left(\frac{N}{2}\right) + 4\left(\frac{N}{4}\right) + \ldots + 2^{\lg(N)}\left(\frac{N}{2^{\lg(N)}}\right)\right) \\
&= c\sum_{i=0}^{\lg(N)} 2^i\left(\frac{N}{2^i}\right) \\
&= c\sum_{i=0}^{\lg(N)} N \\
&= cN(\lg(N)+1),
\end{aligned}$$

for some $c \in \mathbb{R}$. The amount of work in a Phase 2 subtree is $\frac{cN}{P}\left(\lg\left(\frac{N}{P}\right)+1\right)$, by substituting $\frac{N}{P}$ for N in the total work equation. The amount of work in Phase 1 is $cN\lg(P)$ which is obtained by truncating the total work series at $i = \lg(P) - 1$. The speedup is then

$$\begin{aligned}
&\frac{cN(\lg(N)+1)}{cN\lg(P) + \frac{cN}{P}\left(\lg\left(\frac{N}{P}\right)+1\right)} \\
&= \frac{P(\lg(N)+1)}{P\lg(P) + \left(\lg\left(\frac{N}{P}\right)+1\right)} \\
&= \frac{P(\lg(N)+1)}{P\lg(P) + \lg(N) - \lg(P) + 1} \\
&= \frac{P(\lg(N)+1)}{\lg(N) + 1 + (P-1)\lg(P)} \\
&= O(P),
\end{aligned}$$

where the last step follows because the $\lg(N)$ terms asymptotically dominate the numerator and denominator. The constant hidden by the Big O notation can be chosen arbitrarily close to one implying the speedup approaches the ideal value of P. □

A.4 Asymptotic Speedup of Balanced, Recursive $\Omega\left(N^2\right)$ Work

Lemma A.4.1. *The asymptotic speedup using the parallelization framework proposed in Section 5.2 is sub-optimal when the work performed at each node is* $\Omega\left(N^2\right)$

Proof. Without loss of generality assume that N and P are powers of two. The speedup can be calculated as the ratio of the total work to the sum of the work performed during Phase 1 and the work of a single Phase 2 subtree. The total amount of work without any parallel processing is bounded by

$$\begin{aligned}
&c\left(N^2 + 2\left(\frac{N^2}{2^2}\right) + 4\left(\frac{N^2}{4^2}\right) + \ldots + 2^{\lg(N)}\left(\frac{N^2}{2^{2\lg(N)}}\right)\right) \\
&= c\sum_{i=0}^{\lg(N)} 2^i\left(\frac{N^2}{2^{2i}}\right) \\
&= cN^2\sum_{i=0}^{\lg(N)} 2^{-i} \\
&= cN^2\left(\frac{1-2^{-(\lg(N)+1)}}{1-2^{-1}}\right) \\
&= cN^2\left(2-2^{-\lg(N)}\right) \\
&= cN^2\left(2-\frac{1}{N}\right),
\end{aligned}$$

for some $c \in \mathbb{R}$. The amount of work in a Phase 2 subtree is $\frac{cN^2}{P^2}\left(2-\frac{P}{N}\right)$, by substituting $\frac{N}{P}$ for N in the total work equation. The amount of work in Phase 1 is $2cN^2\left(1-\frac{1}{P}\right)$ which is obtained by truncating the total work series at $i = \lg(P) - 1$. The speedup is then

$$\begin{aligned}
&\frac{cN^2\left(2-\frac{1}{N}\right)}{2cN^2\left(1-\frac{1}{P}\right)+\frac{cN^2}{P^2}\left(2-\frac{P}{N}\right)} \\
&= \frac{P^2\left(2-\frac{1}{N}\right)}{2P^2\left(1-\frac{1}{P}\right)+2-\frac{P}{N}} \\
&= \frac{P^2\left(1-\frac{1}{2N}\right)}{P(P-1)+1-\frac{P}{2N}} \\
&= O\left(\frac{P^2}{P^2-(P-1)}\right),
\end{aligned}$$

where the last step follows because the terms involving N asymptotically vanish. The asymptotic speedup is decreasing for $P > 2$. $\square$

Bibliography

[1] A. Wolfgang, P. Cogez, et al, "The International Technology Roadmap for Semiconductors," ITRS, Tech. Rep., 2011. [Online]. Available: http://www.itrs.net/Links/2011ITRS/Home2011.htm

[2] Internaional Symposium on Physical Design contest archive. ACM. [Online]. Available: http://archive.sigda.org/ispd/contests

[3] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.

