Overcoming Physical Design Challenges in Nanometer-Scale Integrated Circuits

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Yaoguang Wei

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Sachin S. Sapatnekar

February, 2013

Acknowledgements

First of all, I would like to express my sincere and deep gratitude to my advisor, Professor Sachin S. Sapatnekar, for his consistent guidance and support in my PhD studies. His strictness in doing research and attitude of cherishing the time to work hard have affected me significantly. He has taught me how to think logically, how to use my mind wisely, and how to plan realistically. I will treasure all these for my whole lifetime.

I am also grateful to Professors Kia Bazargan, Chris H. Kim, and Antonia Zhai, for serving on my PhD degree committee, reviewing my thesis and providing valuable feedback.

In the research process, I have had cooperations with the following people and owe many thanks to all of them: Charles J. Alpert, Andrew D. Huber, Jiang Hu, Shiyan Hu, Douglas Keller, Zhuo Li, Frank Liu, Lakshmi Reddy, Cliff Sze, Gustavo E. Tellez, and Natarajan Viswanathan. They have helped me in many ways, including polishing the writing of my papers, providing constructive suggestions, and/or helping to integrate some algorithms proposed in this thesis to the IBM design flow.

Furthermore, I want to thank many other people who helped me in various ways, such as conducting useful research-related discussions and providing helpful suggestions for coding: Kanak B. Agarwal, Glenn R. Bee, Laleh Behjat, Baktash Boghrati, Jianxin Fang, Song Guo, Saket Gupta, Wen-hao Liu, Dirk Müller, Gi-Joon Nam, Sani Nassif, Weikang Qian, Gregory M. Schaeffer, Meng Wang, Pingqiang Zhou, Ying Zhou, and many others.

A great amount of thanks are also due to Semiconductor Research Corporation for funding my research, as well as to the IBM Austin Research Laboratory (ARL) for providing me the opportunity to work as an intern. In particular, the Design Productivity Group led by Charles J. Alpert in ARL has provided great support for the research projects that I have worked on, and these projects have contributed to my thesis significantly.

Finally, my warm thanks go to my family. They have encouraged and supported me in overcoming the difficulties in my journey to obtain the PhD degree. I would like to especially thank my wife Na, who has accompanied me and greatly helped me in many aspects throughout these years.

Dedication

To my parents, my wife, and my daughter.

Abstract

Through aggressive technology scaling over the past five decades, integrated circuit design has entered the nanometer-scale era. While scaling enables the design of more powerful chips, circuit designers must face numerous challenges that accompany these miniscule feature sizes. Many of these issues are expressed in the step of physical design, an important back-end stage in the integrated circuit design flow. First, the routability of a design becomes an increasingly important and difficult problem, and must be addressed across the entire physical synthesis tool stack. This in turn requires effective routability evaluation methods to be used in the early stages for congestion mitigation. Second, wire delays do not scale down well with process technology, and have exceeded the gate delay in importance, becoming the dominating factor that determines the circuit delay. Wire delays can be reduced by inserting large numbers of buffers, but these can significantly increase the chip area, cost, and power, so that improved methods that control these costs are essential. Third, with shrinking feature sizes, the impact of process variations has become more serious than before. Several important process variation effects show strong dependencies on the underlying patterns on the die, and these challenges can be addressed effectively through appropriate physical design. This thesis presents solutions to these challenges.

To achieve effective routability evaluation, we first analyze the problems associated with mainstream global-routing-based congestion analysis tools. Two major deficiencies of existing approaches are: (i) they do not adequately model local routing resources, which can cause incorrect routability predictions that are only detected late, during detailed routing, (ii) the metrics used to represent congestion may yield numbers that do not provide sufficient intuition to the designer; moreover, they may often fail to predict the routability accurately. We propose solutions for both problems. First, we develop an efficient, accurate and scalable local routing resource model. Experiments demonstrate that our model improves the accuracy of a congestion analyzer and enables designers to use a coarser grid to speed up congestion analysis and achieve similar accuracy as the baseline case. Second, we develop a new metric that represents the congestion map for the chip with high fidelity. Experiments show that compared with conventional metrics, the new metric can predict the routability more accurately and can drive a placer to obtain a design that has better routability characteristics.

To reduce the buffer usage, we make full use of the timing benefits brought by the thick metal layers. In advanced technologies, a larger number of metal layers with thick cross-sections are available for routing. These metal layers have much smaller wire delays than thinner layers, and assigning nets to these layers can improve timing and save buffer usage. However, existing algorithms have various limitations in using thick metal layers. In this work, we propose a novel algorithm to address the issue. Our algorithm tries to assign as many nets as possible to thick metal layers to maximize the timing benefits while simultaneously using heuristics to control the congestion at a manageable level. We also present a new physical synthesis flow that adds our algorithm as a new component at an early stage of an existing industrial design flow. Experimental results demonstrate the effectiveness of our algorithm and flow on a set of industrial designs.

To overcome the challenges from process variations, this thesis presents physical design solutions to two important types of variations induced in the processes of oxide chemical mechanical polishing (CMP) and rapid thermal annealing (RTA). First, since the oxide CMP variation highly depends on the metal pattern density, a common practice to reduce CMP variation is to insert dummy fills. However, dummy fills have side effects on design performance or complexity and should be minimized. Therefore, we propose a novel global routing algorithm directly aiming to minimize the amount of dummy fills necessary to satisfy the requirements for CMP. Since it is not computationally efficient to directly minimize the amount of dummy fills in the routing process, we develop a surrogate optimization objective through theoretical analyses and experiments. Then effective cost functions are elaborated and applied in the routing process to optimize the surrogate metric. Our strategy and algorithm is validated by the experiments on a standard set of benchmark circuits. Second, since RTA variation strongly depends on the density of the STI regions, to minimize RTA variation, this thesis proposes a two-step approach to maximize the uniformity of the STI density throughout the layout. We introduce a concept of effective STI density and propose an efficient incremental method to compute it for the whole circuit. Furthermore, we enhance a conventional floorplanner to handle the new objective of minimizing the variations in effective STI density, using a two-stage simulated annealing heuristic. As the second step of our efforts, we insert dummy polysilicon fills to further minimize the variation in effective STI density. Experimental results demonstrate that our methods can significantly reduce the RTA variations.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since the integrated circuit (IC) was invented around 1958 [6], IC chips have roughly obeyed Moore's law and have maintained a trend of becoming cheaper, denser, and more complex and powerful. Technology scaling, which has shrunk the dimensions of transistors by half every two years or so, has improved the performance of the very large-scale integrated (VLSI) circuits by five orders of magnitude in the past four decades [7]. However, increased circuit sizes and nanoscale effects have caused circuit design to becoming more complex and difficult with time. To counteract these effects, computer-aided design (CAD) tools are playing an increasingly important role in circuit design. CAD tools can significantly improve the productivity of circuit designers by helping to automate the design process and also decrease the cost of IC chips. CAD tools typically work along a design flow, where the circuit design tasks are performed in a succession of stages, with various CAD tools supporting each stage (readers are referred to [8] for a detailed discussion).

Physical design is an important back-end stage in the circuit design flow, and determines the physical locations of gates and the physical paths for interconnections on a die. Decisions made at this stage can significantly affect circuit performance. An inferior physical design solution could cause a circuit to have timing violations, have incorrect functionality (due to incomplete electrical connections), or fail to satisfy the specifications after manufacturing. As technology advances, a series of new challenges is emerging in the area of physical design. Firstly, with shrinking feature sizes, the number of gates in a circuit increases exponentially with technology generations. The typical VLSI circuit today may contain many millions of gates, and such large sizes challenge the scalability of physical design tools. A related issue is that while

the die size is kept similar or even increased, the routing resources are becoming scarcer, and routability has become a key issue in modern physical design. Secondly, in the nanometer era, interconnect delays can exceed the gate delay and become the dominating factor that determines circuit delay. This dramatically changes the entire physical design flow to weigh more heavily on interconnection optimization than before [9]. Though this effect can be mitigated through buffer insertion, an effective way to reduce wire delay, inserting large numbers of buffers also increases the chip area, cost, and power. Thirdly, the reductions in transistor sizes make circuits more sensitive to variations from the manufacturing process than before. Many such variations are affected by physical design solutions, and can be mitigated by being considered in the physical design flow. To help overcome these challenges, this thesis addresses several issues in this domain and proposes physical design techniques for effective routability evaluation, buffer usage reduction, and process variation optimization. In the remainder of this chapter, we will discuss these issues in more detail and highlight the contributions of this thesis.

1.1 Achieving effective routability evaluation

The problem of achieving routability is becoming increasingly important with the explosion in design rules and design for manufacturability requirements that multiply with each technology node. An unroutable design is not useful even it closes all other design metrics, and fast design closure can only be achieved by addressing the routability of a design at all stages in the design cycle. However, traditional approaches have considered such issues only late in the physical design flow, and there is a strong need for building effective routability evaluation methods, which drive routability optimization, in early stages of physical design. Routability evaluation has two key components: (a) the method used to analyze the congestion of a given placement, and (b) the metric(s) used to score or represent the congestion. In recent years, with the advent of fast, high-quality academic global routers [10–14], global-routing-based methods [15–17] have become the mainstream technique for routability evaluation due to their efficiency and reasonable accuracy. However, there are two major drawbacks in these methods:

- These approaches ignore the local routing resources used in detailed routing step due to the higher abstraction level in global routing, and tend to underestimate the real congestion seen in detailed routing.

- The metrics used to represent congestion may yield numbers that do not provide sufficient intuition to the designer; furthermore, they may often fail to predict the routability accurately.

These factors significantly affect the effectiveness of the routability evaluation and must be addressed.

The first contribution of this thesis, described in Chapter 2, presents solutions to both issues. First, we propose three new approaches to model local routing resources. Second, we develop a new metric that represents the congestion map of a design with higher fidelity than past approaches. The impacts of our proposed techniques are demonstrated on several industrial circuits:

1. With our proposed local routing resource modeling, a congestion analyzer can significantly improve the accuracy in routability evaluation, compared with an analyzer without local resource modeling.

2. Our proposed methods also have good scalability, which enables the use of a coarser grid in congestion analysis so that the runtime of congestion analysis can be improved by 66% on average, compared with the case with a smaller grid size.

3. We demonstrate that our method can greatly improve the correlation between congestion analysis and detail routing, compared with a method without local resource modeling.

4. Our proposed metric can predict the routability correctly on a design while the previously proposed metric cannot.

5. When incorporated within a congestion mitigation tool, our new metric can perform much better than other conventional metrics to improve the design routability.

Furthermore, the proposed techniques have been used in an industrial physical synthesis (another name for modern physical design [9]) flow, which shows their practicality.

1.2 Reducing buffer usage

As process technologies shrink to finer geometries, the metal resistance on low metal layers worsens. The resulting increase in the interconnect delay makes design closure much more

difficult. This can be mitigated partially through buffer insertion, but may result in a massive increase in buffer resources and power. To counteract this trend, more and more higher metal layers, with thicker cross-sections and lower resistances, have been introduced. Fig. 1.1 shows an example for the evolution of the metal layer stacks in an IBM technology [2]. We can see that both the number of metal layers and the thickness of upper layers tend to increase with each technology.

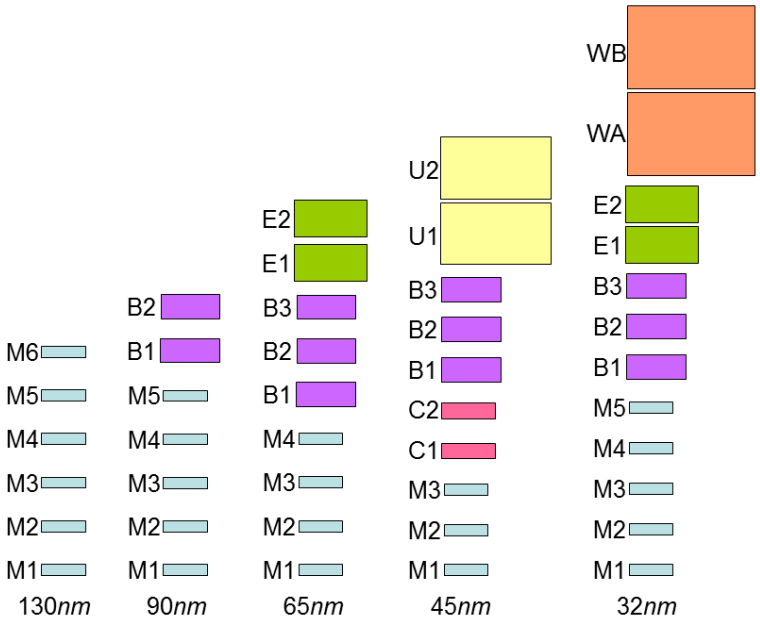


Figure 1.1: An example of possible metal layer stacks in an IBM technology [2].

As stated above, thicker metal layers have smaller unit-length wire delay than thinner layers. For example, on the 2× [4×] layer, signals can roughly go 1.7× [2.5×] faster, with 2× [4.4×] reduction in buffer resources. Assigning nets to thick layers can potentially improve timing and reduce buffer count. This reduction in wire delay in thicker layers provides another dimension to timing optimization, beyond gate/wire sizing and buffering [18]: now interconnect nets may vie for these low-resistance wires to meet timing, but under constraints dictated by the limited availability of these resources. Existing physical synthesis tools typically do not exploit this degree of freedom well and are usually not effective in handling these new thick layers for design closure.

The second contribution of this thesis, described in Chapter 3, is to solve the problem by adding a new component to an industrial physical synthesis flow. A new algorithm is embedded to perform congestion- and timing-aware layer assignment. The algorithm aims to maximize the timing benefits of thick metal layers and to minimize buffer usage by assigning as many nets as possible to thick metal layers while controlling congestion well. Experiments demonstrate the effectiveness of the proposed algorithm. In particular, our algorithm can save the buffer area by 10% on average and up to 18%, while maintaining the similar congestion, timing, and runtime. These savings could help to improve the design power and cost, and help to achieve effective design closure.

1.3 Process variation optimization

Variations in the IC manufacturing process, which include disparities in the processing temperatures between different locations on the wafer, fluctuations in the resist thickness across the wafer, and aberrations in the stepper lens, may cause the parameters of the fabricated circuit different from their intended values [19]. Such variations, usually referred to as *process variations*, may cause the circuit to be unable to satisfy the specifications, and even worse, cause circuit failure [20].

An important category of process variations are within-die variations that are highly dependent on distribution of underlying patterns (devices or metal wires) on the die. Two important examples of such variations are those induced during chemical mechanical polishing (CMP) and rapid thermal annealing (RTA), which largely depend on the metal wire density and shallow trench isolation (STI) density, respectively. Since physical design determines the final distribution of the devices (in floorplanning and placement steps) and the metal wires (in routing steps), it can significantly impact the effects of these variations. In the rest of this thesis, we propose physical design techniques to optimize the variations induced by CMP and RTA.

1.3.1 Optimizing dummy fill for CMP-induced variations

CMP is used in the manufacturing process to polish the wafer whenever a planar surface is required. In a typical CMP tool illustrated in Fig. 1.2, the wafer is mounted to a rotating head (i.e., wafer carrier), and then is pressed against the polishing pad mounted on a rotating table. In addition, a slurry (composed of particles suspended in a chemical solution) is deposited on

the pad as the chemical abrasive. CMP uses both chemical and mechanical means to polish wafer [3].

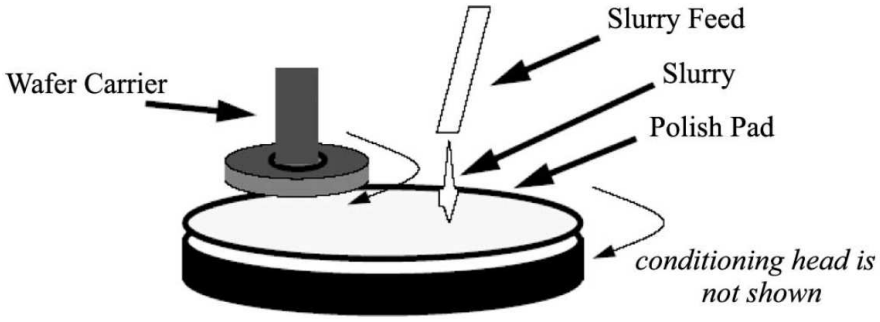


Figure 1.2: A typical CMP tool [3].

As a step in the manufacturing process, oxide CMP is used to polish the interlayer dielectric (ILD) layer to ensure a near-planar surface before depositing and patterning a metal layer. Ideally, it is desirable for the post-CMP ILD thickness across the chip to be uniform. However, due to nonidealities in the CMP process, such as the bent CMP pad, the post-CMP ILD thickness may have variations. This surface topography variation could result in defocusing during lithography, which leads to the variations in wire width and thickness. Such interconnect variations can greatly impact on circuit performance and yield. According to the work in [5], oxide CMP variation is highly dependent on the underlying pattern density. Therefore, to reduce the oxide CMP variation, dummy metal shapes (also called *dummy fills*) are inserted to improve the uniformity of the pattern density. However, dummy fills also bring side effects such as increased coupling capacitance or enlarged routing difficulty [3]. Therefore, dummy fills should be minimized.

The third contribution of this thesis, described in Chapter 4, is to propose a novel global routing algorithm directly aiming to minimize the amount of dummy fill necessary to satisfy the planarity requirements for CMP. Since it is not computationally efficient to directly minimize the amount of dummy fill in the routing process, we develop a surrogate metric through a set of theoretical analyses and experiments. Then effective cost functions are elaborated and applied in the routing process to optimize the surrogate metric. The effectiveness of our strategy and algorithm is validated by the experiments on a standard set of benchmark circuits. Our CMP-aware router can reduce the required dummy fill by 22% on average, and up to 42%, as compared to the CMP-unaware case. In comparison with another CMP-aware routing approach,

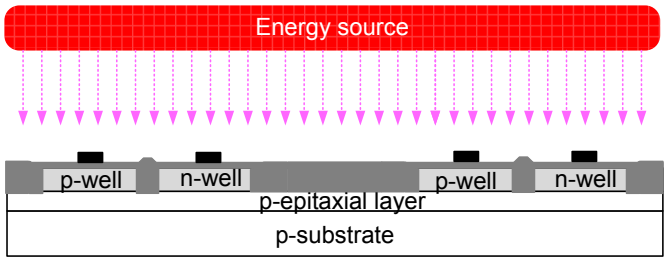
our algorithm is demonstrated to reduce the amount of dummy fill by 14% on average, and up to 24%, over the benchmarks.

1.3.2 Minimizing RTA-induced variations

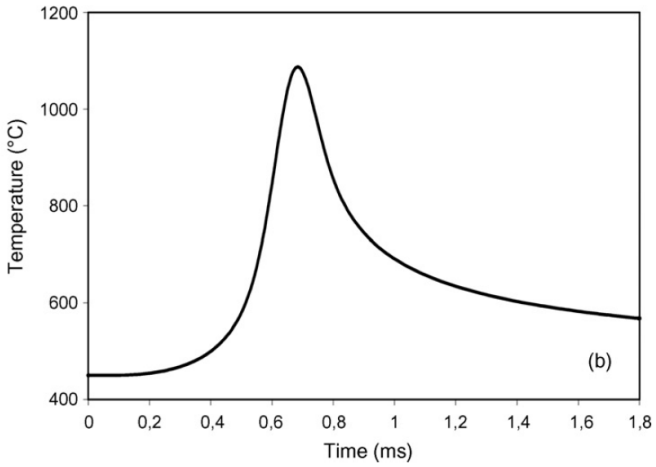
RTA is an annealing process conducted after ion implantation to activate the dopants [21]. A schematic diagram for a RTA process is shown in Fig. 1.3(a). In a RTA process, a energy source, such as scanned laser or flash lamps, is applied on top of the wafer to rapidly heat the wafer surface to achieve a high temperature around 1000°C within a brief period, e.g., several milliseconds. Next, the energy source is shut down and the wafer is quickly cooled down. Usually, a preheating process is used to heat the wafer to a medium temperature around 500°C before annealing to reduce the thermal shock [22]. Fig. 1.3(b) shows the temperature profile at a point of the silicon surface during the laser RTA process in [23].

In RTA, the annealing temperature greatly impacts the extent of dopant activation, which in turn significantly affects several important circuit parameters, such as the threshold voltage and source/drain extrinsic resistance. Therefore, uneven annealing temperatures in RTA across the die will cause large variations in the circuit performance and power. It has been demonstrated in [1] that RTA-induced variability strongly depends on circuit layout patterns, particularly the distribution of the density of the STI regions. Therefore, a primary mechanism for reducing the RTA-induced variations is to obtain an even distribution of STI density throughout the layout.

The fourth contribution of this thesis, described in Chapter 5, is to investigate a two-step approach to reduce the impact of RTA-induced variations. We first solve a floorplanning problem that aims to reduce the RTA variations by evening out the STI density distribution. Next, we insert dummy polysilicon fills to further improve the uniformity of the STI density. Experimental results show that our floorplanner can reduce the global RTA variations by 39% and the local variations by 29% on average with low overhead compared to a traditional floorplanner, and the proposed dummy fill algorithm can further reduce the RTA variations to negligible amounts. Moreover, when inserting dummy fills, for the layouts obtained by our floorplanner, on average, 24% fewer dummy polysilicon fills are inserted, as compared to the results from a traditional floorplanner.



(a) A schematic diagram for a RTA process.



(b) The temperature profile at a point of the silicon surface during the laser RTA process in [23].

Figure 1.3: Illustrations for a RTA process.

Chapter 2

Techniques for Scalable and Effective Routability Evaluation

2.1 Introduction

Routability has become an increasingly important and difficult issue in nanometer-scale VLSI designs, and must be addressed across the entire physical synthesis tool stack. This in turn requires fast, yet reasonably accurate, techniques to identify routing-challenged regions (hot spots) for routability optimization. This chapter focuses on the two key components of routability evaluation: (a) the method used to analyze the congestion of a given placement, and (b) the metric(s) used to score or represent the congestion.

2.1.1 Congestion analysis techniques

Congestion analysis is related to, but different from, routing. The basic purpose of routing is to find the paths to connect all the nets to achieve the correct electrical connection of the circuit, while the goal of congestion analysis is to predict the routability, identify routing hot spots, and provide designers or optimization tools fast feedback on the congestion to improve the routability. A thorough discussion about the differences between congestion analysis and routing can be found in [2].

Routing is traditionally divided to two stages due to its complexity: global routing and then detailed routing. In the global routing stage, the routing region is divided to global routing cells

(g-cells), and only the g-cell-to-g-cell paths are computed for all the nets (see Section 2.2 for more details). Next, detailed routing computes the pin-to-pin connection for all nets, guided by the coarse paths from global routing. Further, detailed routing is constrained by complex design rules, which are usually ignored by global routing, in order to ensure manufacturability.

Typical approaches for congestion analysis can be categorized as follows:

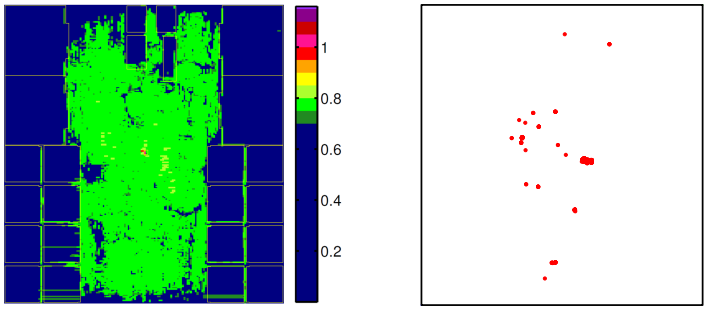
1. Taking a design through detailed routing to determine whether it is routable or not.
2. Using a probabilistic congestion estimation procedure, without performing any routing [24, 25].
3. Performing fast global routing and using its solution to perform congestion analysis [15–17].

In principle, an approach based on detailed routing estimates is the most accurate, but is very time-consuming and impractical during the early stages of design closure. Probabilistic methods are highly inaccurate and fail to capture the behavior of global routing, especially in modern designs with numerous IP blockages, and a large number of metal layers with varying width and spacing. Lately, the third method has become more attractive and mainstream due to the advent of fast, high-quality global routers [10–14].

Although global-routing-based congestion analysis provides a happy medium between probabilistic analysis and detailed routing, it suffers from two key drawbacks.