[4] N. Sherwani, *Algorithms for VLSI physical design automation.* Kluwer Publishers, 1999.

[5] C. Alpert, A. Kahng, G. Nam, S. Reda, and P. Villarrubia, "A semi-persistent clustering technique for VLSI circuit placement," in *Proc. of ISPD*, 2005, pp. 200–207.

[6] J. Li, L. Behjat, and A. Kennings, "Net Cluster: A net-reduction-based clustering preprocessing algorithm for partitioning and placement," *Trans. on CAD*, vol. 26, no. 4, pp. 669–679, 2007.

[7] W. Briggs, V. Henson, and S. McCormick, *A multigrid tutorial: second edition.* SIAM, 2000.

[8] T. Corman, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. of DAC*, 1997, pp. 526–529.

[10] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. of DAC*, 1999, pp. 343–348.

[11] C. Alpert, D. Huang, and A. Kahng, "Multilevel circuit partitioning," *Trans. on CAD*, vol. 17, no. 8, pp. 655–667, 1998.

[12] A. Caldwell, A. Kahng, and I. Markov, "Improved algorithms for hypergraph bi-partitioning," in *Proc. of ASPDAC*, 2000, pp. 661–666.

[13] J. Cong and S. Lim, "Edge separability-based circuit clustering with application to multilevel circuit partitioning," *Trans. on CAD*, vol. 23, no. 3, pp. 346–357, 2004.

[14] B. Hu and M. Marek-Sadowska, "Fine granularity clustering for large scale placement problems," in *Proc. of ISPD*, 2003, pp. 67–74.

[15] J. Li and L. Behjat, "A connectivity based clustering algorithm with applications to VLSI circuit partitioning," *Trans. on CAS II*, vol. 53, no. 5, pp. 384–388, 2006.

[16] J.Yan, C. Chu, and W. Mak, "SafeChoice: A novel clustering algorithm for wirelength-driven placement," in *Proc. of ISPD*, 2010, pp. 185–192.

[17] J. Cong, J. Shinnerl, M. Xie, T. Kong, and X. Yuan, "Large-scale circuit placement," *Trans. on DAES*, vol. 10, no. 2, pp. 389–430, 2005.

[18] C. Ding, C. Ho, and M. Irwin, "A new optimization driven clustering algorithm for large circuits," in *Proc. of EURO-DAC*, 1993, pp. 28–32.

[19] C. Alpert and A. Kahng, "Multi-way partitioning via geometric embeddings, orderings, and dynamic programming," *Trans. on CAD*, vol. 14, no. 11, pp. 1342–1358, 1995.

[20] K. Stüben, "Algebraic multigrid (AMG): experiences and comparisons," *Applied Mathematics and Computation*, vol. 13, pp. 419–452, 1983.

[21] J. Ruge and K. Stüben, *Multigrid Methods*, S. F. McCormick, Ed. SIAM, Philadelphia, 1987, chpt. 4 Algebraic Multigrid.

[22] A. Brandt, "Algebraic multigrid theory: The symmetric case," *Applied Mathematics and Computation*, vol. 19, no. 1-4, pp. 23–56, 1986.

[23] Q. Chang, Y. Wong, and Z. Li, "New interpolation formulas of using geometric assumptions in the algebraic multigrid method," *Applied Mathematics and Computation*, vol. 50, no. 2-3, pp. 223–254, 1992.

[24] S. Bodapati and F. Najm, "Prelayout estimation of individual wire lengths," *Trans. on VLSI*, vol. 9, no. 6, pp. 943–958, 2001.

[25] A. Farshidi, L. Behjat, L. Rakai, and B. Fathi, "A pre-placement individual net length estimation model and an application for modern circuits," *Integration*, vol. 44, no. 2, pp. 111–122, 2011.

[26] B. Fathi, L. Behjat, and L. Rakai, "A pre-placement net length estimation technique for mixed-size circuits," in *Proc. of SLIP*, 2009, pp. 45–52.

[27] A. Kahng and S. Reda, "Intrinsic shortest path length: a new, accurate a priori wirelength estimator," in *Proc. of ICCAD*, vol. 2005, 2005, pp. 173–180.

[28] W. Donath, "Placement and average interconnection lengths of computer logic," *Trans. on CAS*, vol. 26, no. 4, pp. 272–277, 1979.

[29] ——, "Wire length distribution for placements of computer logic," *IBM Journal of R&D*, vol. 25, no. 3, pp. 152–155, 1981.

[30] M. Feuer, "Connectivity of random logic," *Trans. on Computers*, vol. C-31, no. 1, pp. 29–33, 1982.

[31] T. Hamada, C. Cheng, and P. Chau, "A wire length estimation technique utilizing neighborhood density equations," *Trans. on CAD*, vol. 15, pp. 912–922, 1996.

[32] H. Heineken and W. Maly, "Standard cell interconnect length prediction from structural circuit attributes," in *Proc. of CICC*, 1996, pp. 167–170.

[33] M. Pedram and B. Preas, "Interconnection length estimation for optimized standard cell layouts," in *Proc. of ICCD*, 1989, pp. 390–393.

[34] Q. Liu, B. Hu, and M. Marek-Sadowska, "Wire length prediction in constraint driven placement," in *Proc. of SLIP*, 2003, pp. 99–105.

[35] A. Farshidi, L. Behjat, L. Rakai, and B. Fathi, "A pre-placement individual net length estimation model and an application for modern circuits," *Integration*, vol. 44, no. 2, pp. 111–122, 2011.

[36] L. Rakai, L. Behjat, S. Martin, and J. Aguado, "An algebraic multigrid-based algorithm for circuit clustering," *Applied Mathematics and Computation*, vol. 218, no. 9, pp. 5202–5216, 2012.

[37] L. Rakai, A. Farshidi, L. Behjat, and D. Westwick, "A new length-based algebraic multigrid clustering algorithm," *VLSI Design*, vol. 2012, 2012.

[38] Q. Chang, Y. Wong, and H. Fu, "On the algebraic multigrid method," *Journal of Computational Physics*, vol. 125, no. 14, pp. 279–292, 1996.

[39] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov, "Unification of partitioning, floorplanning and placement," in *Proc. of ICCAD*, 2004, pp. 550–557.

[40] T. Chan, J. Cong, M. Romesis, J. Shinnerl, K. Sze, and M. Xie, "mPL6: Enhanced multilevel mixed-size placement," in *Proc. of ISPD*, 2006, pp. 212–214.

[41] A. Kahng and Q. Wang, "Implementation and extensibility of an analytic placer," *Trans. on CAD*, vol. 24, no. 5, pp. 734–747, 2005.

[42] G. Nam, C. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ISPD 2005 placement contest and benchmark suite," in *Proc. of ISPD*, 2005, pp. 216–219.

[43] J. Roy and I. Markov, *Partitioning-driven Techniques for VLSI Placement*, ser. Handbook of Algorithms for VLSI Physical Design Automation, C. Alpert, D. Mehta, and S. Sapatnekar, Eds. CRC Press, 2008.

[44] N. Viswanathan, M. Pan, and C. Chu, "Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Proc. of ASPDAC*, 2007.

[45] (2011) Matlab: Version 2011a. [Online]. Available: http://www.mathworks.com/

[46] Q. Zhu, *High-Speed Clock Network Design*. Boston: Kluwer Academic Publishers, 2003.

[47] A. Kahng, J. Lienig, I. Markov, and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer, 2011.

[48] M. Jackson, A. Srinivasan, and E. Kuh, "Clock routing for high-performance ics," in *Proc. of DAC*, 1990, pp. 573–579.

[49] T. Kim and T. Kim, "Clock tree embedding for 3D ICs," in *Proc. of ASPDAC*, 2010, pp. 486–491.

[50] J. Minz, X. Zhao, and S. Lim, "Buffered clock tree synthesis for 3D ICs under thermal variations," in *Proc. of ASPDAC*, 2008, pp. 504–509.

[51] T. Chao, Y. Hsu, J. Ho, K. Boese, and A. Kahng, "Zero skew clock routing with minimum wirelength," *Trans. on CAS*, vol. 39, no. 11, pp. 799–814, 1992.

[52] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proc. of DAC*, 1993, pp. 612–616.

[53] R. Tsay, "Exact zero skew," in *Proc. of ICCAD*, 1991, pp. 336–339.

[54] D. Lee and I. Markov, "CONTANGO: Integrated optimization of soc clock networks," *VLSI Design*, pp. 1–12, 2011.

[55] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *Trans. on CAD*, vol. 24, no. 6, pp. 879–891, 2005.

[56] L. Van Ginneken, "Buffer placement in distributed RC-tree networks for minimal elmore delay," in *Proc. of ISCAS*, 1990, pp. 865–868.