The first drawback in global-routing-based congestion analysis is that global routing solutions cannot effectively predict the problematic regions shown in detailed routing, since they do not effectively model local routing congestion. Here, *local routing congestion*, or simply, *local congestion* refers to the congestion that does not appear in global routing but shows up in detailed routing. This mismatch appears mainly because the *local routing resources*, or simply, *local resources*, used for *local routing* in detailed routing are not modeled in global routing (see Section 2.3.1 for detailed discussions).

Roughly speaking, local routing, refers to the routing performed in one g-cell. Local routing clearly consume varying amounts of routing resources depending on factors such as design rules, the size of the g-cell, and pin density. Fig. 2.1 demonstrates a concrete instance where ignoring local resources in global routing can significantly mispredict design routability. Without considering local resources, the congestion analyzer only sees very few congestion hot spots in



(a) Combined congestion map from a global-routing-based congestion analyzer.

(b) Open/short map from a detailed router.

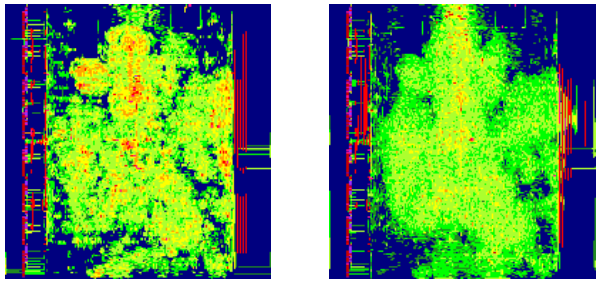
Figure 2.1: Without considering local resources, global-routing-based congestion analyzers cannot predict the locations of opens/shorts well.

the "combined" congestion map[1] (Fig. 2.1(a)), and cannot predict the locations of opens/shots in detailed routing (the red dots in Fig. 2.1(b)), where detailed routing cannot complete. Hence, to achieve more accurate routability evaluation, local resources must be modeled in global routing. Further, any such method should be *flexible* enough to enable it to be adjusted in a straightforward manner from one technology to the next (as design rules are different for each technology). It is also desirable that the local resource model has good scalability on g-cell size so that it can be applied with different g-cell sizes.

The other drawback in global-routing-based congestion analysis is that the routing solutions from the congestion analyzers tend to have hot spots surrounded by noncongested spots, called "noise" hot spots (further discussed in Section 2.4), that could usually be diluted by further routing efforts, and such noise hot spots brings inaccuracy to the congestion analysis. The noise hot spots appear mainly because of the following factor. Due to a large number of invocations in the design flow, congestion analysis tools tend to run very fast by limiting the routing efforts, e.g., by limiting the extent to which each net can detour [2]. Then it is likely that the noise hot spots can be diluted by further routing efforts. For example, Fig. 2.2 shows the congestion maps on the design ckt_i from an industrial routing tool in two modes: congestion analysis mode and

[1]The "combined" congestion map combines all the layers by showing the maximal congestion among all the layers. All the congestion plots in this thesis without a qualifier show the combined maps. Moreover, the color map shown in Fig. 2.1(a) applies to all the subsequent congestion plots skipping a color bar.

global routing mode. We observe that in Fig. 2.2(a), there are quite a few noise hot spots and they are dissolved by further routing efforts in global routing mode (Fig. 2.2(b)). These hot spots are "noise" that prevents us obtaining more accurate routability evaluation, and should be addressed properly.



(a) Congestion analysis mode (b) Global routing mode

Figure 2.2: Congestion maps from an industrial routing tool in two modes.

2.1.2 Metrics to score or represent congestion

Visual inspections of congestion plots, or congestion maps, often serve as a first-order method to compare the routability of different design points. However, optimization tools and designers also require a single metric that can accurately score or represent the design congestion. Commonly-used metrics in academia and industry can be categorized as follows:

Overflow-based metrics include total overflow and maximal overflow that measure the excess of the routing usage over routing capacity on the global routing edges in a global routing graph (defined in Section 2.2). These metrics do not provide sufficient intuition (e.g., how good/bad is an overflow of $14,253$?), which makes it difficult to quantify how much better one design point is versus another. Further, they may even fail to predict the routability correctly in some cases, as will be demonstrated in Section 2.6.3.

Net-congestion-based metrics [2] include[2]: (a) $ACN(x)$, the average net congestion, defined as the average congestion of the top $x\%$ congested nets, where the congestion of a net is the maximum congestion among all the global routing edges traversed by the net. (b) $WCI(y)$, the worst congestion index, defined as the number of nets with congestion greater than or equal to

[2]We name the metrics differently from [2] to facilitate later references.

$y\%$. In practice, $ACN(20)$, $WCI(90)$ and $WCI(100)$ have been used to evaluate routability. The main issue with these metrics is that they fail to differentiate between a net spanning a single congested global routing edge and one that spans multiple congested edges.

In this work, we propose techniques to enhance the accuracy and effectiveness of routability evaluation. Our key contributions include:

- A study of the inaccuracies in existing global-routing-based congestion analyzers, specifically due to the lack of local routing resource modeling and the existence of noise hot spots in the congestion maps.
- An analysis of the weaknesses in existing metrics to score or represent design congestion.
- Methods to model and incorporate the effects of local routing resource usage during global routing. Compared with approaches without modeling local routing resources, our methods improve congestion analysis in three aspects: (a) significant improvement in the accuracy of congestion analysis, (b) better prediction of detailed routing issues such as opens and shorts, and (c) accelerated congestion analysis with a larger g-cell size.
- A smoothing technique that could reduce the noise in congestion maps and further improve the accuracy of routability evaluation.
- A new congestion metric that provides better intuition and represents the design congestion with high fidelity. This metric has been used in DAC 2012 placement contest [26].
- Detailed empirical validation of our proposed techniques on advanced industrial designs.

The rest of this chapter is organized as follows. We begin by presenting background and definitions in Section 2.2. Next, we present our methods for modeling local routing congestion in Section 2.3, and discuss the smoothing technique proposed to filter out the noise in routability evaluation in Section 2.4, followed by a description of our new metric for routability evaluation in Section 2.5. Empirical validation and concluding remarks are provided in Sections 2.6 and 2.7, respectively.

2.2 Preliminaries

Typically, during global routing, the chip is tessellated into $n_r \times n_c$ grids (or *g-cells*), and the global routing graph (GRG), $G_r = (V_r, E_r)$, is constructed. A node in V_r represents a g-cell

in the layout, and an edge (called a *g-edge*) in E_r denotes the boundary between two adjacent g-cells. An example of the GRG is shown in Fig. 2.3.

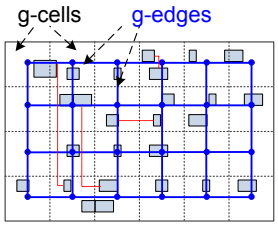


Figure 2.3: Global routing graph (GRG).

We now introduce some notations and terms that will be used in the remainder of this thesis. For each edge e in the GRG, we define c_e as edge capacity — the total or maximal capacity of the edge, b_e as blockage usage, and w_e as the routing demand on the edge. In global routing, c_e, b_e and w_e are generally expressed in the number of routing tracks, where a routing track is the routing resource taken by a single wire passing through an edge in the GRG. We further define total routing usage u_e as sum of b_e and w_e. Then the overflow of an edge e can be defined as:

$$o_e = \max(u_e - c_e, 0). \tag{2.1}$$

The total overflow (TOF) of the layout can be given by $\sum_{e \in E_r} o_e$, and the maximal overflow (MOF) is given by $\max_{e \in E_r} o_e$. The congestion of edge e, denoted as g_e, is given by $g_e = u_e / c_e$.

2.3 Local congestion modeling

In this section, we will analyze the problems associated with existing congestion analysis methods, discuss the sources of local resources, review the previous works related to local resource modeling, and finally propose our methods for local resource modeling. Note that, for convenience, in this chapter we will use the terms “local resource modeling” and “local congestion modeling” interchangeably.

2.3.1 Limitations of existing global-routing-based methods

As mentioned in Section 2.1, global-routing-based congestion analysis is now mainstream. Examples include, FastRoute [27] and NTHU-Route 2.0 [11], used as congestion analyzers within

routability-driven placers IPR [15] and CRISP [16], respectively. However, the major problem in these academic global routers or congestion analyzers is in their inability to model local resource usage, which could lead to the inaccurate prediction of routing hot spots (opens/shorts) in detailed routing.

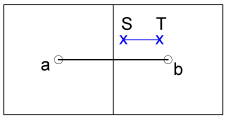
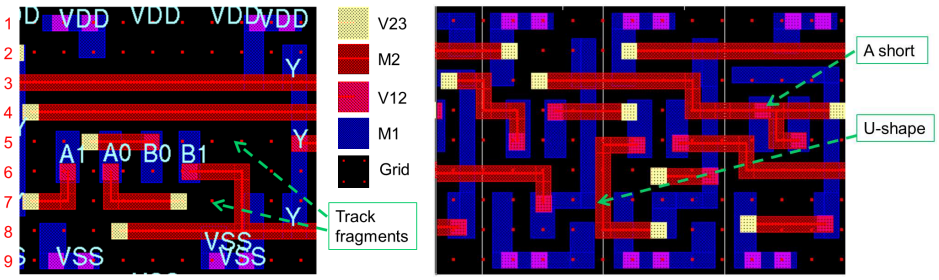


Figure 2.4: Local nets ignored by traditional global routers.

There are two major consumers of local resources. The first of these are the wires used to connect the *local (sub-) nets*, whose pins are all inside a single g-cell. As illustrated in Fig. 2.4, the two-pin net (S, T) is a local net in the g-cell centered at b, and the local wire connecting S to T is not modeled in global routing. For convenience, we denote these kinds of resources as **local-net resources**. Traditional global routers generally abstract the routing problem and only focus on g-cell-to-g-cell routing, while the resources used by local nets are ignored.

The second set of consumers, which we refer to as **pin-access resources**, are the resources used for pin access in detailed routing, which are also ignored in traditional global routing. Fig. 2.5(a) shows a standard cell with five signal pins, shown as blue shapes on metal layer 1 (M1). If we ignore these pins, nine horizontal tracks (marked with numbers 1–9), on metal layer 2 (M2), are available for global routing in this region. However, from detailed routing results shown in Fig. 2.5(a), we can see that horizontal track 5–7 will be mostly blocked in the region for pin access and are no longer available for global routing. Note that only counting the area of wires on layer M2 connecting to pins as local resources used by pins is not enough, since pin access causes many track fragments between the short wires connecting to pins (Fig. 2.5(a) shows two examples), which are hard to use for routing and should also be amortized to the pin-access resources. In addition, pin-access resources also depend on the pin distribution. The closer pins are packed together, the more difficult the detailed router tends to access all the pins, since the wires connecting to some pins could block the pin access to other pins and detour may become necessary in such a case. Therefore, more resources could be consumed than the case where the pins are packed more loosely. As an example, Fig. 2.5(b) shows the pin access for three standard cells with larger pin density than the cell shown in Fig. 2.5(a). Due to denser pin distribution, pin access becomes more difficult in Fig. 2.5(b), and therefore, zigzag wires and

U-shaped wires have to be used to access the pins, which take more resources than flat wires or L-shaped wires used for pin access shown in Fig. 2.5(a). Even worse, a short is caused as shown in Fig. 2.5(b).



(a) Pin access blocks track 5–7.

(b) Pins with large density consumes many routing tracks and a short even occurs.

Figure 2.5: Pin access consumes considerable local routing resources. In the legends, "V12" denotes the vias from M1 to M2, and "V23" the vias from M2 to M3.

In summary, ignoring these local routing resources caused by local nets and pin access will incorrectly make global routing see more tracks than are available, resulting in inaccurate routability evaluation. This motivates the problem of modeling local resources, or local congestion, in global routing. Next, we will first review the previous works for local resource/congestion modeling, and then present our algorithms to address this problem.

2.3.2 Review of previous works for local congestion modeling

Since the fundamental task of routing is to connect the pins of the same net, pin density is closely related to routing congestion, and high pin density is usually correlated with high routing congestion [28]. Therefore, pin density has become a key factor to optimize in many placers [16, 29, 30]. The general idea involved in those works is to spread the cells so that the pin density is not high in a region. In the routing stage, pin density is also used as a metric to drive routers to achieve more uniform wire distribution to reduce the variations in the chemical mechanical polishing process [31, 32]. Though these works have used pin density as a tool to drive placers or routers in optimization, none of them explicitly study the relation between pin density and the local congestion, i.e., how the local congestion can be modeled by pin density in global routing stage to achieve a more accurate routability evaluation.

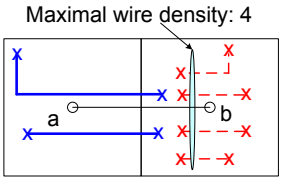


Figure 2.6: Maximal wire density model [4] does not work well for local congestion modeling in global routing.

In [4], an algorithm is proposed to estimate the routing congestion of a circuit considering the local-net resources. It first uses a Steiner tree algorithm to "route" all the nets (including local nets) in the circuit, and then based on the Steiner solution, computes the maximal track usage within a g-cell using a scan-line algorithm. Though the consideration of local connections in a g-cell improves the accuracy of routability evaluation, it has the following problems. Firstly, using Steiner solution for all the nets, including global nets, to estimate the routability can have large errors, since in real routers, significant detours are used for the nets around the congested regions. Secondly, while this method acts more like a congestion estimator (similar to [24, 25]) for a whole circuit, it does not work well in our scenario where the local congestion is to be modeled at the global routing stage. This problem is illustrated in Fig. 2.6. We show two g-cells and assume that the g-edge (a, b) has a capacity of 4 global routing tracks. Using the algorithm in [4], the maximal wire density in the g-cell centered at b would be 4 due to four local nets. However, for global routing, if we reduce the capacity of g-edge (a, b) by 4, then this may be too pessimistic, since there may still be some global nets which can be routed through the g-edge, as shown in Fig. 2.6.

In [33], an interleaved global routing and detailed routing framework, GDRouter, is proposed to improve the detailed routing routability. To improve the consistency in routability evaluation between global routing and detailed routing, three techniques are proposed. First, the cost for each g-cell is calculated to consider pin distribution based on a Voronoi diagram method. Second, the capacity of each g-edge is adjusted based on local routing usage estimated by spine routing which uses a single trunk tree for routing. Third, the capacity of each g-edge is further adjusted by the number of global segments that cannot be assigned to a detailed routing track, which are estimated by performing virtual routing, i.e., fast implementations of FastRoute [10] and RegularRoute [34]. The first technique only uses pin distribution to adjust the cost of g-cells, but

does not adjust the capacity of g-edges to consider the pin-access resources, and thus ignores the fact that pin-access resources would also affect the capacity of g-edges. The second technique could overestimate the local-net resource usage, since the spine routing tree could have more wirelength than a Steiner tree that is used in advanced industrial detailed routers such as [35, 36]. The third technique involves running of fast version of global and detailed routers, which in practice could be computational expensive when applied to a congestion analyzer invoked tens of times in a physical synthesis flow. In summary, these techniques have various limitations when they are used for local resource modeling in congestion analysis.

Some industrial global routers or congestion analyzers also include some methods to model local resources, e.g., some global routers include some form of detailed routing to consider the resources consumed by local net connections and stacked vias [35, 36]. However, these approaches tend to be complex and computationally expensive when such a router acts as a congestion analyzer and is repeatedly invoked during physical synthesis. This is shown in Section 2.6, where we provide runtime data for such an industrial congestion analyzer (different from [35]). The aforementioned problems motivate our work to develop more effective and efficient methods to model local resources when using a global router for congestion analysis.

Next we will present and discuss three methods of local resource modeling: Method 1 based on Steiner tree wirelength estimation, Method 2 based on pin-density estimation, and Method 3 combining techniques from Method 1 and Method 2 and including further enhancements. Before we proceed further to discuss the details of each method, it will be helpful to briefly introduce how we will evaluate the effects of our methods on the accuracy of routability evaluation. Given a design, we first run an industrial congestion analyzer with complex and accurate local resource modeling to get the reference congestion maps. Next, we run another global-routing-based fast congestion analyzer with one of our methods to model local resources on the same design, to obtain another set of congestion maps. Then we use the correlation between the two set of maps to evaluate the effects of our proposed method. Further details will be presented in Section 2.6.1 where we discuss the experimental validation for the proposed methods.

2.3.3 Method 1: Estimation of local resources based on Steiner tree wirelength

In this section, we discuss how to estimate the local resources based on Steiner tree wirelength.

We first discuss the method for local-net resource modeling. We observe that the longer the local wires are, the more likely they are to block global routing tracks. This observation can be

formulated by the following equation:

$$t_b = l_r / s_e, \tag{2.2}$$

where t_b is the number of routing tracks blocked by a local wire, l_r is the length of the local wire, and s_e is the length of a g-edge.

Equation (2.2) is adopted to calculate blocked tracks on a g-edge and can be easily extended to the more complex cases. Consider the case of a multi-pin local net. To estimate local routing, we first build a Rectlinear Steiner Minimum Tree (RSMT) for the pins[3]; in our experiments, we use Flute [37] for this purpose. We then break each horizontal tree segment into two based on the x-coordinate of the g-cell center and apply Eq. (2.2) to calculate blocked global routing tracks on the left and right g-edges associated with the g-cell. Similarly each vertical Steiner tree segment can be broken using the y-coordinate of the g-cell center.

An example is shown in Fig. 2.7, where net (A, B) is a two-pin net while (A, J) and (B, J) are the two segments of a Steiner tree. The global routing tracks blocked by net (A, B) in the horizontal direction can be calculated based on segment (A, J). Since the g-cell center b is between A and J, segment (A, J) blocks global routing tracks on g-edges (a, b) and (b, c). The blocked tracks on g-edge (a, b) can be calculated as $(x_b - x_A)/(x_b - x_a)$, where x_b denotes the x-coordinate of g-cell center b, and other notations are similarly defined. Similarly, the blocked global routing tracks on g-edge (b, c) is $(x_J - x_b)/(x_c - x_b)$. The vertical tracks blocked by net (A, B) can be calculated similarly, based on segment (J, B). As another example, when a segment is completely on the left of (above) or right of (below) the g-cell center, such as the net (C, D) in Fig. 2.7, the blocked tracks can all be attributed, respectively, to the left (top) or right (bottom) g-edge. The blocked tracks on g-edge (b, c), in this case, can be calculated as $(x_D - x_C)/(x_c - x_b)$. The proposed method can be easily applied to more complex Steiner trees, for example A, B, C in Fig. 2.8.

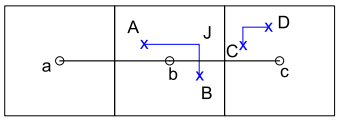


Figure 2.7: Local routing resource estimation for two-pin nets.

[3]We use RSMT since it provides a solution with minimum wirelength for a net, and most modern routers use RSMT as the initial solution for a net.

Next we discuss how we model the pin-access resources. Since most traditional global-routing-based congestion analyzers only produce the g-cell-center-to-gcell-center connections for nets, we must consider the synergy between global and local routing, i.e., how to connect to real pins. As an approximation, the following method is used. We include the g-cell center as a dummy pin when constructing the Steiner tree to model the local resources[4]. For example, for a net (D, E, F, G) shown in Fig. 2.8, the Steiner tree connecting b, E, F, G is used to calculate the blocked global routing tracks on the four boundaries of g-cell with center b. Similarly, the Steiner tree connecting a, D is used to calculate the blocked tracks corresponding to g-cell with center a.

To further consider the track fragments blocked by the local wires connecting to pins as shown in Fig. 2.5(a), we introduce a parameter p ($p > 1$) to scale the estimated local resources using the method discussed above, where p will be tuned empirically for each technology.

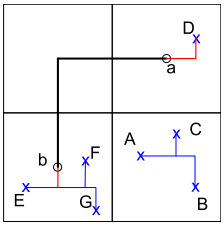


Figure 2.8: Local routing resources consumed by two nets.

In summary, to model local resources in global routing, we add a pre-processing step. Specifically, we iterate through each net in the design, identify the pins inside each g-cell, estimate the local resources using the method presented in this section, and block the global routing tracks from the related g-edges. Local wires inside a g-cell are usually short and for pin accessibility they are typically routed in the second (M2) and third (M3) metal layers during detail routing. Hence, we only block the global routing tracks on g-edges in the M2 and M3 layers during congestion evaluation.

[4]Note that g-cell centers are added as dummy pins only for the nets with global wires. For a local net with all the pins inside a g-cell, the g-cell center is not considered for Steiner tree construction since there is no global wires for this net.

2.3.4 Method 2: Estimation of local resources based on pin density

In this section, we present the second method for local resource modeling, which is even simpler and faster than Method 1, yet is seen to provide similar (or even better) effectiveness. This method is based on pin density, and does not involve constructing Steiner trees to estimate the local resources. It is based on the following observations:

- Each pin is associated with a set of local wires connected to it.
- The number of pins in a g-cell is a good indicator of the number of local wires, and is a first-order estimate for routing tracks blocked by local wires within the g-cell.

Based on the above observations, we model the local resources R_l in a g-cell by

$$R_l = kn, \tag{2.3}$$

where k is a technology-dependent parameter, and n is the number of pins from both local and global nets in the g-cell.

As in Method 1, we use a pre-processing step with Method 2 in a global-routing-based congestion evaluation tool. Specifically, we traverse all the g-cells and nets, and count the number of pins (n), inside each g-cell. Following this, we block kn global routing tracks, due to the local wires in each g-cell, on the four g-edges related to the g-cell. Similar to Method 1, we only block the global routing tracks on g-edges in the M2 and M3 layers.

2.3.5 Method 3: An enhanced method with better scalability

Method 1 and Method 2 provide simple first-order models for local resources, and there is some scope for further improvement. Next, we will first discuss the limitations associated with the two methods, and then present our enhanced method.

2.3.5.1 Limitations in Method 1 and Method 2

As will be shown in Section 2.6.1.2, both Method 1 and Method 2 work well for small g-cell size, but do not scale well to large g-cell sizes. We will analyze the reasons next.

In Method 1, to consider pin access resources, the local resource estimation counts the connection from the real pins to the corresponding g-cell centers, which act as dummy pins. This is based on the assumption that all global wires are connected to g-cell centers. While this seems

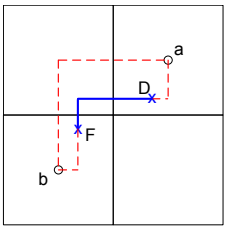


Figure 2.9: Artificially connecting pins to g-cell centers overestimates the routing resources used by two-pin net (D, F).

true in global routing stage, the real situation in detailed routing can be quite different. Fig. 2.9 shows an example, where a two-pin net (D, F) in detailed routing may only consume the routing resources shown as solid blue segments, while by artificially connecting pins to g-cell centers in Method 1, the net will consume resources shown as dashed red segments that significantly overestimate the routing resources required. From the example, we can see that adding g-cell centers as dummy pins to consider pin-access resources can bring errors, which will be even amplified when g-cell size is increased since the wires from pin to g-cell center tend to be longer when g-cell size becomes larger.

There are two major reasons why Method 2 does not scale well with g-cell size. Firstly, Equation (2.3) does not consider the potential relationship between k and g-cell size, and then the k-factor must be tuned for each g-cell size. The dependence of k on the g-cell size is illustrated as follows. Fig. 2.10 shows four g-cells, a, b, c, d, with the size of 10 tracks, and in each there are 10 pins. Assume 10 pins block 5 tracks (showed as red segments) on each edge, and then k for g-cell size 10 is 0.5. Now assume g-cell size increases to 20 tracks, and look at the g-cell F that contains g-cell a, b, c, d and 40 pins. Since the 40 pins now blocked 10 tracks on each edge, k-factor for g-cell size 20 can be calculated as 0.25. This example clearly shows the dependency of k on g-cell size. Secondly, the estimation of local-net resources by pin-density in Method 2 ignores the real wirelength of local nets, and can become inaccurate when g-cell size increases. For example, in Fig. 2.7, two two-pin nets (A, B) and (C, D) have quite different Steiner wirelengths and likely consume different amount of local resources, while Method 2 assumes they consume the same amount of local resources since they both have 2 pins, which may bring error in local resource estimation. This error may be negligible when g-cell size is

small, but could become significant while a large g-cell size is used and the lengths of wires connecting local nets with same pin count can vary in a large range.

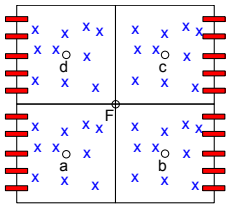


Figure 2.10: When g-cell size doubles, blocked tracks double while the number of pins becomes four times.

In addition to the scalability problem, Method 2 also has other limitations, and can be improved in the following two aspects. Firstly, in Method 2, when adjusting the capacity of four g-edges associated with one g-cell, the amount of blockage added to each g-edge is the same, i.e., kn. This can be further improved by considering the distribution of pins in the g-cell, i.e., if pins are closer to one g-edge, more blockage should be added to that g-edge. Secondly, Method 2 assumes n pins always consume the same local routing resources, kn, but this can also be improved by considering the pin distribution. If the pins are packed with a region smaller than a threshold distance, d_{th}, it will become difficult to access those pins, and more local resources tend to be used for pin access, as illustrated in Fig. 2.5(b). Therefore, an extra weight could be added to scale the local resources used by those pins.

2.3.5.2 The enhanced method to model local resources: Method 3

Motivated by solving the problems in Method 1 and Method 2, we develop an enhanced method to model local resources, Method 3. In Method 3, we use the following strategy to estimate the local routing resources: we spit the estimation of local resources to local-net resources and pin-access resources, and use the suitable techniques to estimate each category. Specifically, we use a modified Steiner tree method to estimate the local-net resources, and an enhanced pin-density method to estimate the pin-access resources with consideration of pin distribution. Finally, we combine the estimation from the two methods as the final estimation. In this way, we make use of the advantages from each method, and avoid the disadvantages.

Firstly, the modified Steiner tree method is used to estimate the local-net resources. Since we do not consider pin-access resources in this step, we do not add g-cell center as dummy pins

when constructing Steiner trees for local nets, and do not scale up the local-net resources. In this way, only the local-net resources are estimated.

Secondly, the enhanced pin-density method is used to estimate the pin-access resources and there are three major improvements. The first improvement is that we change our formulation to make the estimation of the pin-access resources aware of g-cell size. We assume each pin will consume the same amount of routing resources a_r. Different from Method 2, the unit of the routing resources a_r is assumed to be area unit instead of routing tracks, based on the following observation: the routing area consumed by a pin does not depend on g-cell size, while the number of routing tracks consumed by it does. Based on these assumptions, n pins will consume a total of $a_r n$ units of resources. To model these local resources in global routing, we will adjust the capacity of each g-edge in unit of routing track instead of routing area unit. Therefore, we have to do a unit conversion by calculating the equivalent global routing tracks blocked by these local routing resources $a_r n$. Let us assume the resources consumed by each pin a_r to be equal to q unit area a_u, where $a_u = P^2$, where P is the minimum wire pitch in layer M2 (layer M3 usually has the same wire pitch as layer M2). Then we have $a_r n = qP^2 n$. Converting this to the number of global routing tracks by simply dividing the routing area by the area of a routing track, we have:

$$b_p = \frac{qP^2 n}{PS} = \frac{qP}{S} n, \tag{2.4}$$

where S is g-cell size in absolute units. The g-cell size C, in unit of tracks (more commonly used in practice), can be calculated by $C = S/P$. Now combining (2.4), we could calculate the local resources blocked by n pins in unit of global routing track as follows:

$$b_p = \frac{q}{C} n. \tag{2.5}$$

Note that b_p is inversely proportional to C. Equation (2.5) verifies the observations from Fig. 2.10 that the scaling factor associated with pin density n is a function of g-cell size.

The second improvement over Method 2 is that the pin-access resources will be scaled by a weight w_{pa} to be further aware of pin distribution, if the pins are packed within a region smaller than a threshold distance, d_{th}. Given a g-cell c with n pins, when calculating w_{pa}, we only want to consider the pin pair whose distance is smaller than d_{th}, i.e., we assume that the pin pairs with distance larger than d_{th} do not require extra resources. Therefore, we loop through all pin pairs, count the number of pin pairs with distance smaller than d_{th}, denoted as n_{cpp}, and calculate the

mean distance over these pin pairs, denoted as μ_{dp}. Then we calculate w_{pa} as:

$$w_{pa} = \begin{cases} 1 & \text{if } n_{cpp} = 0, \\ 1 + (1 - \mu_{dp}/d_{th}) & \text{otherwise.} \end{cases} \tag{2.6}$$

By (2.6), when there is no pair of pins with distance smaller than d_{th}, w_{pa} equals to 1; otherwise, w_{pa} is a monotonic decreasing function of μ_{dp}. In our implementation, we set d_{th} to the expected distance of the pins in a design, calculated as: $d_{th} = \sqrt{(1 - r_{mp})A_{tot}/n_{totp}}$, where A_{tot} is the chip area, n_{totp} is the total number of pins, and r_{mp} is the ratio of area of the macro blocks to the chip area. Here, the area of macro blocks is subtracted from the calculation, since it is usually impermissible to place pins in such regions.

The third improvement over Method 2 is that the blockage added to each one of the four g-edges associated with a g-cell will be redistributed by considering pin distribution, instead of equal distribution as in Method 2. We will present our method for the horizontal edges, and the case for vertical edges is analogous. For a given g-cell centered at b, as illustrated in Fig. 2.11, denote the two associated horizontal edges as e_l and e_r, and the routing tracks blocked on them as b_l and b_r, respectively. Then we calculate the mean of the x-coordinates of all the pins, denoted as μ_x. Let x_l and x_r be the x-coordinate of the left and right boundary of g-cell b, respectively, and let $S = x_r - x_l$ be the g-cell size. Then we distribute the total local resources $b_p w_{pa}$ (calculated by (2.5) and (2.6)) from n pins in g-cell b by the following formulae:

$$b_l = b_p w_{pa} \frac{x_r - \mu_x}{S} \text{ and } b_r = b_p w_{pa} \frac{\mu_x - x_l}{S}. \tag{2.7}$$

The intuition behind these formulae is that if μ_x is closer to the left horizontal edge, which means more pins are closer to left boundary of the g-cell, we should block more capacity of the left horizontal edge; vice visa.

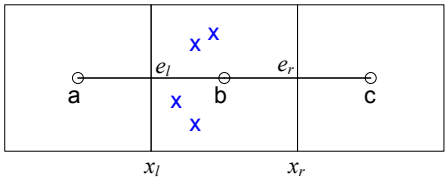


Figure 2.11: The pin-access resources in a g-cell on horizontal layer will be redistributed to the left and right edges based on pin distribution.

Together with all the three improvements, our enhanced pin-access resources algorithm will work in the following steps:

- Count the number of pins in each g-cell, and calculate the pin-access resources b_p by (2.5).
- Calculate the weight w_{pa} for each g-cell by (2.6), and then scale b_p by w_{pa}.
- Distribute the calculated pin-access resources to the four associated g-edges on layer M2/M3 by (2.7).

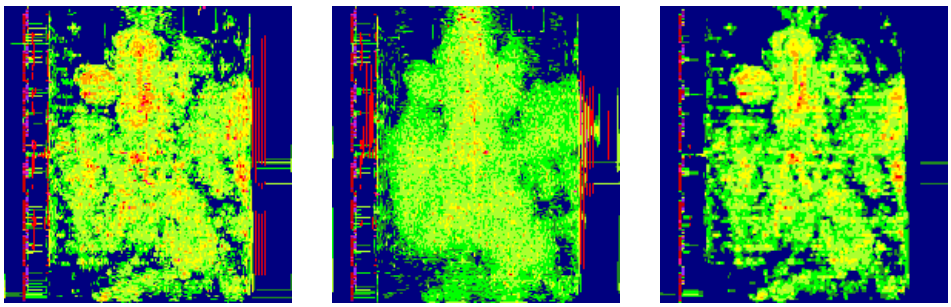
After pin-access resources are computed, the local-net resources will be computed by the improved Steiner method discussed earlier, and finally the blockages from two kinds of local resources will be added to the g-edges in layer M2 and M3. The whole process still works as a pre-processing step, just as in the case of Method 1 or Method 2.

2.4 Filtering out the noise in routability evaluation

As mentioned in Section 2.1.1, the noise hot spots in the congestion maps prevent us obtaining accurate routability evaluation, and should be addressed properly. In this section, we will first present quantitative analysis on this problem, and then propose a smoothing technique to deal with it.

To quantitatively study this problem, we introduce a metric called *noise ratio* to measure the ratio of noise hot spots to the total number of hot spots in a map. If g_{th} is the congestion threshold, then a g-edge e with routing demand $w_e > 0$, and congestion $g_e \geq g_{th}$ will be treated as a hot spot. In this chapter, g_{th} is set to 80% according to the properties of the industrial routing tool set we used. Quantitatively, a hot spot is treated as a noise hot spot when the difference between its congestion and its two parallel adjacent g-edges in the same routing direction is larger than θg_{th}, where θ is a user-defined parameter. With $\theta = 0.25$ (which is the setting used in this chapter), when routing with a typical g-cell size, e.g., 40 tracks, the difference in the routing usage between a noise hot spot and its parallel adjacent g-edges will be larger than $40 \cdot \theta g_{th} = 8$ tracks, which is large enough so that it is likely that more routing efforts could more evenly redistribute the routing usage among these edges. Based on these discussions, the noise ratio can be easily calculated given the congestion data for all the layers. Continuing our analysis on the example shown in Fig. 2.2 (for better readability, we copy the figures to Fig. 2.12), we calculate

the noise ratios for the routing solutions from congestion analysis mode (Fig. 2.12(a)) and global routing mode (Fig. 2.12(b)) of an industrial router. For the congestion analysis mode, the noise ratio is 17.08%, while for the global routing mode, the noise ratio is reduced to 9.98%, which means that the insufficient routing efforts in congestion analysis result in much more noise hot spots than those in global routing.



(a) Map from congestion analysis

(b) Map from global routing

(c) Smoothed map using Gaussian function

Figure 2.12: The proposed smooth technique reduces the noise hot spots in the congestion map.

To reduce the number of the noise hot spots in the congestion maps and achieve more accurate routability evaluation, a smoothing technique can be applied to the congestion maps obtained by a congestion analyzer. Smoothing is a common technique widely used in many fields, such as signal processing, image processing, and CAD. In CAD, smoothing techniques have been widely used in placement algorithms, which use a bell-shaped function [30, 38–42] or the inverse Laplace transform [43] to smooth the unsmooth functions such as density function, so that optimization can proceed more effectively. Additionally, in [42], a Gaussian function was used to further smooth the cell density (or potential) map of the circuit. Smoothing is also used in works related to manufacturability [5, 44–46], where Gaussian function is one of the commonly used smoothing functions to smooth the pattern density.

However, to the best of our knowledge, no application of smoothing techniques directly to the congestion map has been proposed in the literature. In this work, we propose to use a smoothing technique to filter out the noise hot spots in the congestion maps obtained by a congestion analyzer. Unlike [42], we apply a one-dimensional Gaussian function to smooth the congestion map for each layer in the direction perpendicular to the preferred routing direction of that layer.

The reason to use a one-dimensional function is that in routing, to mitigate the congestion, spreading the overflowed wires to neighboring g-edges is directional, i.e., vertical/horizontal spreading on the horizontal/vertical layer. The Gaussian function used is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}, \tag{2.8}$$

where σ is the standard deviation. Since the Gaussian function will be applied on a discrete congestion map, the Gaussian function will be discretized to the GRG using a method similar to those in [44, 45]. We first define the smoothing window size as $(2l + 1)$ g-edges, and the smoothing function will be truncated beyond the smoothing window, which means that when we calculate the smoothed congestion value at one g-edge, other g-edges with distance farther than l will not be counted. Typical values for l can be σ, 2σ and 3σ. After discretization, we should normalize the function values to make sure they sum to 1. Given a congestion map for a layer, if we denote the congestion of g-edge with coordinate (x, y) on the map as $g(x, y)$, then the smoothed congestion, $\bar{g}(x, y)$, can be calculated by

$$\bar{g}(x,y) = \begin{cases} \sum_{j=y-l}^{j=y+l} g(x,j) f(j-y) & \text{for horizontal layers,} \\ \sum_{i=x-l}^{i=x+l} g(i,y) f(i-x) & \text{for vertical layers.} \end{cases} \tag{2.9}$$

Note that when calculating $\bar{g}(x, y)$ by (2.9), whenever the coordinates go over the boundary of the congestion map, e.g., become smaller than 0, the original congestion $g(x, y)$ will be used to fill in the missing $g(x, j)$ or $g(i, y)$. In practice, the smoothing technique can be used multiple times until the map reaches the desired extent of smoothness, specifically, until the noise ratio is reduced below a threshold, e.g., 5.00%.

We now test the smoothing technique on the routing solutions shown in Fig. 2.12(a), assuming $l = \sigma = 1$, i.e., the smoothing window size would be three g-edges. The discretized function will have three values: $f(0) = 0.3989$ and $f(1) = f(-1) = 0.2420$. After normalization, $f(0) = 0.4519$ and $f(1) = f(-1) = 0.2741$. Then applying this smoothing function to the congestion maps of all the layers from congestion analysis mode, we obtain the smoothed combined congestion map shown in Fig. 2.12(c). We can see that as expected, the map in Fig. 2.12(c) is smoother than the original map in Fig. 2.12(a). The noise ratio for the smoothed routing solution is calculated as 1.15%, much smaller than 17.08% for the original solution. Moreover, the difference between smoothed map and global routing map becomes smaller than

that between congestion analysis map and global routing map. This shows that our proposed smoothing technique can indeed filter out the noise hot spots, and improve the accuracy of the routability evaluation.

2.5 Metrics for design congestion

In this section, we will first discuss the limitations of existing routability metrics, and then present our new metric.

2.5.1 Limitations of current metrics

TOF and MOF: Naïve implementations of the TOF and MOF metrics treat the overflow in each layer as identical; however, this is inaccurate as each layer has a different capacity. Normalizing the overflow to the layer capacity can overcome this issue, but other problems remain. The TOF metric does not capture the hot spots in the congestion map, i.e., the severity of congestion in the worst regions of the chip. MOF fares only slightly better, capturing only the maximum overflow value among all the g-edges in the routing graph. This presents a fairly incomplete picture of the congested regions in the design. Moreover, as pointed out in [2], overflow metrics fluctuate greatly, depending on design size, number of g-edges, number of routing layers, etc.
$ACN(20)$, $WCI(100)$ and $WCI(90)$: These metrics fail to differentiate between a net spanning a single congested g-edge and one that spans multiple congested g-edges.

Example 2.1. *Consider two nets in the GRG:* net A *traverses g-edges with congestion* 0.50*,* 0.70*,* 0.80*,* 0.90 *and* 1.10*, while* net B *traverses g-edges with congestion* 0.60*,* 0.80*,* 0.95*,* 1.05 *and* 1.10*. When calculating* $ACN(20)$*,* $WCI(100)$ *and* $WCI(90)$*, both nets will be counted with the same congestion. However, their routability is different: clearly,* net B *is harder to route compared to* net A*, as it traverses more number of g-edges with higher congestion. This fact is not captured by these net-congestion-based metrics.*

Additionally, minor design changes can cause large fluctuations in the $WCI(100)$ and $WCI(90)$ metrics.

Example 2.2. *Assume a design has a g-edge* e*, with* $c_e = 40$*,* $b_e = 0$ *and* $w_e = 39$*. Assume, that we reroute a net to pass through this g-edge (say, to improve timing). Then the congestion of* e *becomes* 100%*, implying that all* 40 *nets crossing* e *now have a congestion of* 100%*. As a result,*

$WCI(100)$ will now report 40 additional congested nets, when in reality we only rerouted a single net. A similar example applies to the $WCI(90)$ metric. Such instability renders these metrics unsuitable for guiding routability optimization.

Although, $ACN(20)$ avoids large swings due to minor design changes, it suffers from the limitation of not accurately capturing design congestion (demonstrated in Section 2.6.3.1).

In addition, existing metrics improperly model the congestion along macro boundaries [2,47], leading to an artificially high reported congestion. Referring to Fig. 2.13, net N routes to a pin on macro block B. Due to the blockage, the congestion of edge e would be rated as being above 90%, but in practice, we find that such nets are easily routable. Including these g-edges with artificially high congestion when calculating the metric introduces unnecessary noise leading to improper estimation of the routability. Note that we only suggest to exclude the edges along macro boundaries when calculating the metric *after* global routing to evaluate the routability, but the high congestion of these edges should *not* be ignored during the global routing process.

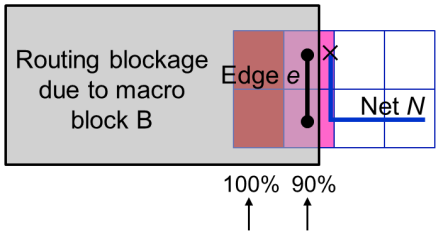


Figure 2.13: An example showing a net N traversing g-cells that are 90% blocked due to a routing blockage. This leads to artificially high reported congestion for g-edge e.

2.5.2 New metric (ACE metric) for design congestion

To address the issues with existing metrics, we propose a new metric that is based on the histogram of g-edge congestion. Our metric has two features:

- It downplays the effects of g-edges with artificially high congestion due to the presence of routing blockages.
- It presents congestion as a histogram, instead of a single number.

To accurately capture the congestion, our metric, denoted as $ACE(x, y)$, computes the average congestion of the top $x\%$ congested g-edges, while ignoring g-edges that are $\geq y\%$

blocked. The role of the parameter y is to void counting the effects of g-edges with artificially high congestion. A typical value for y is 50, implying that all g-edges with $\geq 50\%$ routing blockage are ignored when computing the metric. For convenience, we use $ACE(x)$ to denote $ACE(x, 50)$ in this thesis. For convenience, we also call the metric **ACE metric**.

In practice, the new metric is most useful when expressed as a vector, for different values of x, e.g., for $x \in \{0.5, 1, 2, 5, 10, 20\}$. $ACE(x)$, for a small value of x, (e.g., 0.5 and 1), provides a highly local view, representing congestion in the regions with the highest contention for wiring resources (hot spots). For larger values of x, (e.g., 10 and 20), it provides a broader picture of the design congestion.

2.6 Validation and analysis

Our proposed techniques, hereafter GLARE, are implemented within a congestion analyzer that performs global routing in the spirit of MaizeRouter [48]. This section provides a detailed analysis of GLARE on advanced industrial designs. All experiments were run on a 64-bit Linux server with 4 octa-core CPUs (Intel® Xeon® X7560 2.27 GHz).

We first demonstrate and analyze the impact of methods about local resource modeling on routability evaluation in Section 2.6.1, and then present the results of the smoothing technique in Section 2.6.2, followed by the analysis of the impact of the proposed ACE metric in Section 2.6.3.

2.6.1 Impact of local resource modeling on routability evaluation

For the analyses presented in this section, we use the following engines to evaluate the impact of our proposed techniques for local resource modeling:

- **Analyzer A0**: A fast congestion analyzer that is based on MaizeRouter [48], with the ability to perform global routing on millions of nets in less than 10 minutes[5]. It does not include any local resource modeling.
- **Analyzer A1**: Modification of Analyzer A0, incorporating the Method 1 for local resource modeling.
- **Analyzer A2**: Modification of Analyzer A0, incorporating the Method 2 for local resource modeling.

[5] This is achieved by running on our Linux server with a proper g-cell size on the designs.

- **Analyzer A3**: Modification of Analyzer A0, incorporating the Method 3 for local resource modeling.

- **Reference Analyzer:** A full-blown industrial router that has a mode for performing congestion analysis with complex modeling of local resources. The reference analyzer is used to judge the quality of all results, and typically runs at least 10 times slower than Analyzer A0–A3.

To quantitatively measure the correlation between Analyzer A0–A3 to the Reference Analyzer, the average relative error (AVRE) between the congestion maps of Analyzer Ai, $0 \leq i \leq 3$, and those from the Reference Analyzer, is computed for each design. For each g-edge e on the congestion map, relative error is calculated as

$$e_{re} = \frac{|g_{ye} - g_{xe}|}{g_{xe}},$$

where g_{ye} is the congestion from Analyzer Ai, $0 \leq i \leq 3$, and g_{xe} is that from the Reference Analyzer. Then the AVRE is the average of e_{re} over all the g-edges considered. Since, in practice, only the congestion values close to or over congestion threshold g_{th} are of interest, we use a threshold when calculating AVRE, counting only the g-edges whose congestion is larger than 70% in the Reference Analyzer. Correspondingly, in our color map, any congestion below 70% is colored blue. When using AVRE, a smaller error means better correlation.

2.6.1.1 Improving congestion analysis accuracy

First we will present the impact of our methods about local resources modeling on the overall accuracy of congestion analysis.

Nine designs from three technology nodes are tested, and their chief characteristics are listed in the first three columns in Table 2.1. Since all three methods for modeling local resources involve parameter tuning at a given technology node, the parameters for each method are first tuned[6] on the first design from each technology node shown in the table, and then the tuned parameters are used to test the other two designs in the same technology node. In this set of experiments, g-cell size equal to 20 tracks is used.

[6]In the tuning process, each of the analyzers A1–A3 is applied to the given design with a series of trial values for the parameter to be tuned, and then for all the routing solutions with different parameters, we calculate the AVRE between them and the routing solution from the Reference Analyzer. Finally, we pick the parameter which produces the smallest AVRE value.

Table 2.1: Comparison of accuracy in routability evaluation with and without local resource modeling, with g-cell size 20 tracks.

Technology	Circuits	#nets	AVRE				Tuned parameters		
			A0	A1	A2	A3	A1 (p)	A2(k)	A3(q)
32 nm	ckt_fb	320,357	13.02%	7.17%	7.09%	7.12%	2.20	0.29	7.20
	ckt_dl16	1,191,615	13.51%	7.30%	7.23%	7.43%			
	ckt_dl12	1,337,659	14.36%	9.34%	9.21%	9.29%			
45 nm	ckt_i	354,771	11.21%	7.47%	7.42%	7.48%	1.13	0.15	3.20
	ckt_y	324,102	12.03%	8.59%	8.58%	8.62%			
	ckt_m	118,911	10.95%	7.87%	7.86%	7.94%			
65 nm	ckt_12	1,660,223	10.91%	6.96%	6.90%	6.94%	1.80	0.24	6.00
	ckt_18	533,530	12.12%	8.97%	8.87%	9.06%			
	ckt_x	464,661	8.96%	6.92%	6.81%	6.91%			
Average			11.90%	7.84%	7.77%	7.87%			

Table 2.2: Comparison of accuracy in routability evaluation using different methods of local resource modeling and pre-tuned parameters from Table 2.1, with g-cell size 80 tracks.

Circuits	AVRE			
	A0	A1	A2	A3
ckt_fb	8.83%	20.56%	21.05%	4.80%
ckt_dl16	13.08%	10.96%	13.27%	6.77%
ckt_dl12	10.50%	15.48%	19.11%	6.60%
ckt_i	7.73%	12.18%	21.47%	5.62%
ckt_y	9.50%	9.76%	17.66%	6.39%
ckt_m	8.74%	10.97%	20.37%	5.87%
ckt_12	9.55%	10.38%	16.28%	5.60%
ckt_18	9.98%	5.90%	10.98%	6.37%
ckt_x	6.23%	5.72%	9.40%	4.17%
Average	9.35%	11.32%	16.62%	5.80%

In Table 2.1, the column "tuned parameters" lists the parameters tuned for the three methods of local resource modeling. The column "AVRE" presents the comparison of the errors between Analyzers Ai, $0 \leq i \leq 3$, and the Reference Analyzer, and the row "Average" shows the average error over all the designs. From these data, we can see that analyzers with the three proposed methods for local resource modeling can achieve more accurate routability evaluation than Analyzer A0 without any local resource modeling, which is also true for the designs which the parameters are not tuned for, demonstrating the effectiveness of the proposed local resource

modeling methods. Furthermore, the three methods produce very similar accuracy on average, though Method 2 is a little better than the other two methods.

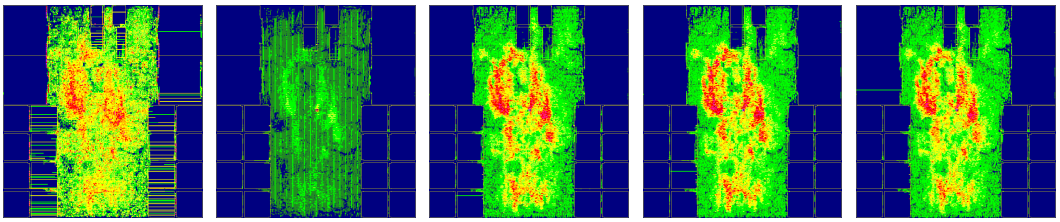
To visually see the impact of local resource modeling on accuracy of routability evaluation, in Fig. 2.14, we show the results of running all the five analyzers on design ckt_fb. From Fig. 2.14(b) we see that using a congestion analyzer with no modeling of local routing resources significantly underestimates the actual congestion. Alternatively, the congestion maps from the GLARE-based congestion analyzers (A1–A3) (Fig. 2.14(c)–2.14(e)) are much closer to the one obtained from the Reference Analyzer[7], both in terms of the congested regions and their intensity. This result assumes significance in the context of using analyzers within congestion mitigation tools such as CRISP [16], where the effectiveness of the tool is highly dependent on accurately identifying the regions of high congestion as well as their relative intensity.