[57] J. Singh, V. Nookala, Z. Luo, and S. Sapatnekar, "Robust gate sizing by geometric programming," in *Proc. of DAC*, 2005, pp. 315–320.

[58] C. Chu and D. Wong, "Closed form solution to simultaneous buffer insertion/sizing and wire sizing," *Trans. on DAES*, vol. 6, no. 6, pp. 343–371, 2001.

[59] J. Lillis, C. Cheng, and T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. of ICCAD*, 1995, pp. 138–143.

[60] A. Jagannathan, S. Hur, and J. Lillis, "A fast algorithm for context-aware buffer insertion," in *Proc. of DAC*, 2000, pp. 368–373.

[61] M. Lai and D. Wong, "Maze routing with buffer insertion and wiresizing," in *Proc. of DAC*, 2000, pp. 374–378.

[62] Ngspice 24. http://ngspice.sourceforge.net/.

[63] K. Wang, Y. Ran, H. Jiang, and M. Marek-Sadowska, "General skew constrained clock network sizing based on sequential linear programming," *Trans. on CAD*, vol. 24, no. 5, pp. 773–782, 2005.

[64] M. Guthaus, D. Sylvester, and R. Brown, "Clock buffer and wire sizing using sequential programming," in *Proc. of DAC*, 2006, pp. 1041–1046.

[65] D. Lee and I. Markov, "Multilevel tree fusion for robust clock networks," in *Proc. of ICCAD*, 2011, pp. 632–639.

[66] Y. Chang, C. Wang, and H. Chen, "On construction low power and robust clock tree via slew budgeting," in *Proc. of ISPD*, 2012, pp. 129–136.

[67] B. Wong, A. Mittal, Y. Cao, and G. Starr, *Nano-CMOS Circuit and Physical Design*, 1st ed. Wiley-Interscience, 2004.

[68] N. Matloff, *Programming on Parallel Machines*. University of Calfornia, Davis, 2011.

[69] J. Shen and M. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Professional, 2005.

[70] A. Tanenbaum, *Modern Operating Systems*, 2nd ed. Pearson Prentice Hall, 2007.

[71] H. El-Rewini and T. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

[72] G. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *Proc. of AFIPS*, 1967, pp. 483–485.

[73] J. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[74] D. Knuth, *The Art of Computer Programming*, 3rd ed. Addison-Wesley, 1997, vol. Volume 1: Fundamental Algorithms.

[75] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[76] A. Ben-Tal and A. Nemirovski, "Robust solutions of linear programming problems contaminated with uncertain data," *Mathematical Programming*, vol. 88, pp. 411–424, 2000.

[77] D. Bertsimas, D. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, pp. 464–501, 2011.

[78] Mosek 6.0. http://www.mosek.com.

[79] S. Boyd, S. Kim, D. Patil, and M. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, pp. 899–932, 2005.

[80] M. Mani, A. Singh, and M. Orshanksy, "Joint design-time and post-silicon minimization on parametric yield loss using adjustable robust optimization," in *Proc. of ICCAD*, 2006, pp. 19–26.

[81] J. Yan and C. Chu, "DeFer: Deferred decision making enabled fixed-outline floorplanning algorithm," *Trans. on CAD*, vol. 29, no. 3, pp. 367–381, 2010.

[82] ISPD 2009 clock network synthesis contest. ACM. http://ispd.cc/contests/09/ispd09cts.html.

[83] ISPD 2010 high performance clock network synthesis contest. ACM. http://archive.sigda.org/ispd/contests/10/ispd10cns.html.

[84] Z. Xing and P. Banerjee, "A parallel algorithm for zero skew clock tree routing," in *Proc. of ISPD*, 1998, pp. 118–123.

[85] S. Olivier and J. Prins, "Evaluating OpenMP 3.0 run time systems on unbalanced task graphs," in *Proc. of IWOMP*, 2009, pp. 63–78.

[86] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *JCSS*, vol. 7, pp. 448–461, 1973.

[87] The OpenMP API specification for parallel programming. http://www.openmp.org.

[88] S. Boyd and S. Kim, "Geometric programming for circuit optimization," in *Proc. of ISPD*, 2005, pp. 44–46.

[89] H. Chang and S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *Proc. of ICCAD*, 2003, pp. 621–625.

[90] K. Kasamsetty, M. Ketkar, and S. Sapatnekar, "A new class of convex functions for delay modeling and its application to the transistor sizing problem," *Trans. on CAD*, vol. 19, no. 7, pp. 779–788, 2000.

[91] H. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Addison–Wesley, 1990.