2.6.1.2 Scalability of the proposed methods on increasing g-cell size

We have just shown that the three proposed methods for local resource modeling all work similarly well on the nine designs when the g-cell size is set to 20. Next, we will check their scalability on increasing g-cell size.

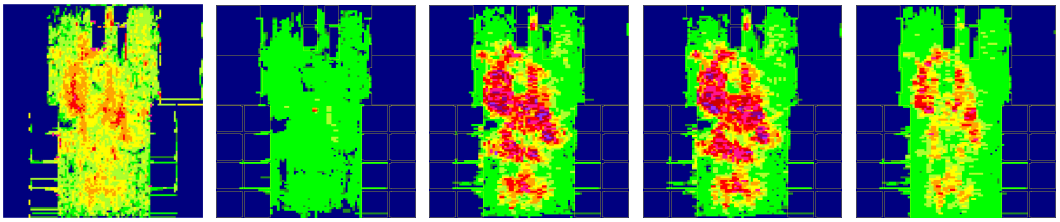
We rerun the experiments on all the nine designs using the same tuned parameters but a g-cell size of 80 tracks. In Table 2.2, the column "AVRE" presents the comparison of the errors between Analyzers Ai, $0 \leq i \leq 3$, to the Reference Analyzer, and the row "Average" shows the average error over all the designs. Comparing the error data in Table 2.2 with those in Table 2.1, we can see that different analyzers show different scalability. When using the same parameters tuned with g-cell size 20 tracks, Analyzer A2, integrating the simple pin-density-based method (Method 2) for local resource modeling, has the largest error among all the analyzers, and the error is even larger than Analyzer A0 without any local resource modeling. Analyzer A1, integrating the Steiner-tree-based method (Method 1) for local resource modeling, has smaller errors than Analyzer A2, but still on average larger than Analyzer A0. These results show the bad scalability in Method 1 and Method 2, which is caused by the factors discussed in Section 2.3.5.1. In contrast, Analyzer A3, integrating the enhanced method combining the Steiner tree technique and the pin-density technique with pin distribution (Method 3), achieves the best scalability and

[7] It is expected that the GLARE-based congestion analyzers do not exactly match the maps from Reference Analyzer, since they run much faster, and do not work as hard as the Reference Analyzer. However, the GLARE-based congestion analyzers can generally predict the hot spots well.



(a) Reference Analyzer (b) Analyzer A0 (c) Analyzer A1 (d) Analyzer A2 (e) Analyzer A3

Figure 2.14: Congestion maps for ckt_fb with five analyzers using a g-cell size of 20 tracks.



(a) Reference Analyzer (b) Analyzer A0 (c) Analyzer A1 (d) Analyzer A2 (e) Analyzer A3

Figure 2.15: Congestion maps for ckt_fb with five analyzers using a g-cell size of 80 tracks.

the smallest error on average among all the four congestion analyzers A0–A3. This shows the effectiveness of the proposed techniques in Method 3.

Fig. 2.15 shows the congestion maps from all the analyzers on design ckt_fb with the g-cell size set to 80 tracks. Analyzer A0 without local resource modeling still significantly underestimates the actual congestion. Analyzer A1 and A2 both generate over-pessimistic congestion maps as compared with the Reference Analyzer. The congestion map from Analyzer A3 is closest to that from Reference Analyzer among all analyzers, demonstrating the good scalability of the proposed enhanced method (Method 3).

2.6.1.3 Runtime and acceleration by using a larger g-cell

Now we will present the analysis on the runtime of different methods for local congestion modeling and different analyzers, with g-cell sizes of 20 and 80 tracks. We will show that the proposed local resource modeling methods only take a small portion of the runtime of the

whole congestion analysis, and with good local resource modeling, congestion analysis can be accelerated by using a larger g-cell size.

In Table 2.3, the column "LRM (size 20)" lists the CPU time of the pre-processing step for local resource modeling (LRM) in A1–A3 with a g-cell size of 20 tracks, "Total runtime" shows the CPU time for Analyzer A0–A3 ("size 20": g-cell size 20 tracks, "size 80": 80 tracks), and "RA20" presents the CPU time of Reference Analyzer with a g-cell size of 20 tracks. In the row "ratio", the numbers in the column "LRM (size 20)" list the ratio of LRM CPU time to the total runtime of the corresponding analyzer, and the other numbers show the CPU time normalized by that of Analyzer A3 (g-cell size 20), averaged over all the designs.

Table 2.3: Runtime comparison for congestion analyzers with different g-cell sizes.

Circuits	LRM (size 20) (s)			Total runtime (size 20) (s)				RA20 (s)	Total runtime (size 80) (s)			
	A1	A2	A3	A0	A1	A2	A3		A0	A1	A2	A3
ckt_fb	3.7	2.7	6.3	41.6	74.3	81.5	61.0	1499.1	18.4	28.0	25.7	31.2
ckt_dl16	14.7	9.4	22.9	801.3	1384.4	1251.6	1376.4	129927.5	125.7	206.4	188.7	227.6
ckt_dl12	17.2	12.2	26.6	610.7	1036.5	889.8	994.8	29614.2	110.1	176.9	175.6	193.5
ckt_i	3.4	2.2	4.4	45.2	51.0	68.4	70.6	1564.1	21.1	31.1	29.0	33.3
ckt_y	3.5	2.1	5.4	55.8	73.4	70.2	75.9	2790.4	18.7	28.6	29.4	33.9
ckt_m	1.3	0.9	1.8	13.6	22.7	19.1	24.9	503.8	7.0	8.9	8.9	10.1
ckt_12	18.7	15.5	32.5	428.8	653.2	612.9	645.7	41425.5	135.0	216.4	200.7	229.8
ckt_18	6.6	4.4	10.0	281.7	459.7	418.6	456.7	8354.4	53.3	88.9	80.3	90.4
ckt_x	5.5	4.3	8.5	135.4	230.9	209.1	222.2	4984.0	33.4	58.4	57.6	59.1
Ratio	3.5%	2.5%	5.1%	0.63	0.99	0.96	1.00	36.98	0.20	0.31	0.29	0.34

Let us first look at the runtime of different LRM methods. From column "LRM (size 20)", we can see that LRM with Method 2 (sub-column "A2"), the pin-density method, runs fastest among all the three methods, due to it simplicity. Method 3 (sub-column "A3"), the enhanced method, is the slowest, but we find it is still fast enough in practice, taking up only 5% of the total routing time.

Now look at the runtime of different analyzers with g-cell size 20. Among Analyzers A0–A3, Analyzer A0 runs fastest since it does not have any local resource modeling, and sees least congestion, translating to fewest routing efforts. The runtime of Analyzer A1–A3 with LRM is quite similar and a little longer than that of Analyzer A0, but much shorter than the Reference Analyzer[8].

[8]Note that Reference Analyzer is using multi-threads routing, and the numbers listed for Reference Analyzer are total CPU time of all the threads, while the wall time can be various depending on the number of threads used.

Considering accuracy, scalability and runtime, Method 3 is the winner among all the three methods of local resource modeling, and we use Method 3 within GLARE for all subsequent analyses in this chapter.

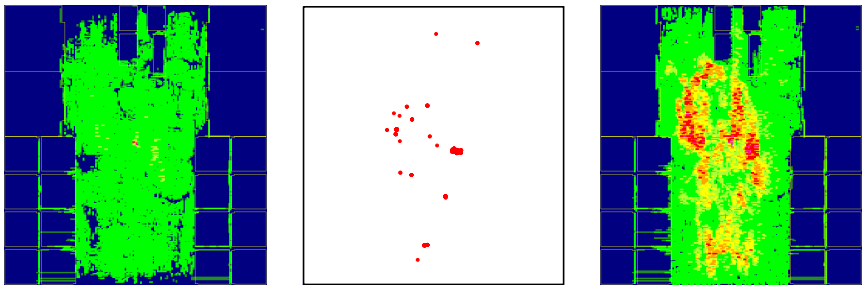
Since our method can accurately incorporate the effects of local routing within a g-cell, it provides the freedom to increase the size of the g-cell, thereby accelerating congestion analysis. As shown in Table 2.3, with Method 3, when increasing g-cell size from 20 to 80, runtime is reduced by 66% on average.

2.6.1.4 Better prediction of detailed routing issues

Often a design that seems routable after global routing can end up with multiple opens/shorts at the end of detailed routing. Early prediction of such issues without performing the time-consuming step of detailed routing is highly beneficial as it enables designers to take appropriate measures, thereby improving overall turn-around time for design closure. As mentioned in Section 2.1, ignoring local resources in global routing can significantly mispredict design routability and cannot predict the opens/shorts locations, as illustrated in Fig. 2.1. These opens/shorts indicate the problematic locations in detailed routing, which, in our experience, are usually due to high local congestion at these locations. Next, we will demonstrate that our proposed local resource modeling method can enable the congestion analyzer to predict detailed routing opens/shorts with high fidelity. Fig. 2.16 shows the comparison of the opens/shorts plots (during an intermediate stage of an industrial strength detailed router) and the congestion maps from Analyzer A0 and Analyzer A3 on design ckt_fb[9]. Comparing these plots, we see that the GLARE-based Analyzer A3 clearly indicates congestion hot spots, which translate to the problematic regions for detailed routing — a fact not captured by Analyzer A0.

To quantitatively measure the predictability of the analyzers, we compute the ratio of the number of opens/shorts present in g-cells with congestion greater than 85% to the total number of opens/shorts in the design. We call this ratio as *match ratio*, and it measures the extent of matching between opens/shorts and the highly congested regions of the design. Here, using 85% as the threshold to consider g-cells in the computation is based on our prior experience that regions with such high global congestion are usually problematic for detailed routing. By this method, the match ratios for Analyzer A0 and the GLARE-based Analyzer A3 are computed

[9]For this experiment, a g-cell size of 40 tracks is used, and Fig. 2.16(a) and Fig. 2.16(b) are copied from Fig. 2.1 for convenience of comparison.



(a) Congestion map from Analyzer A0

(b) Open/short map from detailed routing.

(c) Congestion map from Analyzer A3

Figure 2.16: Open/short maps and congestion plots for ckt_fb. The GLARE-based congestion Analyzer A3 could predict the problematic regions in detailed routing with higher fidelity.

as 0 and 0.97, respectively. In other words, Analyzer A3 was able to point to the congested regions which capture 97% of all opens/shorts, while A0 did not capture any. This further demonstrates the effectiveness of the GLARE-based congestion analyzer in predicting detailed routing opens/shorts.

2.6.2 Impact of the smoothing technique on routability evaluation

As for the proposed smoothing technique, we have showed some results in Section 2.4 on a motivating example, and now we present more results on several industrial circuits in this section. In our experiments, we use $l = \sigma = 1$.

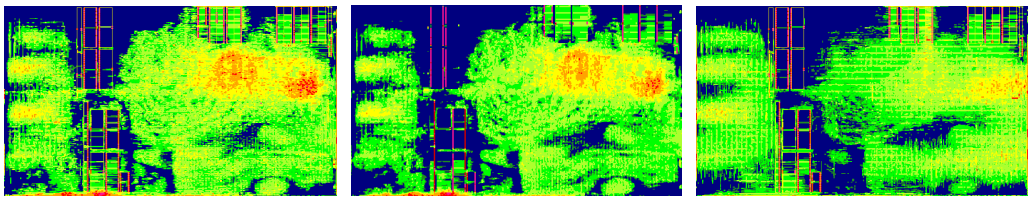
Table 2.4 shows the effects of applying the smoothing technique once to the solutions from a congestion analyzer on several designs. In the table, column "C.A." lists the results of an industrial congestion analyzer, column "G.R." the results of an industrial global router and column "Smooth" the results after applying the proposed smoothing technique to "C.A." results. It can be seen that applying the proposed smoothing technique once would reduce the noise ratio from more than 6% down to below 1%, and the ACE metrics after smoothing become more accurate than those from the congestion analyzer's maps, when compared with those from the global router, which demonstrates the effectiveness of the smoothing technique.

Fig. 2.17 shows the congestion maps for congestion analysis solution, smoothed solution, global routing solution on design ckt_x. It can be seen that the congestion map from the congestion analyzer has many more hot spots than that from the global router, and there are

Table 2.4: Smoothing reduces noise hot spots and improves the accuracy of routability evaluation by a congestion analyzer.

Circuits	Noise ratio (%)			ACE metrics (0.5, 1, 2, 5) (%)		
	C.A.	Smooth	G.R.	C.A.	Smooth	G.R.
ckt_fb	16.39	0.62	8.64	(92.9, 91.5, 89.5, 86.6)	(89.9, 88.5, 87.0, 84.7)	(88.9, 88.0, 86.9, 84.4)
ckt_y	13.99	0.38	6.24	(100.0, 97.7, 95.7, 92.9)	(97.8, 96.1, 94.3, 92.0)	(94.4, 92.1, 90.0, 87.0)
ckt_x	6.34	0.19	2.86	(94.1, 92.5, 91.2, 88.5)	(93.2, 91.8, 90.2, 87.7)	(88.7, 87.2, 86.0, 84.0)

obvious noise hot spots on the map. After smoothing, the map becomes smoother with fewer noise hot spots than the original map, and the intensity of congestion is reduced by some extent, which makes the calculated ACE metrics closer to those for global routing solutions. Note that the congestion map after smoothing do not match well with that from the global router, because routing efforts in the congestion analyzer is much fewer than those in the global router, which cannot be compensated by only using the smoothing technique.



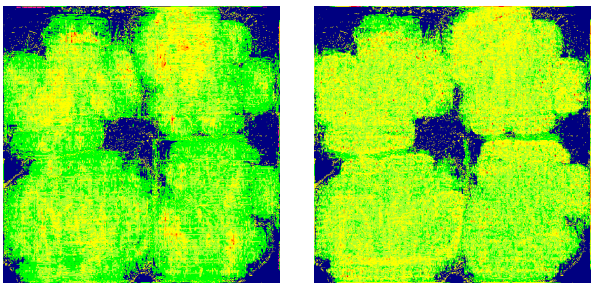
(a) Congestion analysis. (b) After smoothing. (c) Global routing.

Figure 2.17: Congestion maps on design ckt_x.

2.6.3 Impact of ACE metric on routability evaluation

2.6.3.1 Comparison of routability metrics

Visual inspection of a congestion plot is widely used to quickly evaluate the routability of a design point. We now demonstrate that our new metric can capture a congestion plot with higher fidelity compared to prior metrics for routability evaluation. Consider Fig. 2.18 displaying the congestion plots from two global routing solutions on an identical placement for the design ckt_s, which is a 45 nm design with 1,006,029 nets. The corresponding values of the different congestion metrics for the routing solutions are given in Table 2.5. For ACE metric, the congestion is expressed as an ordered pair representing (Horizontal, Vertical) layer congestion.



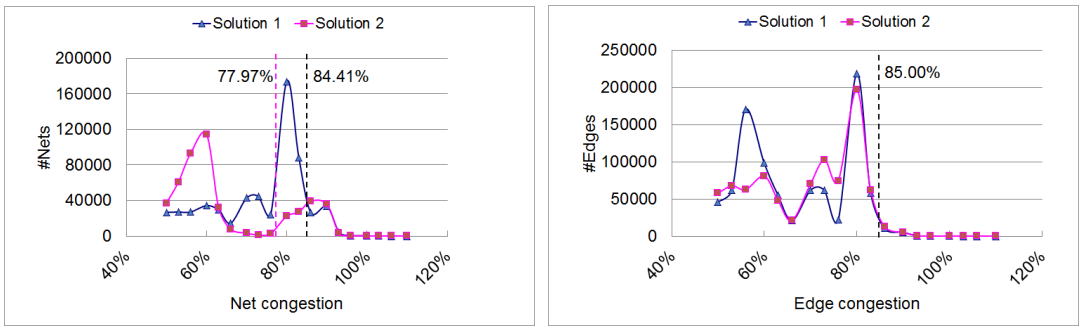
(a) Solution 1 (b) Solution 2

Figure 2.18: Congestion plots for two routing solutions on design ckt_s.

Table 2.5: Congestion metrics for two routing solutions on design ckt_s.

Metrics		**Solution 1**	**Solution 2**
Overflow based metrics	TOF	194373	217499
	MOF	10	11
Net congestion based metrics	$ACN(20)$	84.41	77.97
	$WCI(90)$	39494	40548
	$WCI(100)$	274	276
New metric	$ACE(0.5)$	(90.47, 90.54)	(90.27, 90.46)
	$ACE(1)$	(89.23, 89.12)	(89.13, 89.10)
	$ACE(5)$	(85.93, 85.45)	(86.14, 85.49)
	$ACE(10)$	(84.16, 83.39)	(84.59, 83.51)
	$ACE(20)$	(82.48, 81.58)	(82.57, 81.53)

From Table 2.5, the overflow-based metrics[10] indicate that Solution 1 has better congestion, while the net-congestion-based metrics indicate that Solution 2 is better, as $ACN(20)$ of Solution 2 is much better than that of Solution 1, even though $WCI(90)$ and $WCI(100)$ are worse. However, a visual examination of the congestion plots indicates that they are quite similar, demonstrating the deficiencies in the existing metrics. Alternatively, our new metric correctly identifies the congestion of these two routing solutions to be similar.



(a) Distribution of net congestion.

(b) Distribution of g-edge congestion.

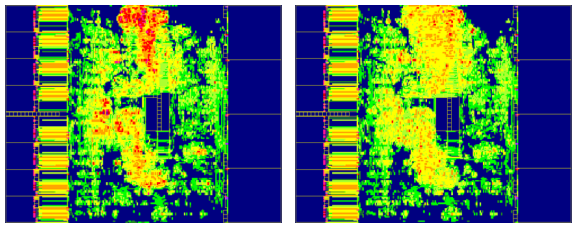
Figure 2.19: Distribution of congestion for two routing solutions of ckt_s.

The significant difference in the $ACN(20)$ values for the two comparable routing solutions can be explained by Fig. 2.19(a) which plots the distribution of the worst congestion on the nets. From Fig. 2.19(a), Solution 1 has a considerably higher number of nets in the $[77.97\%, 84.41\%]$ congestion range compared to Solution 2, leading to the difference in the $ACN(20)$ values. In practice, our experience on industry designs is that nets with congestion less than 85% are often not difficult to route, and considering them within the congestion metric introduces unnecessary noise during routability evaluation. In contrast, looking at Fig. 2.19(b) which plots the distribution of congestion on the g-edges, we observe the distributions for the two routing solutions to be quite similar above 85.00% (even similar above 80.00%). This explains why the new metric (correctly) rates the two solutions to have similar congestion.

[10]To counteract the drawbacks of overflow metrics discussed earlier, when we calculate overflow in this chapter, the capacity is scaled down to 80% of the original, and the overflow is in unit of number of minimum-width tracks, e.g., one overflowed track on a layer with $4\times$ width tracks would be counted as four in the overflow number.

2.6.3.2 Further comparison between overflow metrics and ACE metrics

Next we will use the ultimate routability metric obtained from running an industrial detailed router as the judge to further compare the overflow metrics and ACE metrics.



(a) Placement A. (b) Placement B.

Figure 2.20: Congestion maps from running the same congestion analyzer on two different placements on design ckt_y.

Table 2.6: Routing metrics for two placements A and B on design ckt_y.

Metrics	Detailed routing		Overflow		ACE metric (%)					
	#errors	Time (h)	TOF	MOF	ACE(0.5)	ACE(1)	ACE(2)	ACE(5)	ACE(10)	ACE(20)
A	2,424	49.5	829,164	18	96.4	95.3	93.6	91.0	87.0	79.5
B	816	29.0	829,261	18	94.7	93.5	92.2	90.3	87.2	80.9

Fig. 2.20 shows the congestion maps obtained by running the same congestion analyzer on two different placements for the same design ckt_y. From the congestion plots, we can see the congestion hot spots in placement B is more spread than placement A, and placement B is very likely to have better routability than placement A. Next, we run detailed routing on the two placements using an industrial detailed router, and report the results in Table 2.6. We find that although neither of the two placements are routable, placement B is better than placement A since both the number of routing errors and runtime (wall-clock time reported) on placement B are much better than those on placement A. This verifies our visual observation.

We will now examine the metrics calculated on the routing solutions for the two placements from an industrial congestion analyzer, as shown in Table 2.6. The overflow metrics for the two placements are almost the same, and cannot effectively differentiate which placement is better, again demonstrating the previously pointed out drawbacks of overflow metrics. From the data for ACE metrics (only the maximum between horizontal and vertical layers is reported), we can see that $ACE(0.5)$ and $ACE(1)$ for placement A are higher than that for placement B, implying

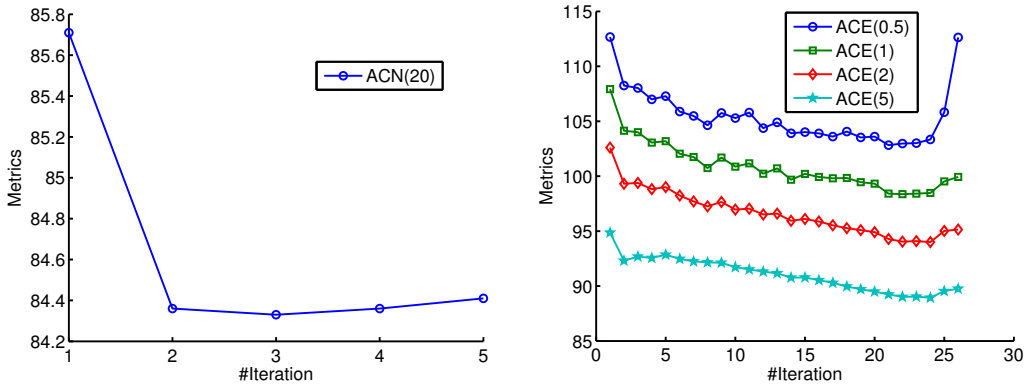
that the hot spots in congestion map for placement A have higher congestion than those in congestion map for placement B. Metrics $ACE(2)$ and $ACE(5)$ for placement A are also worse than placement B. These ACE metrics correctly indicate that placement B has better routability than A, which is consistent with the results of detailed routing. However, the $ACE(10)$ and $ACE(20)$ metrics for placement A is even slightly better than B, and this does not match the fact that placement B has better routability. This can be explained by observing that when we go too deep in the histogram of g-edge congestion while calculating the ACE metrics, the g-edges with smaller congestion are also counted in, which brings noise to the routability evaluation. Therefore, a conclusion drawn from this experiment is that we should not use too many values of x in $ACE(x)$ when using ACE metrics to evaluate routability, i.e., the values $x = 0.5, 1, 2, 5$ are suggested.

2.6.3.3 Guiding routability optimization

We will now discuss the performance of different metrics in guiding routability optimization, and demonstrate that a good congestion metric is essential to effectively guide routability optimization.

In our experiments, we employ CRISP [16], an incremental placer that iteratively spreads logic cells in congested regions to improve the routability of a design. To track its progress, CRISP relies on a congestion metric to compare the routability of placements over successive iterations. It terminates if the metric does not improve for two consecutive iterations. In other words, it returns the placement solution from iteration i if the congestion metrics for both iterations $i + 1$ and $i + 2$ are worse than iteration i.

We first employ the $ACN(20)$ congestion metric to guide CRISP and check the stopping criterion as outlined above. A plot of the progression of the $ACN(20)$ metric across CRISP iterations is shown in Fig. 2.21(a). We see that CRISP terminates after just five iterations and returns the placement solution from the third iteration. The reason for the early termination of CRISP is as follows: CRISP spreads cells in the most congested regions of the design. This may reduce the number of nets in regions with 100% congestion ($WCI(100)$), but increase the number of nets in regions with over 90% congestion ($WCI(90)$). As a result, the $ACN(20)$ metric may not change, or in fact degrade (Fig. 2.21(a), iterations four and five). Such a scenario can cause CRISP to exit prematurely without resolving all the congestion issues.



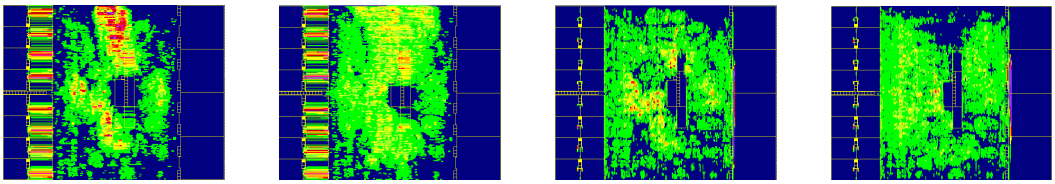
(a) CRISP terminates after five iterations when using the $ACN(20)$ metric.

(b) CRISP iterates longer when using the new metrics for congestion evaluation.

Figure 2.21: The new metrics provide a more accurate view of the congestion, enabling CRISP to be more effective.

Next, we replace the $ACN(20)$ metric with a set of values obtained from our new metric (ACE). As mentioned before, ACE metric expresses congestion as an order pair of (Horizontal, Vertical) layer congestion values. We use the maximum of these two values to check the stopping criterion for CRISP. In this experiment, we use $ACE(0.5)$, $ACE(1)$, $ACE(2)$ and $ACE(5)$ and adopt the following strategy: during every iteration, we determine the four metric values, and consider it as an improvement in congestion if there is a reduction in any one of the metric values. Same as before, CRISP terminates if the congestion evaluated by ACE metric does not improve for two consecutive iterations. The effect of using a set of metric values derived from our new congestion metric is shown in Fig. 2.21(b), wherein CRISP runs for up to 26 iterations.

The intuition behind using a set of edge-congestion-based metric values is as follows: since CRISP spreads cells from the topmost congested g-cells, it should reduce the congestion on the topmost congested g-edges. This is captured by the $ACE(0.5)$ metric value. In doing so it may increase the congestion on other g-edges, but this is still acceptable as it is reducing the peak of the congestion histogram. Over successive CRISP iterations, the $ACE(0.5)$ metric value may saturate, but a reduction in any of the other metric values implies that CRISP is still improving congestion, except that it is now targeting g-edges that are deeper in the congestion histogram.



(a) Post-CRISP congestion plot when using the $ACN(20)$ metric (horizontal direction).

(b) Post-CRISP congestion plot when using the ACE metric (horizontal direction).

(c) Post-CRISP congestion plot when using the $ACN(20)$ metric (vertical direction).

(d) Post-CRISP congestion plot when using the ACE metric (vertical direction).

Figure 2.22: Post-CRISP congestion plots when using different metrics to check the stopping criterion.

Finally, Fig. 2.22 shows the post-CRISP congestion plots[11] when employing the two congestion metrics. There is a significant improvement in the design routability (Fig. 2.22(b) and Fig. 2.22(d)) by running CRISP for more iterations, as enabled by the ACE metric.

2.7 Conclusion

Fast and accurate routability evaluation techniques are critical to address the increasingly important and difficult issue of routing closure in nanometer-scale physical synthesis. In this chapter, we have addressed two important aspects of routability evaluation: the accuracy of congestion estimation and a metric for evaluating the routability of a design. We have shown that ignoring the effects of local congestion can result in large errors during congestion analysis. This observation motivates our models for local resources based on:

1. Method 1: the Steiner tree wirelength of the local nets;
2. Method 2: the pin density in each g-cell;
3. Method 3: an enhanced method combining the good techniques in Method 1 and Method 2, and further considering the scalability on increasing g-cell size and pin distribution.

Experimental results show that the proposed modeling can improve the accuracy and fidelity of congestion analysis, and better predict detailed routing issues such as opens and shorts.

[11] Note that congestion plots in horizontal direction show the maximal congestion among all the horizontal layers. Vertical direction plots are similar.

Especially, the enhanced method has best scalability on g-cell size among the three methods, which enables designers to use large g-cells to accelerate the process of congestion analysis, thereby speeding design closure.

Furthermore, we have discussed the effects of noise hot spots in the congestion maps on routability evaluation, and proposed a smoothing technique, which could be used to obtain more accurate routability evaluation, as demonstrated in our experiments on several industrial circuits.

In addition, we have analyzed the limitations of existing congestion metrics including overflow, etc., and proposed a new metric, ACE metric, based on g-edge congestion. We have demonstrated that ACE metric can represent a congestion plot with higher fidelity. In particular, we use detailed routing data to demonstrate that ACE metric can predict the routability accurately while overflow metrics do not work for the test case used. Finally, we have showed that with ACE metric, a routability-driven placer can perform better and can improve the routability of a design significantly.

Chapter 3

CATALYST: Planning Layer Directives for Effective Design Closure

3.1 Introduction

Physical synthesis is a critical component of modern design methodologies, enabling timing closure at the physical design stage. Technology scaling brings new challenges and opportunities to physical synthesis. Wire resistance per unit length increases quadratically with technology scaling and results in significant increases in wire delay. However, the work [18, 47] shows that the availability of thicker wires in higher metal layers (shown in Fig. 1.1) could potentially relieve this problem, which has also been mentioned in Chapter 1. At 65 nm technology, there are four $1\times$ layers, three $2\times$ layers and two $4\times$ layers (Fig. 3.3). On a $2\times$ layer, the single-width wires are $2\times$ thicker and $2\times$ wider than those on $1\times$ layer, and therefore the per-unit wire resistance is reduced by roughly $4\times$ (the per-unit capacitance is roughly similar across all layers, which is ensured by design rules including wire spacing and process specifications such as inter-layer dielectric thickness), greatly compensating for technology scaling effects. On the $2\times$ [$4\times$] layer, signals can roughly go $1.7\times$ [$2.4\times$] faster, with $2.2\times$ [$4.5\times$] reduction in buffer resources. Therefore, the difference in wire delays in different layers provides another dimension to timing optimization, beyond gate/wire sizing and buffering. Assigning timing-critical nets to thick layers can reduce area/power and improve timing closure by reducing delays and the buffer count. As illustrated in Fig. 3.1, the slack for a two-pin net A on a $4\times$ layers is improved from -10 ps to 10 ps as compared to the corresponding route on the $1\times$ layer, and the number

of buffers is reduced from 7 to 1. Moreover, by using thick layers wisely, it could be shown that a 31% reduction on buffer area averaged over several industrial circuits can be achieved (Sec. 3.6.2). On the other hand, there are limited resources on thicker layers, and if too many nets are assigned to thicker layers, the design may not be routable or have large post-routing timing degradation.

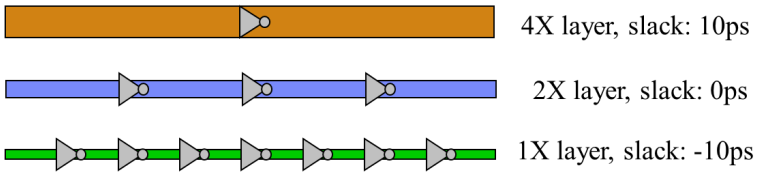


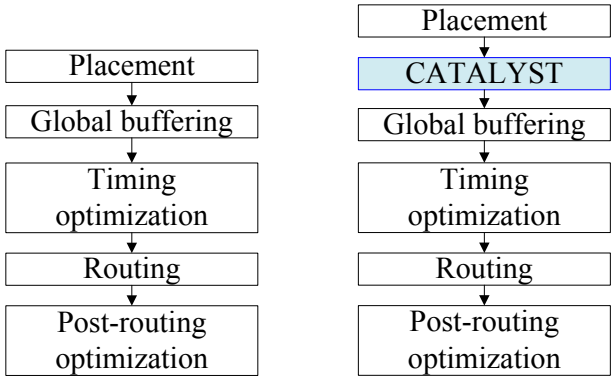
Figure 3.1: Assigning the same net to thicker layers improves timing and buffering.

This extra dimension in timing optimization affects the traditionally predicted trend for interconnect synthesis and buffering, and presents a new problem to the physical synthesis: how to use these thick metal layers wisely. This problem is pervasive in all steps of the design flow, from the early stages to the late stages and relates to the classical tradeoff between flexibility and accuracy: as one progresses deeper into the design flow, timing estimation becomes more accurate but the level of flexibility in changing layer assignments diminishes.

Existing works on layer assignment have focused only on late stages of design, mainly during the routing and buffering stages. Most of the previous related works are from the global and detailed routing literature, but do not address the problem of early planning for layer assignment. Conventional routers perform layer assignment purely for routing congestion minimization and many works focus on how to perform layer assignment with via minimization [10–13, 48, 49]. In these works, the timing benefit of thick layers is not leveraged at all. Subsequent work on timing-driven layer assignment [50–53] has used timing information to drive layer assignment. While it is certainly necessary to consider layer assignment during routing, the timing gain in these works is limited since routing is performed after all optimizations are completed, or at least after a majority of buffers are placed. Recent papers [54–56] focus on how to obey the given layer assignment constraints from the prior synthesis stage in the routing algorithm, but do not discuss the process of generating these constraints.

The work in [18] performs layer assignment during the timing optimization stage and is combined with buffering. It presents two algorithms to perform simultaneous layer assignment

and buffer insertion on a single net given the Steiner topology, and shows significant timing benefits and buffer area savings. However, it has three major limitations. First, it is not aware of the routing congestion. The approach attempts to control the number of nets promoted to thick layers, but its guess and trial approach could still easily cause over-promotion (assigning too many nets to thick layers) in the design, which causes the design to be unroutable [47]. Second, it does not explicitly minimize the buffer usage: it may underuse the thicker layers if the timing can be closed by buffer insertion on thinner layers, with excessive buffers inserted. Third, since the approach is a net-based algorithm, it does not discuss the wrapper around it to be used in a design flow, especially which nets to choose and in what order.



(a) Current flow. (b) Proposed flow.

Figure 3.2: Current and proposed physical synthesis flows.

In this chapter, we propose a novel algorithm, **CATALYST**: **C**ongestion **A**nd **T**iming **A**ware **LaY**er a**S**signmen**T**, to perform layer assignment to maximize the timing benefits of thick metal layers with congestion control at early stages. CATALYST alters a traditional physical synthesis flow [57, 58] (a simplified version is depicted in Fig. 3.2(a)), and is inserted just before global buffering, as shown in Fig. 3.2(b). We believe it is a *catalyst* to enable faster and better design closure if thick metal layers are wisely used earlier. Unlike [18], our algorithm tries to assign a large number of nets to thick layers with the goal to minimize the buffer usage. Moreover, since our method has a global routing engine embedded inside, it has good control of the congestion when performing layer assignment.

Our work has several significant contributions. It presents

- a novel problem formulation for layer assignment at early stages;

- techniques to control congestion during layer assignment;
- techniques to maximize the timing benefits of layer assignment guided by a delay model (to be discussed in Sec. 3.2.3);
- techniques to minimize buffer usage by assigning as many nets as possible to thick metal layers while controlling congestion.

Our algorithm has been tested on several industrial designs across 65 nm, 45 nm, and 32 nm technologies. Compared with another aggressive layer assignment algorithm, CATALYST can achieve similar timing improvements (improving the worst slack by 0.8 ns on average) but avoid high congestion. Moreover, CATALYST has been embedded and tested in an industrial physical synthesis flow (Fig. 3.2(b)). Experimental results demonstrate that CATALYST can save the buffer area by 10% on average and up to 18%, while maintaining the similar congestion, timing, and runtime. These savings could also help to improve the design power and cost, and help to achieve effective design closure.

The remainder of this chapter is organized as follows. We introduce some concepts and notations, routing and timing models, and the problem formulation of layer assignment in Section 3.2. Next, an overview of the CATALYST algorithm is given in Section 3.3, and the detailed discussions of the CATALYST algorithm are presented in Sections 3.4 and 3.5. Experimental results are then reported and analyzed in Section 3.6, followed by a conclusion in Section 3.7.

3.2 Preliminaries

In this section, we discuss the routing and timing models used in CATALYST, the notations used and the problem formulation.

3.2.1 Layer directives and notations

An important concept involved in this chapter is that of the **layer directive** (simply, **directive**): a directive is a constraint on a net which specifies the valid layers the net can be routed on, and is typically given by a pair of layer names. For example, a net n with directive $[M_5, M_9]$ means that net n can only be routed between layer M_5 and M_9. Here, M_i denotes the i^{th} metal layer.

CATALYST differs from the traditional layer assignment step in global routing [10–13,48,49], which assigns the wires to different layers as a last step to complete the routing of a net. In contrast, CATALYST generates layer directive *constraints* for timing-critical nets, and these directives are propagated throughout the physical synthesis flow. We will use the term **layer directive assignment (LDA)**, or simply **directive assignment**, to refer to our layer assignment process.

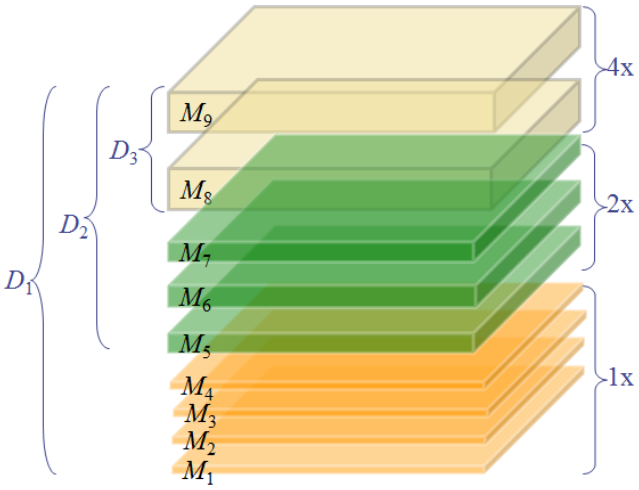


Figure 3.3: A pictorial view of an IBM 65 nm technology.

We now elaborate on the way layer directives are provided, using an example of an IBM 65 nm technology, as illustrated in Fig. 3.3. Let a **plane** be a set of layers with the same thickness. Then this technology consists of three planes: $\{M_i|1 \leq i \leq 4\}$, $\{M_i|5 \leq i \leq 7\}$, and $\{M_i|8 \leq i \leq 9\}$. Correspondingly, three layer directives can be derived: $[M_1, M_9]$ as D_1, $[M_5, M_9]$ as D_2 and $[M_8, M_9]$ as D_3. From the example, we can see that for j^{th} directive D_j, the bottom layer will be the bottom layer in j^{th} plane, and the top layer will be always the top metal layer. Note that D_1 imposes no constraint, and is not used in practice.

It is useful to assign the most timing-critical nets to D_3, so that these nets can obtain best delay gains from the thickest layers. If assigning a timing-critical net n to D_3 results in congestion bottlenecks, we could give it greater flexibility by assigning it to D_2, allowing n to be routed between M_5 and M_9. In such a case, we use the parasitics in $2\times$ plane when computing the wire delay and gate delay associated with net n, since industrial routers tend to route nets in the lowest allowable metal layers, which is consistent with the objective of via minimization. If

there are no resources in the lowest layers, a wire may be routed in a higher layer, and the timing computation based on the parasitics of the $2\times$ plane[1] is guaranteed to be pessimistic.

We further introduce some notations. Let M be the number of planes in a design. Correspondingly, there will be M layer directives D_j, $1 \leq j \leq M$. For simplicity, we overload the notation to also use directive D_j to denote the set of layers specified by D_j. We refer to D_1 as the *bottom* directive, and D_M the *top* directive. The *promotion* of a net will refer to the case where we assign a net from D_j to a directive with a higher index (corresponding to higher metal layers); similarly, the *demotion* of a net will refer to the opposite. Since the parasitics of the lowest metal layer in a directive will be used to evaluate the delay of the nets in that directive, promoting (demoting) a net will improve (worsen) its estimated delay.

3.2.2 Global routing with layer directives

As discussed in Section 2.2, in global routing, a GRG with g-cells and g-edges is constructed, as shown in Fig. 2.3. In the presence of multiple layers, the 2D GRG becomes a 3D GRG, with a 2D GRG representing each layer, and the z-direction g-edges representing vias that connect the 2D GRG's.

Most of traditional global routers adopt a three-stage routing scheme: plane projection, 2D routing and layer assignment. However, routing with layer directives is quite different. Several methods have been proposed to address layer directives in global routing [54–56]. We use the progressive projection method in [56] due to the flexibility it affords. In this method, the nets are first partitioned to several sets according to their directives. Let $\mathcal{P}_k$ be the set of nets with directive D_k, where $1 \leq k \leq M$. A set of routing subproblems I_k are constructed with a set of nets $\mathcal{P}_k$ and a set of layers in D_k. Next, the subproblems are solved one by one. Since the nets in higher directive have stronger limitations (with fewer available layers), subproblem I_k with larger k will be solved first. In solving I_k, the 2D GRG is constructed in the following way: the capacity is aggregated by only using the layers in D_k, and the routing solution from the previous subproblem I_{k+1} (if available) will be treated as existing wires in the 2D GRG. In summary, the whole routing process will have M passes: in each pass, 2D-routing and then layer assignment will be performed on the 2D GRG constructed using the method discussed, and finally an accumulated 3D solution will be output.

[1] The parasitics of lowest layer and highest layer specified by the layer directive may be used as a $[\min, \max]$ range in timing analysis to get more information if necessary.

The traditional goals of global routing are to minimize the wirelength, congestion and via count. Congestion can be evaluated by different metrics such as overflow defined in (2.1) or the new ACE metric which was proposed in Chapter 2, and has been shown to be more effective than conventional overflow-based metrics. Recall that the ACE metric computes the average congestion of the top $x\%$ congested g-edges, denoted as $ACE(x)$. Furthermore, a derived metric, peak weighted congestion (PWC), was adopted in DAC 2012 contest [26], given by $(ACE(0.5)+ACE(1)+ACE(2)+ACE(5))/4$, which will be used to evaluate the congestion in the work presented in this chapter.

3.2.3 Timing metrics and model

In this chapter, we use the following timing metrics: the worst slack (WSLK), the figure of merit (FOM) which is sum of the gap in slack to a target (s_{tar}) for all timing endpoints with slack below s_{tar}, and the number of timing endpoints with negative slack, n_{neg}. Here, we set s_{tar} to 0 ps, so that FOM is effectively the total negative slack.

As pointed out in [57], for appropriate tradeoffs between runtime and optimization accuracy, timing models with different levels of accuracy are used at different stages in a typical physical synthesis flow (Fig. 3.2(a)). Before placement, an optimistic zero-wire-load model is typically used during logic synthesis. Subsequent to placement, one could switch to a traditional RICE model [59]. However, such a timing analysis will result in huge numbers of critical paths because buffering has not yet been performed, and it becomes impossible to distinguish real critical paths from paths that simply lack buffers. Long wires without buffers will have quadratic delays and make the timing appear much worse than it potentially will be after timing optimization. Hence, to avoid this problem, a *linear delay model* [60–62] can be used for wires, where the interconnect delay is estimated to be linear with the wirelength assuming optimal buffering. It allows a reasonable estimate of post-placement timing, and allows the buffering to be deferred until layer assignment has been performed.

In this work, CATALYST is performed before buffering (Fig 3.2(b)), and therefore, we use a linear delay model similar to that used in [62][2].

[2]Our enhanced implementation also considers the effects of vias on the wire delay.

3.2.4 Layer directive assignment

The LDA problem can be stated as follows: given a design, assign directives to nets with the goal of satisfying the timing and congestion constraints, and minimizing the number of buffers required.

To evaluate the effects of assigned layer directives on routing congestion, a global router with ability to obey the layer directives, such as [54–56], can be used to evaluate the congestion. We should try to keep the congestion with newly generated directives similar to that without any directives. Precisely, the congestion constraint to the LDA problem can be $g_{LDA} \leq \beta g_{org}$, where g_{LDA} and g_{org} are calculated by a congestion metric of the routing solutions with and without assigned directives, respectively, and β is a user-defined parameter.

3.3 Overview of CATALYST

In this section, we discuss how we solve LDA problem and overview the CATALYST algorithm. We solve the LDA problem by decomposing it into two subproblems:
Subproblem 1: Timing-driven directive assignment.
Subproblem 2: Congestion- and timing-aware directive assignment.
The goal of the Subproblem 1 is to generate initial layer directive assignment solution to meet the timing constraints, while minimizing the number of nets with directive assigned. Subproblem 2 has two goals. First, it tries to keep the initial directive assignment from the solution of Subproblem 1 within the allowable congestion range. Second, it tries to assign as many nets as possible to higher directives to further improve timing and reduce potential buffer usage.

To solve Subproblem 1, we propose a simple but effective timing-driven directive assignment heuristic by promoting the timing-critical nets to higher directives one by one with incremental timing updates. To solve Subproblem 2, our tool embeds a global routing engine inside to control the congestion. First it will examine the directives obtained from the first step, and relax the constraints/directives if they cause congestion. Next, it will try to perform directive assignment on the rest of nets to further improve timing and reduce potential buffer usage. It then calls the timer to update the timing at the end to capture the effects of the nets touched during this step. Simply speaking, the first step focuses on timing improvement, and the second step focuses on congestion control and buffer usage improvement.

The CATALYST algorithm is an iterative process. After every solution of Subproblem 2, we rerun timing analysis based on the new directive assignments, which change the timing, and then start a new iteration. The iterations continue until the stopping criterion is satisfied: either WSLK or FOM becomes worse, or the improvement on both of them is smaller than a threshold θ, or the user-defined maximal number of iterations n_{itr} is reached.

3.4 Timing-driven directive assignment

Subproblem 1 can be formulated as follows. Given a circuit, to compute a directive assignment solution to maximize the timing improvement and to minimize the total cost. Depending on the purpose, one can model this cost as congestion, to avoid unroutable regions, or a directive cost by assigning a higher cost to a higher directive to minimize the usage of higher layers. Our work uses a directive cost to minimize the directives assigned in solving Subproblem 1, as will be explained later in this section.

In this subproblem, the circuit can be modeled by a directed acyclic graph (DAG) [63], $G_m = (V_m, E_m)$. The nodes in V_m are the logic gates in the circuit and the primary inputs and outputs of the circuit. Each node corresponding to a gate in the DAG has a weight equal to the gate delay. Each edge in E_m denotes the interconnect from one node to the other. Each edge has a wire delay under a given layer directive. Here, each edge will have M choices, each with different delay values, and also different associated costs. As shown in [53], even a simplified form of this problem, where only a single routing tree topology of a net is considered, is NP-complete.

In this section, we propose a simple yet efficient heuristic. The main idea is to use as few higher directives as possible by only promoting the currently most timing-critical nets. We will explain our algorithm assuming we have just started the k^{th} iteration of CATALYST, and there are some nets with directives from last iteration. A demotion stage is first performed to check whether a net can be demoted from its original directive to lower directive with no timing violations created. This helps create a small set of layer directives for the congestion-aware directive assignment step. Note that in the first iteration of CATALYST, all the nets are in D_1 and no demotion is required. Next, the timing-driven promotion stage starts. This stage assigns timing-critical nets to higher directives gradually, by looping through each directive from D_2 to D_M. In each step with target directive D_j, we will first put all the timing-critical nets to a

list A, and then sort them by their slacks[3] in an increasing order. Then, in this order, we loop through each net n_i in list A, and promote n_i to D_j if its current directive $D(n_i)$ is lower than D_j (note that n_i may have already been assigned to a higher directive from a previous CATALYST iteration). If n_i is promoted, then the timing graph is updated with incremental static timing analysis. Since the timing of other nets may change due to the promotion of n_i, the subsequent nets in A may be skipped in later processing if they become non-critical. This heuristic promotes only "necessary nets" to higher directives to avoid potential congestion problems.

3.5 Congestion- and timing-aware directive assignment

There are two major goals of Subproblem 2: first, to maintain the initial directive assignment to the extent permitted by congestion, while demoting some of the directives that may cause congestion problems, and second, to assign as many nets as possible to higher directives/layers in order to reduce buffer usage without degrading congestion. This is different from the method for Subproblem 1 which tries to promote as few nets as possible to higher layers. Besides promoting timing-critical nets, this step also promotes non-timing-critical nets for potential buffer savings. The positive slack of a net under the linear delay model is obtained assuming optimal buffering with the current directive, and if it is promoted to higher directives, fewer buffers will be required to keep the positive slack.

Subproblem 2 can be formulated as follows. Given a circuit, a congestion constraint and a group of nets with initial directives, try to maintain as many of the initial directives as possible, and further promote as many nets as possible to higher directives to maximize the sum of scores of the promoted nets. Here, the score of a net is used to quantify the benefits in timing improvement and buffer savings by promoting it to a higher layer. In this work, as a first attempt, we simply use the following score function for net n_i:

$$w(n_i) = \exp(-s(n_i)/T_c), \tag{3.1}$$

where $s(n_i)$ is the worst slack of all the sinks of n_i, and T_c is the clock period used for normalization.

[3]The slack of a net is the worst slack of all the sinks of the net.

Next, we will first present the overall algorithm to solve Subproblem 2 in Section 3.5.1, and then discuss two important procedures used in our algorithm in Section 3.5.2 and 3.5.3, respectively.

3.5.1 Overall algorithm for Subproblem 2

In this section, we will introduce the algorithm for Subproblem 2. For convenience of later reference, we refer to our algorithm as CADA (**C**ongestion and timing **A**ware **D**irective **A**ssignment). A key function of CADA is to control congestion, which is achieved by performing 2D directive-aware routing and directive assignment. In this work, a 2D-routing engine adapted from MaizeRouter [48] is used but the maze routing in MaizeRouter is replaced by extreme edge shifting (refer to [48] for details about this technique) to improve the speed; however, it should be pointed out that our flow can work with any other 2D routing engine. To deal with directives in 2D routing, we use the progressive projection method in [56] as discussed in Sec. 3.2.2. In the LDA problem formulation, the constraint on congestion requires $g_{LDA} \leq \beta g_{org}$. To consider this directly inside CADA, it is necessary to invoke global routers before and after LDA. However, this may be inefficient. Instead, for simplicity, we use a conservative method to control congestion: a net can be promoted to directive D_j only if routing a net in this directive causes zero overflow; otherwise, the net will stay in the lowest directive D_1. Note that in the solution of Subproblem 1, this strategy is not used since the congestion of a net at that step cannot be efficiently obtained.

The basic flow of CADA is shown in Fig. 3.4. CADA has three stages: initial directive adjustment, greedy directive assignment and directive assignment refinement. First, the initial directive assignment from the timing-driven directive assignment step (Sec. 3.4) is re-evaluated in terms of congestion. At the allowable congestion level, we will try to maintain as many initial directives as possible, and demote the nets to lower directives if the initial assignment causes congestion. The purpose of the second stage is to assign/promote as many nets as possible to the higher directives to further improve timing and reduce potential buffer usage. The goal of the third stage is to verify the effects of directive assignment on congestion and to fine-tune the layer directives by performing a trial routing process with the directive assignment.

As stated earlier, at the first stage, only the nets with initial directives from the timing-driven directive assignment step are processed. With the progressive projection method, the nets with top directive D_M will be routed first. After 2D-routing, we will first sort these nets by their scores as defined in (3.1), and then process the nets one by one. We first attempt to assign a

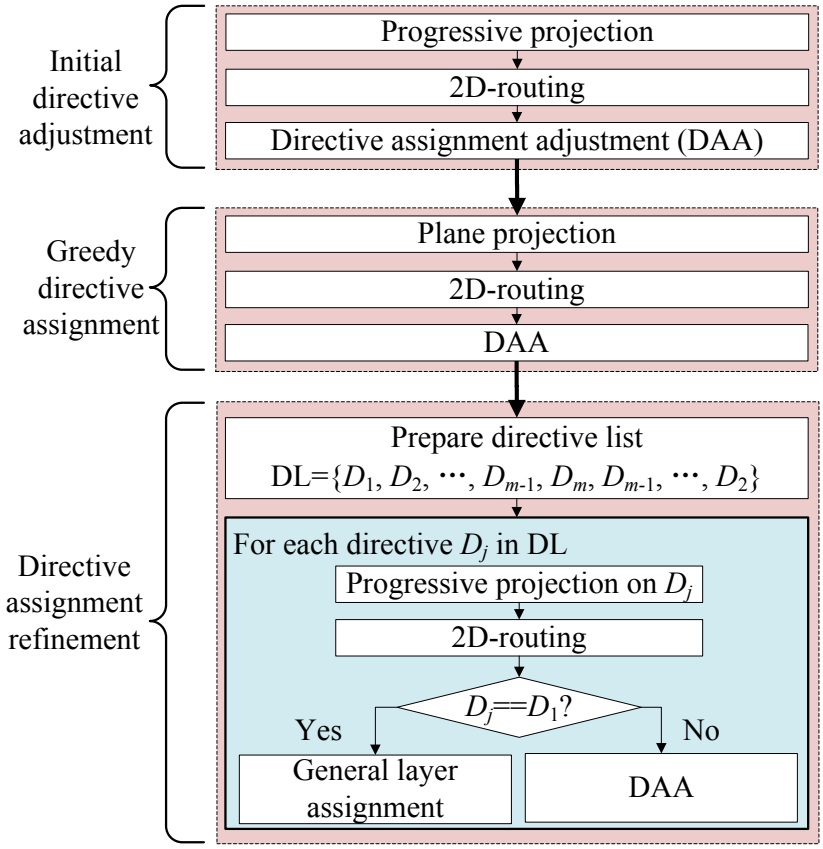


Figure 3.4: Basic flow of congestion- and timing-aware directive assignment.

net to its initial directive, and then check the congestion. If no congestion violation is found, this net will be marked to be assigned to that directive; otherwise, an attempt to assign it to lower directives will be made until a directive without causing congestion is found, or the lowest directive D_1 is reached. This procedure to find the best directive with an initial target directive is called **Directive Assignment Adjustment (DAA)**, and will be discussed in more details in Sec. 3.5.3. After the nets with D_M are all routed, we repeat a similar process for nets with directive D_{M-1}. This process iterates until all the nets with initial directives are processed. Since the nets with initial directives are the most timing-critical nets, and their number is typically relatively small as compared to the total number of nets, the wiring resources consumed by these nets are locked down after this step, which means that their routing paths will be kept in the 3D GRG and not be changed in later stages.

The second stage works on the nets without initial directives, with the goal to maximizing the sum of timing scores of nets promoted to directives higher than D_1. First, all of the layers are projected to a 2D GRG and 2D-routing is performed. Next, we perform directive assignment. Here we use a greedy method. First, all the nets without directives are sorted by their timing scores in a decreasing order, and then in this order, each net is tried to be promoted to higher directives. For each net, directives are tried one by one from D_M to D_1. This trial repeats until a directive without causing congestion is found, or the lowest directive D_1 is reached. This process is performed by calling DAA procedure on each net with D_M as the initial target directive. The intuition is that we first assume all the nets can be greedily assigned to the highest directive D_M and then call the DAA procedure to find the best directive for each net under the congestion constraints.

The third stage fine-tunes the layer directives obtained from the second stage. After the second stage, a large number of nets could have been promoted to directives higher than D_1. Note that at the second stage, when performing 2D routing on these nets, no directives are assumed and the routing resources from all the layers can be used to route them. In the third stage, these nets are constrained by their directives, and then the previous unconstrained 2D routing results become inaccurate. Therefore, we must reroute some nets, and we do so using the progressive projection method. As suggested by [56], we should route the nets in the order: $D_M \rightarrow D_{M-1} \cdots \rightarrow D_1$. However, we find this order is not appropriate for the purposes of directive refinement. This can be illustrated by the following example, which shows the impact of net ordering. Consider a net A that has a small timing score and is left with directive D_1. It

sits on top of a routing blockage and the only possible route for this net is to route on layers in D_3 passing a g-edge e with only one available track. Another net B, which was not assigned with a layer directive from the timing driven directive assignment step but has a higher score than net A, may have two possible routes to choose: one passing through the same g-edge e using D_3, and another taking a path in directive D_2 without creating overflow. When B is processed during greedy directive assignment step, the information of net A is not seen since net B has a higher score than net A, and B is assigned to D_3. In this scenario, if we route the nets with directive D_3 including B first, B will take the pass through e in D_3. Then later when net A with directive D_1 is routed, an overflow is created. In contrast, if we route A with directive D_1 first, A will take resource on g-edge e, and then B would be re-assigned to D_2 and no overflow is created.

Therefore, we proceed in the order $D_1 \rightarrow D_2 \cdots \rightarrow D_{M-1} \rightarrow D_M \rightarrow D_{M-1} \cdots \rightarrow D_2$ at the stage of directive assignment refinement. We explain this using an example with three directives. We will first route the nets with directive D_1 and then perform **general layer assignment** (to be discussed in Section 3.5.2) without promotion and demotion. Next, we will route the nets with directive D_2 with the solution from previous step treated as existing wires. Then we will perform DAA for nets with directive D_2 to adjust the existing directive D_2, i.e., demote a net to D_1 if congestion occurs. Since the nets with directive D_1 have been routed and their routing solutions can be seen now, the directive adjustment for the nets with directive D_2 will be more realistic than that in the $D_3 \rightarrow D_2 \rightarrow D_1$ flow. After demotion for the nets with directive D_2 finishes, we start the same process for the nets with directive D_3, with all the previous routing solutions treated as existing wires. After routing pass D_3 completes, we need to rip-up-reroute all the nets with directive D_2 and call DAA for them again including some nets just demoted from D_3, with existing wires from the nets with directives D_1 and D_3. Since our purpose is to generate layer directives and not to perform routing, we do not need another pass D_1 to reroute the nets in that directive.

3.5.2 General layer assignment

In the third stage, general layer assignment will be performed for the nets with directive D_1 without layer directives. Our goal is to minimize the number of vias and keep the overflow for the 3D GRG the same as that for the 2D GRG. Several layer assignment algorithms have been proposed in the literature [10, 12, 64]. For ease of implementation, we use a greedy layer

assignment algorithm, but note that any of the existing layer assignment can be used in our framework.

We first sort all the nets by the total wirelength in the nonincreasing order, and then perform layer assignment for each net in the order. The reason for this ordering is that generally, the nets with larger wirelength will take more routing resources, which implies less flexibility in layer assignment, and then should be processed earlier. For each net n_i, we will loop through each segment s on the routing path of n_i. For each segment s, we will first try to find a layer which can hold it without overflow. If such a layer cannot be found, we cut the segment and find the best layer for each edge on the segment. The best layer here is the lowest layer on which there is no overflow, or on which congestion is smallest among all the layers if all the layers have overflows. Here, we will try to assign these nets to lower layers first, since no layer directives will be assigned to them and assignment to higher layers is a wasteful use of resources there. Moreover, trying to use lower layers tends to help reduce the via count.

3.5.3 Directive assignment adjustment (DAA)

This procedure is an important procedure in our algorithm, and is used in all the three stages to adjust the given directive of a net. The inputs are a net n_i and its initial target directive D_j, and the output is the adjusted directive D_k. Given a net n_i with its initial target directive D_j, DAA will first attempt to assign the net to the layers specified by D_j using the general layer assignment procedure, with two differences. First, this method uses a constrained layer range, rather than allowing assignment to any layer. Second, the purpose of this attempt is just to quickly check whether this assignment will cause overflow, and therefore, this attempt assignment will stop at once if overflow is found on a g-edge, instead of continuing to complete the assignment of the whole net. If the attempt of assigning n_i to D_j finally succeeds without overflow, the routing resources consumed by the net will be added to 3D GRG, and directive D_j will be returned as the adjusted directive for n_i. Otherwise, D_{j-1} will be tried with the same procedure. This process repeats until n_i can be assigned to a directive D_k without overflow, or the directive D_1 is reached.

3.6 Experimental results

CATALYST has been implemented using C++ and Tcl (tool command language) in an industrial physical synthesis tool. This section presents the experimental analysis on a set of advanced industrial designs described in Table 3.1. The first four letters of the circuit name explains its technology node. The circuits with "top" in the names are top-level designs (designs at the first level hierarchy where a majority of gates are buffers), while the rest of circuits are random-logic macros. All the experiments run on 64-bit Linux servers with 4 octa-core CPUs (Intel® Xeon® X7560 2.27 GHz). In our experiments, the parameters used in the stopping criterion of CATALYST are: $\theta = 10\%$ and $n_{itr} = 2$.

Table 3.1: Information for benchmark circuits. The column "Thick layers" lists the distribution of thick layers.

Circuits	#gates	#nets	Thick layers
cu32top1	329,082	467,889	[M6, M7]: 2X; [M8, M9]: 4X; [M10, M11]: 16X
cu32rlm2	1,392,744	1,505,994	[M6, M7]: 2X; [M8, M9]: 4X;
cu32rlm3	892,452	935,582	[M6, M7]: 2X; [M8, M9]: 4X;
cu45top1	45,655	76,062	[M6, M8]: 2X; [M9, M10]: 10X
cu45rlm2	2,464,339	2,555,753	[M6, M8]: 2X; [M9, M10]: 10X
cu45rlm3	1,282,736	1,405,029	[M6, M8]: 2X;
cu65rlm1	895,334	916,865	[M5, M7]: 2X; [M8, M9]: 4X

We first demonstrate the immediate impact of CATALYST on timing and congestion by comparing different LDA algorithms in Section 3.6.1, and then present the impact of CATALYST in the physical synthesis flow in Section 3.6.2. A full-blown industrial global router using a different routing algorithm than that used in CATALYST, is used to evaluate the congestion using the PWC metric. Note that in all of our experiments, some routing resources are reserved for the power grid and for clock routing. In addition, all timing numbers are generated using an industrial static timing analyzer.

Table 3.2: Comparison among baseline, CATALYST (CATA in short) and simpLDA. Timing metrics are computed with the linear delay model. r_{net} is the percentage of nets promoted to thick layers, and r_{wl} is the percentage of the routed wirelength of these nets to the total wirelength.

Circuit	WSLK (ns)			FOM (ns)			n_{neg}			PWC (%)			r_{net} (%)	r_{wl} (%)
	Base	CATA	simpLDA	Base	CATA	simpLDA	Base	CATA	simpLDA	Base	CATA	simpLDA	CATA	CATA
cu32top1	-14.46	-11.56	-2.95	-44794.2	-16870.0	-2118.6	87739	36270	22339	96.83	94.97	127.56	30.52	52.39
cu32rlm2	-1.93	-1.76	-1.78	-8861.6	-1031.6	-617.4	17547	7414	4093	88.49	89.87	295.58	5.67	24.41
cu32rlm3	-1.51	-0.41	-0.37	-5125.0	-264.1	-183.1	19175	2763	1626	87.42	88.36	198.39	8.63	36.11
cu45top1	-2.41	-2.39	-2.17	-2513.0	-2074.1	-1985.5	3026	967	960	87.44	86.95	121.62	46.49	73.76
cu45rlm2	-1.48	-0.74	-0.30	-2464.1	-134.1	-38.0	10359	1055	398	87.13	87.51	N/A	12.01	30.20
cu45rlm3	-0.33	-0.05	-0.05	-583.6	-4.7	-4.7	7600	258	258	87.58	87.26	N/A	19.41	24.05
cu65rlm1	-1.00	-0.37	-0.37	-726.8	-94.4	-113.5	7521	1323	1368	89.92	89.85	661.32	17.34	43.92
Average	0	0.84	2.16	0	6370.8	8572.5	1	0.23	0.16	1	1.00	3.13	20.01	40.69

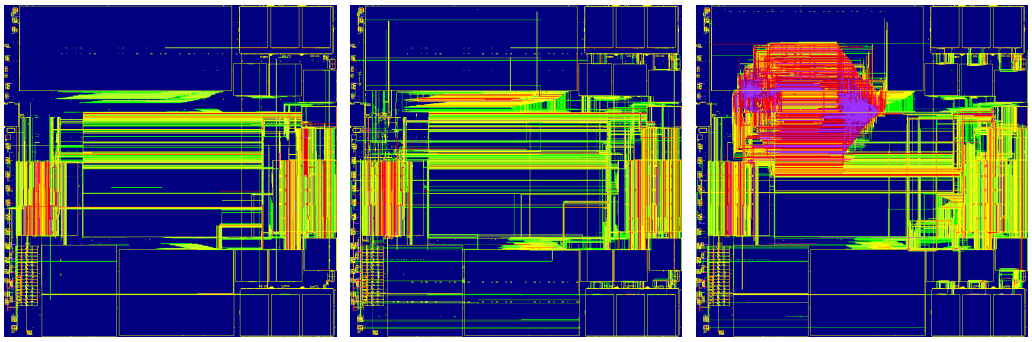
Table 3.3: Comparison among baseline, CATALYST (CATA in short) and NoDA physical synthesis flows.

Circuit	WSLK (ns)			FOM (ns)			n_{neg}			PWC (%)			Buffer area ($\times 10^6$)			CPU time (h)		
	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA	Base	CATA	NoDA
cu32top1	-2.94	-2.96	-9.90	-2488.75	-2564.09	-23497.00	21982	21251	73652	90.13	90.05	72.61	9.54	7.78	14.43	59.7	44.7	60.2
cu32rlm2	-0.16	-0.14	-0.52	-18.87	-7.93	-1333.68	783	442	7001	88.02	87.70	88.51	0.82	0.78	1.17	34.1	42.4	53.0
cu32rlm3	-0.04	-0.02	-0.78	-0.15	-0.03	-408.34	6	2	2004	87.53	87.66	93.37	1.03	0.93	1.58	17.3	19.2	22.0
cu45top1	-2.42	-2.42	-2.60	-2139.97	-2130.29	-3187.95	963	963	4720	80.70	81.07	73.73	2.46	2.09	2.91	10.6	11.9	10.2
cu45rlm2	-0.34	-0.38	-1.45	-23.03	-17.03	-5258.76	336	109	21037	87.39	86.78	87.13	4.00	3.55	4.67	63.8	67.1	76.7
cu45rlm3	-0.05	-0.05	-0.35	-12.01	-3.37	-512.86	521	163	6924	87.96	87.85	89.15	3.99	3.85	4.75	37.5	41.5	54.6
cu65rlm1	-0.24	-0.24	-0.75	-4.30	-4.21	-68.38	153	154	749	89.79	89.57	91.94	1.21	1.15	1.53	15.3	17.1	19.3
Average	0	0.00	-1.45	0	-5.69	-4225.70	1	0.64	61.71	1	1.00	0.98	1	0.90	1.32	1	1.07	1.24

3.6.1 The immediate impact of CATALYST

We compare three cases here, just before the global buffering phase: first, the traditional flow with no LDA (baseline); second, the results of CATALYST (denoted as "CATA"); and third, the results of an alternative LDA method (simpLDA). For the third set of results, since no public tools to generate layer directives are available, we create a simple layer directive assignment algorithm, simpLDA, which promotes nets only based on their worst slacks. We first sort all the nets based on their worst slacks in increasing order, and then promote the first n_M nets to the top directive D_M, the next n_{M-1} nets to the directive D_{M-1}, and so on. For a fair comparison, we ensure the number of nets promoted to each directive is the same for simpLDA and CATALYST. In this set of experiments, given the same placements, we run simpLDA and CATALYST and then evaluate the timing and congestion. Since this is done before global buffering, the linear delay model is used in computing all the timing metrics in this set of results.

Table 3.2 presents the comparison of timing and congestion results among the three cases, which shows the state just prior to global buffering. The row "Average" lists the average on differences from baseline (for WSLK and FOM), or the ratios to baseline (for n_{neg} and PWC), or real values (for r_{net} and r_{wl}) over all the circuits. The congestion entries shown as "N/A" ("not available") correspond to cases where global routing cannot finish after running 6 hours due to high congestion and is terminated manually. On average, CATALYST improves WSLK by 0.8 ns, improves FOM by 6370.8 ns, and reduces n_{neg} by 77%, while maintaining similar congestion to the baseline. On the other hand, although simpLDA improves the timing more than CATALYST, it degrades the congestion significantly. Fig. 3.5 shows the congestion plots for baseline, CATALYST and simpLDA on cu45top1. From the pictures, we can see that even after promoting 46% nets to higher directives, the congestion of CATALYST is still similar to that of baseline, while that of simpLDA is much worse. The analysis demonstrates the effectiveness of CATALYST in improving timing and controlling congestion. Though simpLDA promotes the same number of nets as CATALYST, the congestion is quite different, which shows that assigning which nets to higher directives/layers is important, and improper choices can choke the router.



(a) Baseline. (b) CATALYST. (c) simpLDA.

Figure 3.5: Congestion plots for cu45top1 (the color map used is the same as that in Fig. 2.1(a)).

3.6.2 The impact of CATALYST in the flow

This section discusses the impact of CATALYST when embedded in the physical synthesis flow. We compare three flows: baseline flow (Fig. 3.2(a)) that integrates the directive assignment algorithms from [18] in the global buffering stage, CATALYST flow (Fig. 3.2(b)) that adds CATALYST to the baseline flow after placement stage, and an altered flow that removes the two sets of directive assignment algorithms, denoted as "NoDA" flow. Note that the baseline flow is the state of the art, corresponding to a relatively recent paper [18]. Prior to this, a NoDA-like flow was widely used and possibly is still in use. The comparison with NoDA highlights the impacts of directive assignment and reveals some data not found in the previous papers including [18], such as the overall impact of directive assignment through the design flow. Here simpLDA is not tested in the whole flow since it has already been seen to provide unacceptably high congestion. For runtime consideration, we stop the flows after global routing, and then evaluate the timing and congestion of the designs. At this stage, the accurate RICE model is used in timing analysis.

Table 3.3 presents the comparison of timing, congestion, buffer area, and CPU time at the end of different flows. The row "Average" lists the average on differences from baseline (for WSLK and FOM) or the ratios to baseline (for all other metrics) over all the circuits.

We first compare the CATALYST flow with baseline. Though WSLK and FOM are quite similar for two flows, the CATALYST flow reduces the n_{neg} by 36% on average, which indicates fewer further efforts are required to finally close timing. Moreover, the CATALYST flow can

reduce the buffer area by 10% on average and up to 18%[4]. This indicates that the timing optimization techniques later in the baseline flow, other than CATALYST, can also achieve similar WSLK and FOM to the CATALYST flow, but with an expense of inserting more buffers, and a series of other consequences such as higher cost, larger power, etc. We also observe that the buffer saving tends to be larger for the top-level designs than macros (17% vs. 7% on average), since in top level designs, a larger portion of nets are very long nets, and without layer directive assignment, more buffers have to be inserted to close the timing. Furthermore, the CATALYST flow achieves similar congestion to baseline, which demonstrates again that CATALYST has good control on the congestion. Finally, the runtime of CATALYST flow is 7% more than baseline on average, and even for the absolute values, the runtime of CATALYST flow is still acceptable in practice. Note that for cu32top1, CATALYST runs 15 hours faster than baseline. This indicates that for some designs, a better initial solution obtained by CATALYST can speed up the later optimization stages significantly.

Next we further compare NoDA flow with the other two. The timing, buffer area and runtime of NoDA flow are significantly worse than those of the other two flows. Compared with the CATALYST flow, NoDA requires 47% more buffers (or CATALYST flow could save 31% than NoDA) on average.

The analyses above clearly show the impact of CATALYST: it can improve the timing and greatly reduce the buffer area. Since buffers could account for more than 20% of the gates in a modern design [58], the reduction of buffer usage can significantly decrease the design power and cost, and improve the design closure.

3.7 Conclusion

The availability of multiple thick metal layers provides a new dimension in timing optimization besides gate/wire sizing and buffer insertion, but the thick layers are underused due to the lack of effective CAD tools. To solve this problem, we have formulated the LDA problem and solved it using a two-step algorithm. First, a greedy heuristic is used to promote only the timing-critical nets to thick layers regardless of congestion. Second, the layer directives from first step are adjusted considering real congestion, and the algorithm further promotes as many nets as possible to the thick layers according to their timing scores. The congestion control in the algorithm is

[4]There are some buffers that are fixed in the designs and cannot be changed.

implemented based on a trial layer assignment process along with the embedded routing engine. Moreover, based on our algorithm, we have proposed a new physical synthesis flow by adding a CATALYST stage to the old flow before global buffering. Experiments have been conducted to test our algorithm and flow on a set of industrial designs from different technology nodes including top level designs and random logic macros. The results have demonstrated that our algorithm could promote 20% of the nets to thick layers without worsening the congestion and the flow with CATALYST could improve the n_{neg} by 36% and reduce the buffer area by 10% on average.

Chapter 4

Dummy Fill Optimization for Enhanced Manufacturability

4.1 Introduction

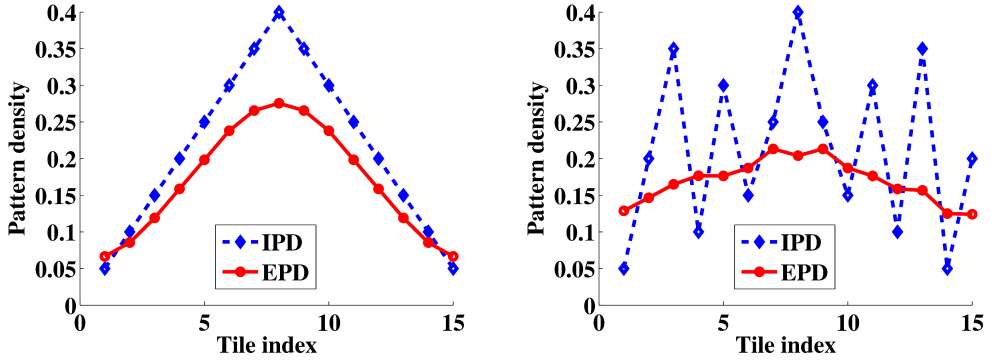
As mentioned in Section 1, CMP is used to achieve the planarization on the wafer. In nanometer-scale IC technologies, variations in the CMP process represent a significant potential source of yield loss. For example, oxide CMP is used to polish the ILD layer to ensure a near-planar surface before depositing and patterning a metal layer. If significant surface topography variations are seen after this process, then the depth of focus in lithography is affected, which in turn leads to variations in the critical dimension [3], resulting in performance degradation and yield loss. In order to improve the quality of CMP, in addition to the metalized interconnects that serve an electrical function, dummy features are typically added to the layout to control the variation in the post-CMP topology [65]. A dummy feature, also referred to as dummy fill, may either be connected to power/ground (tied fill) or left floating (floating fill), and lead to increased parasitic capacitance in the layout. Floating fill increases the coupling capacitance uncertainty and can lead to signal-integrity issues, while tied fill reduces this problem, but results in high routing costs, increasing the likelihood of requiring engineering change orders [3]. Therefore, it is desirable to reduce the volume of dummy fill inserted into a layout.

It is known that the post-CMP ILD thickness is linearly determined by the effective pattern density (EPD) of the layout [5]. The EPD is formally defined in Section 4.2, but coarsely speaking, the EPD at a given location is a weighted average of the wire density in its neighborhood. If

the EPD can be made more even throughout the layout, the variation of the ILD thickness after CMP can be reduced, leading to a reduction in the amount of dummy fill. Since routing plays a very major role in deciding the spatial distribution of wires in the layout, a good way to achieve greater uniformity in the EPD is to leverage the router in ensuring that the density after routing is as uniform as possible, while meeting other traditional routing objectives.

Several routing algorithms considering wire distribution have been proposed in the literature. The algorithm in [66] is among the first to incorporate CMP variation into a routing algorithm, and it proceeds by attempting to balance the initial pattern density (IPD) to decrease CMP variation. The IPD is formally defined in Section 4.2, and is essentially the wire density. A global routing algorithm considering Cu CMP variation is proposed in [31], using an empirically developed predictive CMP density model from industry. As pointed out in [67], the algorithms in [31, 66] attempt to optimize CMP variation by only considering the IPD *inside* a routing tile/grid, which is not a right metric for CMP control, since the topographic variation is a long range effect that is affected by the IPD in neighboring tiles. In [67] a multilevel routing algorithm for oxide CMP variation is presented, using the IPD gradient as an optimization objective. However, this objective also suffers from significant limitations, as demonstrated by an example in Fig. 4.1, which shows two different layouts of an illustrative circuit. Layout II shows larger IPD gradients, but due to averaging effects (the size of weighting window, defined in Section 4.2, is 5 tiles for each layout), the EPD/CMP variation is smaller. This indicates that the gradient of the IPD may not be a good metric in the optimization of CMP variation. This claim will be demonstrated experimentally in Section 4.6. In essence, the methods in [31, 66, 67] attempt to decrease the variation of CMP only according to the IPD in a tile and its immediate neighbors. However, metrics such as the IPD and the IPD gradient are only indirect measures of the EPD, and their ability to optimize CMP variation is likely to be inferior to an approach that addresses the EPD variation more directly.

In addition, two routing algorithms considering EPD optimization directly have been presented [68, 69]. In [68, 69], the EPD is taken as part of the cost of a routing tile directly. While this is better than using the IPD in the cost function, the approach only considers the EPD *inside* a tile t_i as a route passes through t_i, but the impact of this route on the EPD of neighboring tiles is not considered in the cost function. Moreover, the amount of dummy fill is not directly optimized as an objective of routing. These factors limit the effectiveness of the optimization.



(a) Layout I: IPD gradient ≤ 0.05. EPD variation = 0.209.

(b) Layout II: IPD gradient ≥ 0.10. EPD variation = 0.089.

Figure 4.1: IPD and EPD topographies for Layout I and II. Layout I has smaller IPD gradients, but larger EPD variation.

This chapter proposes a global routing algorithm that incorporates the optimization of oxide CMP variation, in addition to the usual routing objectives. Our goal is to minimize the required dummy fill under an accurate oxide CMP model [5], and we demonstrate, both theoretically and through experiments, that a good surrogate for this objective is to minimize the maximum EPD during routing. We elaborate cost functions to achieve this goal, and build a router that attempts to minimize the maximal EPD in routing. Our router is based on NTHU-Route 2.0 [11], which will be introduced in Section 4.3. Experimental results demonstrate that our algorithm can reduce the dummy fill substantially.

The remainder of this chapter is organized as follows. We introduce the global routing model, CMP model, and CMP dummy fill methods in Section 4.2. Next, the algorithm of NTHU-Route 2.0 is reviewed in Section 4.3, and the flow of the proposed routing algorithm, considering minimization of dummy fill, is described in Section 4.4. The cost function that guides the routing process is presented in Section 4.5. Experimental results are then reported and analyzed in Section 4.6. Finally, a conclusion is provided in Section 4.7.

4.2 Preliminaries

The goal of our global router is to optimize the overflow[1], wirelength, and the amount of dummy fill, D, inserted for CMP planarization. Like most global routing approaches, our approach tessellates the chip into $n_r \times n_c$ grids, and constructs the GRG as illustrated in Fig. 2.3. In this chapter, we will continue using the definition of overflow and related symbols in Section 2.2.

In relation with the dummy fill metric, D, we define the *initial pattern density* (IPD) in a region as the ratio of the area of metal in the region to its total area. In our routing model, we assume reserved horizontal/vertical routing layers, and separately consider routing edges in the horizontal and vertical directions. In practice, since our algorithm will associate the increase in the EPD with edges of the GRG, we use a shifted grid for CMP computations. On the horizontal layer, each CMP grid/tile around an edge e has the same size as a GRG grid, but is centered about the edge instead of the vertex (i.e., it is offset to the right by half a grid relative to the GRG grid centered at the left endpoint of e). A CMP tile on the vertical layer is similarly defined.

Next, we introduce the oxide CMP model in [5]. Fig. 4.2 shows a schematic that describes the primary variables used in this model, which defines the ILD thickness z at location (x, y) in the layout. This can be calculated using the following formula:

$$z = \begin{cases} z_0 - [K\tau/\rho(x,y)] & \tau \leq (\rho z_1/K), \\ z_0 - z_1 - K\tau + \rho(x,y)z_1 & \tau \geq (\rho z_1/K), \end{cases} \tag{4.1}$$

where K is the blanket oxide polishing rate, z_0 is the thickness of oxide deposition, z_1 is the initial step height, τ is the total polish time, and $\rho(x, y)$ is the EPD in location (x, y) before oxide CMP. The variables K, z_0, z_1 and τ are assumed to be constants for a specific CMP process.

Generally, the total polish time τ is larger than $(\rho z_1/K)$, and therefore, the final oxide thickness z is between 0 and $(z_0 - z_1)$. As a consequence, the final ILD thickness in different locations has an affine relationship with the EPD in that location. The *effective pattern density* (EPD) can be calculated by convolving the IPD with the weighting function [5]:

$$f(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right). \tag{4.2}$$

[1] In the work presented in this chapter, we use overflow metrics instead of the ACE metric proposed in Chapter 2, since this work was done before we proposed ACE metric and we have not had a chance to reevaluate the congestion using ACE metric.

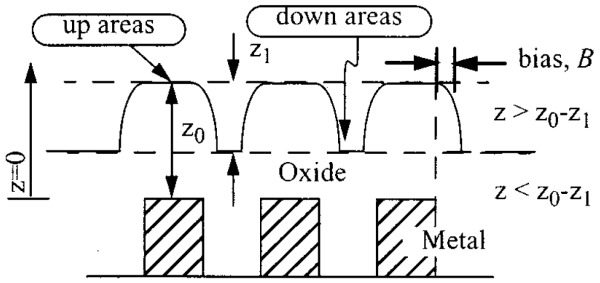


Figure 4.2: Depiction of the primary variables in the oxide CMP model [5].

Here, $f(x, y)$ is a Gaussian function with standard deviation σ. This function can be discretized to the tile grid, and truncated beyond a *weighting window* of size $L \times L$ distance units, equivalent to $(2l+1) \times (2l+1)$ tiles. For this Gaussian function, $\sigma = L/2$, equivalent to $\lfloor (2l+1)/2 \rfloor$ tiles, and the value of $f(x, y)$ will be truncated to 0 beyond the weighting window. If we discretize the value of the weight function in tile t_{ij} to $f(i, j)$, and denote the IPD of t_{ij} as d_{ij}, then the EPD of t_{ij}, ρ_{ij}, can be calculated by a circular convolution as [44]:

$$\rho_{ij} = \sum_{n_1=-l}^{l} \sum_{n_2=-l}^{l} d_{i+n_1, j+n_2} \times f(n_1, n_2). \tag{4.3}$$

Thus, the EPD of a tile is calculated as a weighted sum of IPDs.

Dummy fill optimization inserts the minimum fill, D, to ensure a spatially even EPD, i.e.,

$$\mathrm{ran}(\rho) = \rho_H - \rho_L \leq \epsilon, \tag{4.4}$$

where ϵ is a user-specified parameter, and $\mathrm{ran}(.)$ is an operator that finds the range of a function, i.e., the difference between its maximum and minimum. Here, ρ_H (ρ_L) is the maximum (minimum) value of the EPD, ρ, over the layout.

Methods for dummy fill insertion can be classified into two categories: rule-based and model-based. Rule-based methods reduce the CMP variation by inserting dummy fill to ensure that the IPD in each region meets a certain threshold. While these methods are simple to execute, they are heuristic in nature and do not guarantee optimality. Model-based methods use CMP models such as the one introduced above, and directly optimize the variation of the EPD globally based on the model. These methods are more computational, but can use a significantly lower amount of dummy fill [44]. The work in [44] presents a linear programming based algorithm to

optimize the total amount of dummy fill, D, with constraints on $\mathrm{ran}(\rho)$. In this chapter, we use D as the metric to evaluate the different global routing solutions, and for convenience, we use the density of dummy fill, i.e., the ratio of the area of dummy fill to the total area of the layout, instead of the area of dummy fill, to measure D.

4.3 Previous work

Our routing algorithm is based on NTHU-Route 2.0 (NTHR) [11], which is briefly reviewed in this section. There are four stages in the NTHR algorithm: the initial stage, the main stage, the refinement stage, and the layer assignment stage.

The purpose of the initial stage is to generate an initial global routing solution. First, a multi-layer design is projected to a 2D plane, and then FLUTE [70] is used to decompose each multi-pin net into a set of two-pin nets. Next, NTHR sets up the probabilistic congestion map by adding half a unit of demand to each edge on the two probabilistic L-shape routes, or a full demand to each edge on a straight route. Then the topology of every multi-pin net is modified using the edge-shifting technique [27]. Finally, every two-pin net is routed by L-shaped pattern routing. During this stage, the cost function in [27] is used to calculate the cost of an edge:

$$\mathcal{C}_e' = 1 + \frac{p_3}{1 + e^{-p_4(u_e - c_e)}}, \tag{4.5}$$

where p_3 and p_4 are user-defined parameters. In NTHR, $p_3 = 0.8$, $p_4 = 2$.

In the main stage, the initial solution is improved by iteratively ripping up and rerouting (RRR) every congested two-pin net. A two-pin net is considered to be congested if there are one or more overflowing edges on its path. NTHR uses a technique of identifying congested regions to choose the ordering for RRR. In the RRR process, each ripped-up two-pin net is first rerouted by monotonic routing [27], and then by the adaptive multi-source multi-sink maze routing method, if an overflow-free path cannot be found by monotonic routing. The RRR process is repeated until the total overflow is no more than a predefined threshold or the number of iterations reaches a predefined value. A history-based cost function is used in this stage, and the basic form is as follows:

$$\mathcal{C}_e'' = X_e \times B_e + H_e \times G_e + V_e \times B_e, \tag{4.6}$$

where $\mathcal{C}_e''$ is the cost of edge e; X_e is the wirelength cost, which is set to 1 since the wirelength will increase by 1 when a net is routed through e; $H_e \times G_e$ is congestion cost, H_e is the historic

overflow term and G_e is the penalty cost (refer to [11] for details); V_e is via cost; B_e is a factor defined as follows:

$$B_e = 1 - e^{-\alpha e^{-\beta i}}, \tag{4.7}$$

where α and β are user-defined parameters, and i is the current iteration count. In NTHR, $\alpha = 5$, and $\beta = 0.1$. Thus, B_e will be bounded between 1 and 0, and will decrease as i increases. By incorporating B_e into the wirelength cost and via cost, the congestion cost will gradually take the dominant role in the total cost as the iteration number increases, which is helpful for NTHR to obtain paths without overflow.

The main purpose of the refinement stage is to search an overflow-free path for every congested two-pin net, which is adapted from the main stage with the following two major differences. First, while in the main stage, the congested region identification technique is used to determine the RRR order, in the refinement stage, NTHR rips up and reroutes every congested two-pin net in the nonincreasing order of the number of overflowed edges on the path of the net. Second, the cost of an edge e is defined as follows:

$$\mathcal{C}_e''' = \begin{cases} 1 & \text{if edge } e \text{ has overflow,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

For multi-layer designs, the layer assignment stage is performed to map the routing solution from the projected 2D plane to the original multiple layers with the algorithm in [64].

4.4 The flow of the CMP-aware routing algorithm

There are two major differences between the proposed routing algorithm and the original NTHR.

Firstly, we augment NTHR so that the router becomes CMP-aware. We study several possible metrics related to dummy fill and their correlations with the amount of inserted dummy fill, D, and then through a set of theoretical analyses and experiments, we develop a surrogate metric ρ_H for minimizing D. Based on the surrogate metric, we elaborate effective cost functions to be integrated into NTHR to perform dummy fill optimization by minimizing ρ_H. A detailed presentation of the surrogate metric and the cost functions used to guide dummy fill optimization is provided in Section 4.5.

Secondly, we add another stage after the refinement stage in order to reduce ρ_H and thus reduce D further by ripping up and rerouting the nets passing through the tiles related to ρ_H[2]. This stage, which we call *the EPD postprocessing stage*, has the following steps. We initially identify the tile t_k that has the maximum EPD in the layout, and then find all the two-pin nets whose paths pass through the edges in the weighting window of t_k. Note that the maximal EPD is attributed not only to wires in t_k but also to wires in other tiles within the weighting window of t_k. Next, we sort all of these two-pin nets in nonincreasing order of the minimal distance of the two pins to t_k. The motivation is that for the two-pin nets further from t_k, it should be easier to find an alternative path that does not pass through the weighting window of t_k. In case of ties, we employ other sorting criteria, such as the number of edges that are both in the weighting window of t_k and on the path of a two-pin net, and the total overflow along the path of a two-pin net.

After this step, we rip up and reroute the nets one by one using a method adapted from the refinement stage of NTHR. The goal is to find a path to decrease ρ_H, and at the same time ensure that the overflow is not increased. The cost function used here will be introduced in Section 4.5.3. In the RRR process, once ρ_H is decreased sufficiently, and the tile with new maximum EPD is different from t_k, a new iteration will start; otherwise, if ρ_H can not be decreased after trying all the two-pin nets identified in an iteration, this stage stops. In practice, to control the runtime, we monitor the improvement in ρ_H, and if its percentage reduction over the past N_{stop} RRR iterations is smaller than ϵ_{stop}, we stop this stage. We calibrate reasonable values of these parameters as $N_{stop} = 100$ and $\epsilon_{stop} = 1\%$. In an iteration, after rerouting a net, if ρ_H or total overflow becomes larger than that before ripping up the net, its original routing path is restored.

In rare instances, there may be more than one tiles with EPD equal to ρ_H. In this case, we just first process the tile found first in the algorithm. Then after a couple of RRR operations, it is likely that the EPD of this tile becomes smaller than ρ_H, and next we process another tile with EPD equal to ρ_H. In other words, the tie is broken arbitrarily, and then the other tile(s) with the same EPD will be considered soon in a subsequent iteration.

[2]In other stages, we do not adopt this measure to reduce ρ_H, but integrate the dummy fill cost function into the router to minimize ρ_H, because routability has the highest priority at those stages.

4.5 Cost function

The success of the routing framework is critically dependent on the choice of a cost function. It is vitally important for the cost function to be computationally easy to evaluate, and yet hold enough fidelity to capture a more complex underlying objective that it represents. Therefore, we elaborate efficient and effective cost functions to achieve our goal.

To address the new objective of minimizing D, we define Φ_e as the dummy fill cost of edge e, which will be integrated to the original cost function used in NTHR. The remainder of this section will first discuss the computation of Φ_e, and then describe how it is used at various stages of the routing process.

4.5.1 Finding a surrogate for the dummy fill cost

Our cost function requires the computation of Φ_e, which is related to the dummy fill metric, D. The direct calculation of D is highly computational [44], and therefore, we have to find a metric which correlates well with D and is easy to use in the routing process.

As a first step, we present a metric Γ that is linearly related to D:

$$\Gamma = \sum_{t_{ij} \in Q_1} (\rho_U - \rho_{ij}), \tag{4.9}$$

where $\rho_U = \rho_H - \epsilon$, which is the lower bound on the EPD after a dummy fill procedure achieves the constraint described in (4.4), and Q_1 is the set of tiles for which $\rho_{ij} < \rho_U$ before inserting dummy fill. For these tiles, ρ_{ij} must be increased by inserting dummy fill. Note that unlike $\mathrm{ran}(\rho)$, which merely captures the difference between the maximum and the minimum values of ρ, but is insensitive to the distribution of ρ within this range, the metric Γ captures the distribution of ρ over all the related tiles.

We will now present results that link Γ to D.

Lemma 4.1. *Given a weighting function $f(i,j)$ whose value is truncated to 0 outside the weighting window,*

$$\sum_{\text{all } t_{ij}} \rho_{ij} = b \sum_{\text{all } t_{ij}} d_{ij}, \text{where } b = \sum_{i=-l}^{l} \sum_{j=-l}^{l} f(i,j).$$

Proof. From (4.3), we have

$$\begin{aligned}\sum_{\text{all } t_{ij}} \rho_{ij} &= \sum_{\text{all } t_{ij}} \left(\sum_{n_1=-l}^{l} \sum_{n_2=-l}^{l} \left(d_{i+n_1,j+n_2} f(n_1,n_2) \right) \right) \\ &= f(-l,-l) \sum_{\text{all } t_{ij}} d_{i-l,j-l} + \cdots + f(l,l) \sum_{\text{all } t_{ij}} d_{i+l,j+l}.\end{aligned}$$

For circular convolution, $\sum_{\text{all } t_{ij}} d_{i+p,j+q} = \sum_{\text{all } t_{ij}} d_{ij}, \forall p,q \in \mathbb{Z}$. Therefore, $\sum_{\text{all } t_{ij}} \rho_{ij} = f(-l,-l) \sum_{\text{all } t_{ij}} d_{ij} + f(-l,-l+1) \sum_{\text{all } t_{ij}} d_{ij} + \cdots + f(l,l) \sum_{\text{all } t_{ij}} d_{ij} = b \sum_{\text{all } t_{ij}} d_{ij}$. □

The above lemma may be used next to show a key result that drives our approach.

Theorem 4.1. *Let Q_0 be the set of the tiles with EPD no less than ρ_U before dummy filling, and Q_1 be the complement of Q_0. Let d'_{ij} (ρ'_{ij}) be the IPD (EPD) of t_{ij} after dummy filling, and let*

$$\begin{aligned}\mu &= \sum_{t_{ij} \in Q_0} (\rho'_{ij} - \rho_{ij}), \\ \nu &= \sum_{t_{ij} \in Q_1} (\rho'_{ij} - \rho_U).\end{aligned}$$

Then

$$D = c\left(\Gamma + \mu + \nu\right), \tag{4.10}$$

where $c = 1/b > 0$, and b is defined in Lemma 4.1.

Proof. By definition, $D = \sum_{t_{ij} \in Q} (d'_{ij} - d_{ij})$, where $Q = Q_0 \cup Q_1$. By Lemma 4.1 and considering $Q_0 \cap Q_1 = \phi$,

$$\begin{aligned}D &= \frac{1}{b} \sum_{t_{ij} \in Q} (\rho'_{ij} - \rho_{ij}) \\ &= \frac{1}{b} \left(\sum_{t_{ij} \in Q_1} (\rho'_{ij} - \rho_{ij}) + \sum_{t_{ij} \in Q_0} (\rho'_{ij} - \rho_{ij}) \right) \\ &= \frac{1}{b} \left(\sum_{t_{ij} \in Q_1} (\rho_U - \rho_{ij}) + \sum_{t_{ij} \in Q_1} (\rho'_{ij} - \rho_U) + \mu \right) \\ &= c\left(\Gamma + \mu + \nu\right).\end{aligned}$$

□

Note that only the tiles in set Q_1 require the insertion of dummy fill. In practice, since the minimal amount of dummy fill is inserted, it can be expected that after dummy fill insertion, ρ'_{ij} of $t_{ij} \in Q_0$ including the tile with ρ_H will remain unchanged or, at worst, increase by a very small amount. Therefore, μ is a small number compared to Γ. Moreover, ρ'_{ij} for $t_{ij} \in Q_1$ will be approximately equal to or just a little larger than ρ_U, and therefore ν is also a small number compared to Γ. In the following analysis, we will always assume that μ and ν are negligible compared to Γ. These *dummy fill assumptions* imply that

$$D \approx c\Gamma. \tag{4.11}$$

This relationship will be demonstrated experimentally in Section 4.6. Since $c = 1/b$, and is a positive constant for a given weighting function, to minimize D, we should minimize Γ.

Though Γ is much easier to compute than D, it is still too complex to be used directly in routing process, since it requires an enumeration over all tiles in Q_1; as we will see, the cardinality of this set can be very large. In order to find another simpler metric which can be used in routing, next we will analyze the impact of our routing procedure on Γ, and then D.

In the routing process, every two-pin net passing through overflowing edges or passing through the weighting windows of the tile(s) with ρ_H is ripped up and then rerouted one by one in some ordering. Next we will present two theorems to reveal the impact on Γ of ripping up and rerouting a single two-pin net: the first theorem is for rerouting a two-pin net, and the second one for ripping up a two-pin net.

Theorem 4.2. *Consider a partial routing solution $\mathcal{S}$, and the solution $\bar{\mathcal{S}}$ after routing one more net, and consider the contribution of each edge e of this net. Let $\Delta\Gamma$ and $\Delta\rho_H$ be, respectively, the change in Γ and ρ_H from $\mathcal{S}$ to $\bar{\mathcal{S}}$ due to edge e.*
(a) The following inequality holds:

$$\Delta\Gamma \geq |Q_1| \cdot \Delta\rho_H - b \cdot \Delta_d^M, \tag{4.12}$$

where Δ_d^M is the increase in the IPD of a tile when routing through one edge in the tile, and is equal to the ratio of the area occupied by a wire track to that of a tile.
(b) If $\Delta\rho_H = 0$, $\bar{Q}_1 = Q_1$ and $W_e \subseteq Q_1$, then

$$\Delta\Gamma = -b \cdot \Delta_d^M, \tag{4.13}$$

where $\bar{Q}_1$ is the new Q_1 after routing through e, and W_e is the weighting window of the tile associated with e.

Proof. We first prove part (a). Suppose that the changes from $\mathcal{S}$ to $\bar{\mathcal{S}}$ due to routing through e are: ρ_{ij} changes to $\bar{\rho}_{ij}$, ρ_H changes to $\bar{\rho}_H$, ρ_U changes to $\bar{\rho}_U$, Q_1 changes to $\bar{Q}_1$, Q_0 changes to $\bar{Q}_0$, and Γ changes to $\bar{\Gamma}$. Then $\Delta\rho_H = \bar{\rho}_H - \rho_H$. Assume ϵ for the two routing solutions is the same. Then $\bar{\rho}_U = \rho_U + \Delta\rho_H$, $\bar{\Gamma} = \sum_{t_{ij}\in\bar{Q}_1}(\bar{\rho}_U - \bar{\rho}_{ij})$. Then $\Delta\Gamma = \bar{\Gamma} - \Gamma$.

Let $Q_1^+ = \{t_{ij}|t_{ij} \notin Q_1 \text{ and } t_{ij} \in \bar{Q}_1\}$, $Q_1^- = \{t_{ij}|t_{ij} \in Q_1 \text{ and } t_{ij} \notin \bar{Q}_1\}$. Then $\bar{Q}_1 = Q_1 + Q_1^+ - Q_1^-$. Then

$$\begin{aligned}\bar{\Gamma} &= \sum_{t_{ij}\in\bar{Q}_1}(\bar{\rho}_U - \bar{\rho}_{ij}) \\ &= \sum_{t_{ij}\in Q_1}(\bar{\rho}_U - \bar{\rho}_{ij}) + \sum_{t_{ij}\in Q_1^+}(\bar{\rho}_U - \bar{\rho}_{ij}) - \sum_{t_{ij}\in Q_1^-}(\bar{\rho}_U - \bar{\rho}_{ij}).\end{aligned}$$

By the definition of Q_1^-, we can know that $\bar{\rho}_{ij} \geq \bar{\rho}_U, \forall\, t_{ij} \in Q_1^-$, and then

$$\sum_{t_{ij}\in Q_1^-}(\bar{\rho}_U - \bar{\rho}_{ij}) \leq 0.$$

Similarly, by the definition of Q_1^+, $\sum_{t_{ij}\in Q_1^+}(\bar{\rho}_U - \bar{\rho}_{ij}) \geq 0$. Note that Q_1^+ can be empty if ρ_{ij} of tile $t_{ij}, \forall\, t_{ij} \in Q_0$, is no less than $\bar{\rho}_U$. Then

$$\begin{aligned}\bar{\Gamma} &\geq \sum_{t_{ij}\in Q_1}(\bar{\rho}_U - \bar{\rho}_{ij}), \\ \Delta\Gamma = \bar{\Gamma} - \Gamma &\geq \sum_{t_{ij}\in Q_1}(\bar{\rho}_U - \bar{\rho}_{ij}) - \sum_{t_{ij}\in Q_1}(\rho_U - \rho_{ij}) \\ &= \sum_{t_{ij}\in Q_1}\Delta\rho_H - \sum_{t_{ij}\in Q_1}(\bar{\rho}_{ij} - \rho_{ij}).\end{aligned}$$

Let t_e be the tile associated with edge e and W_e be the weighting window of t_e. Note that only ρ_{ij} of tile $t_{ij} \in W_e$ can increase after routing through edge e. Let $W_1 = Q_1 \cap W_e$. Since $W_1 \subseteq W_e$ and $\bar{\rho}_{ij} \geq \rho_{ij}$, we have

$$\begin{aligned}\Delta\Gamma &\geq \sum_{t_{ij}\in Q_1}\Delta\rho_H - \sum_{t_{ij}\in W_1}(\bar{\rho}_{ij} - \rho_{ij}) \\ &\geq \sum_{t_{ij}\in Q_1}\Delta\rho_H - \sum_{t_{ij}\in W_e}(\bar{\rho}_{ij} - \rho_{ij}).\end{aligned} \tag{4.14}$$

Since the increase of ρ_{ij} of $t_{ij} \in W_e$ is due to the increase in the IPD of t_e by Δ_d^M, we have

$$\sum_{t_{ij}\in W_e}(\bar{\rho}_{ij} - \rho_{ij}) = \Delta_d^M \sum_{i=-l}^{l}\sum_{j=-l}^{l} f(i,j) = b\Delta_d^M. \tag{4.15}$$

By (4.14) and (4.15), we have $\Delta\Gamma \geq |Q_1|\Delta\rho_H - b\Delta_d^M$.

Note that the conclusion holds when $\Delta\rho_H \geq 0$, which is easy to see from the proof itself.

For part (b), we use the same symbols defined above. Using the conditions $\Delta\rho_H = 0$, $\bar{Q}_1 = Q_1$ and $W_e \subseteq Q_1$, similar to the proof above, we have:

$$\Delta\Gamma = -\sum_{t_{ij}\in Q_1}(\bar{\rho}_{ij} - \rho_{ij}) = -\sum_{t_{ij}\in W_e}(\bar{\rho}_{ij} - \rho_{ij}) = -b\Delta_d^M. \qquad \square$$

It is important to point out that Equation (4.12) holds regardless of whether $\Delta\rho_H > 0$ or $\Delta\rho_H = 0$.

Corollary 4.1. *Consider a partial routing solution $\bar{\mathcal{S}}$, and the solution $\mathcal{S}$ after ripping up one more net, and consider the contribution of each edge e of this net. Let $\Delta\Gamma$ and $\Delta\rho_H$ be, respectively, the change in Γ and ρ_H from $\bar{\mathcal{S}}$ to $\mathcal{S}$ due to edge e. Then*

$$\Delta\Gamma \leq |Q_1| \cdot \Delta\rho_H + b \cdot \Delta_d^M, \tag{4.16}$$

where Δ_d^M has the same meaning as in Theorem 4.2.

The proof is simple. Ripping up a net N from $\bar{\mathcal{S}}$ is a symmetric process of starting from S and routing net N with the same routing path. In this case, the change in Γ (ρ_H) from S to $\bar{\mathcal{S}}$ is $-\Delta\Gamma$ ($-\Delta\rho_H$), using the terminology defined in the statement of Theorem 4.2. Therefore, $-\Delta\Gamma \geq |Q_1| \cdot (-\Delta\rho_H) - b \cdot \Delta_d^M$. This corollary holds when $\Delta\rho_H \leq 0$. $\square$

Table 4.1: The cardinality of set Q_1 for ISPD07 circuits

Circuits	Horizontal layer		Vertical layer	
	#tiles	$\lvert Q_1\rvert$	#tiles	$\lvert Q_1\rvert$
adaptec1	6561	4566	6561	4542
adaptec2	11236	10557	11236	10435
adaptec3	24180	21356	24180	20874
adaptec4	24180	22925	24180	22469
adaptec5	24180	18119	24180	19181
newblue1	6400	5531	6400	5766
newblue2	28830	24619	28830	23964
Average	1.00	0.85	1.00	0.85

In practice, $|Q_1|$, the cardinality of set Q_1, is seen to be very large. Table 4.1 shows the values of $|Q_1|$ for the routing solutions obtained by NTHR for the 2D ISPD07 benchmarks[3]. We can see that on average 85% of the tiles are in Q_1. The above theorems imply that if the cardinality of set Q_1 is large, then $\Delta\Gamma$ will be significant even for a small change in ρ_H, which implies a nontrivial change in D. In other words, D is sensitive to the change in ρ_H due to the large cardinality of set Q_1.

Example: For the benchmark newblue2, $b = 0.532$ and $\Delta_d^M = 7.41 \times 10^{-4}$. We use $|Q_1| = 23964$, the value for the vertical layer. Now suppose that we are routing a single wire through an edge associated with a tile with EPD equal to ρ_H, and this causes a small increase[4] in ρ_H: here, $\Delta\rho_H = 2.36 \times 10^{-6}$ for newblue2. By computing the lower bound for $\Delta\Gamma$ using (4.12), and using this to predict a lower bound on ΔD using (4.11), it can be shown that the increase in D is equivalent to at least 142 tracks. By Corollary 4.1, if $\Delta\rho_H$ is reduced by the same amount in the above example, D will reduce by at least 142 tracks. This example shows clearly that even though ρ_H changes by a very small amount, D will change greatly due to the large cardinality of Q_1. □

The analysis above shows that D is highly sensitive to the change in ρ_H. In other words, minimizing ρ_H is a good surrogate for minimizing D. Therefore, our routing objective is to minimize the ρ_H after routing finishes. We do this by trying to leave ρ_H unchanged, or minimizing the increase in its value, when routing each net, and by trying to reduce ρ_H when ripping up a net in the EPD postprocessing stage. Intuitively, this objective tries to ensure that all ρ values are low and as well-balanced as possible.

4.5.2 Dummy fill cost function

In NTHR, monotonic routing and maze routing are used in the RRR process. When rerouting a net, the costs of edges in the searching region are used to guide the router to find a new path for the net. In order to optimize D in the RRR process, we compute the dummy fill cost for every edge e, denoted as Φ_e, and then integrate Φ_e to the router.

Based on the results in Section 4.5.1, we now show how we compute the dummy fill cost, Φ_e. Our approach is based on the previous analysis, which shows that minimizing ρ_H in routing

[3]The characteristics of benchmarks used are listed in Table 4.2.

[4]As a reference, the values of ρ_H for the circuits in our benchmarks are between 0.10 and 0.20 for both the horizontal and the vertical layers.

process is a good surrogate for minimizing D. To capture this objective well, the following three aspects should be considered in the dummy fill cost function.

One possible component of the cost function could be to capture the effect of routing through an edge e on the increase in ρ_H. In the case that routing through an edge e will cause an increase in ρ_H, a large cost should be assigned to e as a penalty. To achieve this, the following term could be used:

$$\Omega_e = \begin{cases} \exp\left(\frac{\Delta\rho_H}{\Delta_\rho^M}\right) & \text{if } \Delta\rho_H > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{4.17}$$

where $\Delta\rho_H$ is the change in ρ_H after routing through edge e, and Δ_ρ^M is the possible maximal increase in the EPD of any tile and equal to the increase in the EPD of the tile associated with edge e, when a wire is routed through edge e. Here the role of Δ_ρ^M is to normalize the numerator to a value between 0 and 1. Since Ω_e captures the direct increase of ρ_H, the exponential function is used to magnify the penalty.

However, as we will soon see, such a function is not general enough since it only considers the contribution of a single edge, rather than that of a path. In particular, it is possible that the cost of each single edge e_i on path P is 0 by (4.17), since no single edge increases the value of ρ_H; however, routing through path P may still increase ρ_H due to the cumulative effects of all edges on the path, due to which the dummy fill cost of path P should not be 0.

For example, consider a tile t_k with large EPD which is near a path P, and let W_k be the weighting window of tile t_k, as shown in Fig. 4.3. In this example, due to congestion, the net $S \to T$ is detoured and path P shown in the figure is chosen as a candidate. The edges of P that lie along wire segments AB and CD are within W_k, and together, these may increase ρ_k by a significant amount. Before routing this net, if ρ_k was smaller than $\rho_H - \Delta_\rho^M$ but still close to ρ_H, then after routing through P, it is likely that ρ_k may exceed ρ_H. However, Equation (4.17) may not capture this effect, since each edge individually may have zero cost according to this function, and therefore we have to build a new cost component to address this problem.

Let S_P denote the set of all paths that may be used to route the net that is currently under consideration. Let J_P be the increase in the EPD of a tile t_k, if route P is chosen, and $J_M = \max_{S_P} J_P$. Then for tile t_k, if $\rho_k < \rho_H - J_M$, then ρ_k cannot exceed ρ_H after routing through any path; otherwise, ρ_k may become larger than ρ_H, depending on which path P is chosen. To deter the router from choosing a path that creates this violation, we assign a large penalty to the edges within W_k to deter the router from choosing a path that contains edges in W_k. From

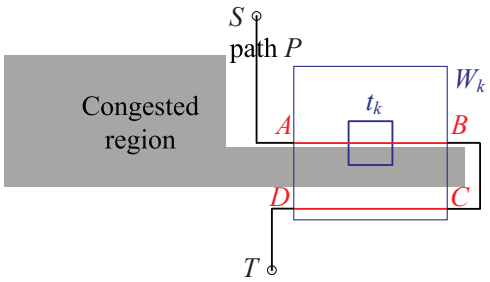


Figure 4.3: Routing through path P may increase the EPD of tile t_k significantly.

the point of view of the cost associated with an edge e, if there is a tile t_k within W_e with $\rho_k \geq \rho_H - J_M$, a large penalty should be assigned to e and the penalty should increase for higher ρ_k, where W_e is the weighting window of the tile associated with e. We define

$$\Delta\rho'_H = \max_{t_k \in W_e} (\rho_k + J_M - \rho_H), \tag{4.18}$$

where the role of the max function is to determine the largest possible increase in ρ_H. Then we use the following as the first component of the dummy fill cost for e:

$$\Theta_e = \begin{cases} \exp\left(\frac{\Delta\rho_k}{\Delta_\rho^M}\right) \cdot \exp\left(p_0 \cdot \frac{\Delta\rho'_H}{\Delta_\rho^M}\right) & \text{if } \Delta\rho'_H > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{4.19}$$

where $\Delta\rho_k$ is the increase of ρ_k after routing through edge e, and $\Delta\rho'_H$ is the adapted version of $\Delta\rho_H$, and p_0 is a user-defined parameter to control Θ_e in a proper range. In our experiments, p_0 is tuned on circuit newblue2 to make $\max(\Theta_e) = 16.3$. Here, the first exponential term is used to capture the fact that the larger $\Delta\rho_k$ is, the larger is the possibility that ρ_k will exceed ρ_H. The term Δ_ρ^M is used to normalize the numerators.

The computation of J_M discussed above requires the determination of a realistic upper bound of the increase in the EPD of tile t_k over all routes for the net being considered. The most pessimistic estimate assumes that when path P passes through all the edges in W_k, J_P reaches its maximum. However, this case is excessively pessimistic and extremely rare even in a very congested design, since practical routes do not go through so many bends and detours within a small region. A reasonable practical upper bound for J_M corresponds to the case shown in Fig. 4.3, where a "U"-shaped path is used to detour and passes through the weighting window of

t_k twice. In this case, J_M can be calculated as:

$$J_M = \Delta_d^M \sum_{i=-l}^{l} \sum_{j=0}^{1} f(i,j), \tag{4.20}$$

where Δ_d^M is the same as in Theorem 4.2, l the same as in (4.3), and $f(i,j)$ is the weighting function. To be safer still, our implementation uses twice the calculated value of J_M in (4.20) as the guardband.

A second component of the dummy fill cost function can be determined as follows. For the tiles whose EPDs are close to ρ_H, routing through the edges in these tiles will increase their EPDs further and it is likely that their EPDs will exceed the current ρ_H soon in the routing process. To control this trend, we assign large cost penalties to the edges associated with such tiles. On the other hand, we do not want the CMP optimization to affect the routing with normal objectives such as wirelength and overflow too much, and therefore, for the tiles with EPD values not large enough to cause the increase of ρ_H, their costs should be small. To achieve this goal, we adapt the cost function (4.5) in [27] to use in our work, since it increases very slow when the variable is not close to a limit. We remove the constant factor 1 in the cost function in [27] to make its low bound to be 0, and normalize the EPD value of a tile by the current ρ_H. Then the cost of an edge e is:

$$\Psi_e = \frac{p_1}{1 + e^{p_2(1-\rho_e/\rho_H)}}, \tag{4.21}$$

where ρ_e is the EPD of the tile associated with edge e, and p_1 and p_2 are user-defined parameters. In this work, we choose $p_1 = 4.0$ and $p_2 = 11.0$, which makes $\Psi_e = 1$ (the cost equal to that of wirelength) when $\rho_e = 0.9 \times \rho_H$, and makes $\Psi_e = 2$ when $\rho_e = \rho_H$. Here, $\Psi_e = 2$ is the maximum of Ψ_e, which is double of the cost of wirelength. On the other hand, when ρ_e is not so close to ρ_H, Ψ_e will be a relatively small number compared with wirelength cost. For example, when $\rho_e = 0.7 \times \rho_H$, $\Psi_e = 0.14$, which is rather small and will not affect the routing too much.

The total dummy fill cost of edge e is defined as follows:

$$\Phi_e = \Theta_e + \Psi_e. \tag{4.22}$$

Here, Ω_e is not used since it is covered by Θ_e. Furthermore, note that the parameter p_0 in (4.19) will modulate Θ_e to adjust the ratio of Θ_e to Ψ_e and to determine the maximal value of Φ_e. Though our cost function is essentially heuristic, we expect it to work well due to the following two factors. First, it considers not only the impact of routing through one tile/edge on its own

EPD (by Ψ_e), but also the impact on the neighboring tiles (by Θ_e), which captures the long range effect of CMP variation. Second, our cost function is consistent with the previous theoretical judgment: it always tries to control and avoid the increase of ρ_H, which is desirable for the minimization of D.

4.5.3 Cost function in different stages

In this section, we will introduce how the proposed cost functions are integrated into the routing framework.

First, we point out that we do not integrate the dummy fill cost functions to every stage of NTHR. In NTHR, the main purpose of the initial stage is to obtain the initial congestion map and the initial routing solution for later use, and the space for optimizing all the objectives together is limited since L-pattern routing is used. Furthermore, from our empirical observation, the addition of dummy fill optimization in the initial stage does not improve the final solution but increases the runtime of later stages. The principle here is that the initial stage is very useful in controlling conventional routing metrics, and dummy fill can be effectively optimized in the later stages of NTHR. Therefore, we do not add a dummy fill optimization objective in the initial stage. We also do not perform dummy fill optimization in the layer assignment stage in our current work, since we use the single-layer CMP model in this work as a first step to dummy fill optimization, and leave multi-layer dummy fill optimization as future works.

In the main stage, we consider the dummy fill optimization by integrating the proposed cost function with traditional cost function of NTHR. In the main stage, the cost of edge e becomes as follows:

$$\mathcal{C}_e = \mathcal{C}_e'' + \gamma_1 \times \Phi_e \times \max(B_e, 0.1), \tag{4.23}$$

where $\mathcal{C}_e''$ is the original cost function (4.6) in NTHR, Φ_e is the dummy fill cost function (4.22), B_e is a scaling factor defined in (4.7) and γ_1 is the user-defined weight for Φ_e. In our implementation, γ_1 is tuned on circuit newblue2 to be 2.5. The mechanism of using term $\max(B_e, 0.1)$ is similar to that of using B_e in the original cost function (4.6) in NTHR: when the number of RRR iterations increases, which means it is more and more difficult to route the current net, B_e will decrease from 1 gradually towards 0, and then dwarf all the cost function components but the congestion cost, which helps the router to obtain a path without overflow. However, to prevent ρ_H from increasing by a large amount when B_e becomes too small, we use 0.1 as the lower

bound of dwarfing dummy fill cost. From our empirical observation, the value of 0.1 achieves a good balance between routability and dummy fill optimization.

In the refinement stage, most nets have been routed without overflow in the previous stages, and only a few nets must be ripped up and rerouted due to overflow. Since the primary goal is to reduce the overflow with limited routing resources, high priority should be given to reduction of overflow, and thus the following cost function is used:

$$\mathcal{C}_e = \gamma_2 \times \mathcal{C}_e''' + \Phi_e, \tag{4.24}$$

where $\mathcal{C}_e'''$ is the original cost (4.8) used in NTHR, Φ_e is the dummy fill cost, and γ_2 is an empirically chosen weight whose value should be large enough to make $\mathcal{C}_e'''$ dominant to give a high priority to the reduction of overflow. We choose $\gamma_2 = M_P \cdot \max(\Phi_e)$, where M_P is an upper bound on the length of the longest possible path for a two-pin net in the layout. In this way, a path with smallest overflow will always be chosen by the router. When there are several candidate paths with zero overflow, the dummy fill cost will guide the router to choose the best one out of them. In our implementation, we choose $M_P = N_g$ when determining γ_2, where N_g is the total number of grids in the layout: this is a realistic estimate of the upper bound. In estimating $\max(\Phi_e)$, we note that $\max(\Psi_e) = 2$ (from (4.21)), and that the value of $\max(\Theta_e) = 16.3$; this is based on empirical tuning on circuit newblue2, as will be explained in Section 4.6. Therefore, we set $\max(\Phi_e) = 18.3$.

In the EPD postprocessing stage that we introduce, our goal is to reduce the maximal EPD further but not to increase overflow, and therefore the cost function used is the same as (4.24).

4.5.4 Why optimizing ρ_L is not important

As seen in (4.4), the range of the EPD (and hence the range of CMP variation) can be reduced by either reducing ρ_H or by increasing ρ_L. However, our arguments above primarily focus on reducing ρ_H in order to reduce D. A natural question to ask is whether it would also be useful to make efforts to increase the value of ρ_L in order to reduce D. Here, "make efforts" means taking measures similar to what we have done for minimizing ρ_H, e.g., a bonus is given to an edge or a path when routing through it will increase ρ_L.

Given a layout, let us consider the change of D after routing through an edge e. We consider two cases for routing e:

- If $\Delta\rho_H > 0$, then as shown by the argument and example after Theorem 4.2, D could increase by a large amount due to the large cardinality of set Q_1, so we elaborate cost functions to avoid using edge e on the routing path.

- In contrast, if $\Delta\rho_H = 0$, then as revealed by Theorem 4.2, $\Delta\Gamma \geq -b\Delta_d^M$, i.e., the largest possible reduction in Γ is $b\Delta_d^M$. Furthermore, by Theorem 4.2, if the three conditions $\Delta\rho_H = 0$, $\bar{Q}_1 = Q_1$ and $W_e \subseteq Q_1$ are satisfied, then $\Delta\Gamma = -b\Delta_d^M$. In other words, routing through any edge which satisfies these three conditions will achieve the *same* maximum reduction $b\Delta_d^M$ on Γ, *no matter how* ρ_L *changes*. Therefore, there is no reason we should make extra efforts to route through a few special edges, that are in the weighting window of the tile(s) with EPD equal to ρ_L, to increase ρ_L. By (4.11), the above analysis also holds for D, so we do not need to care about how ρ_L changes in terms of minimizing D.

The argument for routing through a path is similar to the analysis above: in order to obtain the same change in D, we do not need to make extra efforts to let the path pass through a few special edges to increase ρ_L, because there are many other choices of edges to pass through, which have the same or similar effect on D. Based on the analysis above, in our algorithm, we focus our efforts to minimize ρ_H but do not make efforts to increase ρ_L.

4.6 Experimental results

We have implemented the algorithm in C++, and have tested it on a 64-bit Linux machine with an Intel® Core™ 2 Duo 3.00 GHz CPU and 8 GB memory. The routing algorithm is implemented as a program with two options: MaxEPD optimizes the dummy fill based on minimizing the maximum EPD, and NoCMP does not consider dummy fill.

We have tested our routing scheme on the 2D ISPD07 benchmarks [71, 72], whose characteristics are listed in Table 4.2. In the table, the column "guardband" lists the percentage of the given capacity to the actual capacity after the guardband adjustment. The unit for grid size is the number of wire tracks. Since newblue3 is unroutable using current routers [73], it is not considered in our experiments. We assume 0.13 um technology is used for the circuits, and the wire width equal to 5 times of the minimal value, 0.65 um. We use a typical planarization length, $L = 1$ mm. The size of a CMP tile (which is used to evaluate IPD and EPD) is set to about

100 um to obtain a good balance between accuracy and performance[5]. As a result, the weight window size for CMP will be around 121 CMP tiles.

Table 4.2: Benchmark information

Circuits	Grid dimension	Grid size	Guard-band	#Nets ($\times 10^3$)	Grid HPWL ($\times 10^5$)
adaptec1	324 × 324	35	90	176	30.00
adaptec2	424 × 424	35	100	208	28.82
adaptec3	774 × 779	30	90	368	86.20
adaptec4	774 × 779	30	90	401	81.75
adaptec5	465 × 468	50	100	548	88.97
newblue1	399 × 399	30	90	271	20.80
newblue2	557 × 463	50	100	374	41.91

To demonstrate the effectiveness of our cost functions, we compare MaxEPD with the method proposed in [68], which attempts to minimize the CMP variation using maze routing under a CMP-aware cost function, in which the cost of an edge is the EPD of the associated CMP tile. Since the codes in [68] are not available for public access, in order to compare with this method, we developed another router by replacing the cost function we propose in MaxEPD by the cost function in [68][6]. We denote this router as "Yet another CMP-aware router" (YaCMP).

For MaxEPD, we tune the weights p_0 in (4.19) and γ_1 in (4.23) with circuit newblue2[7], and then use these weights for all the circuits. Specifically, we use $\gamma_1 = 2.5$, and p_0 is tuned so that $\max(\Theta_e) = 16.3$. These weights determine the tradeoff between traditional routing and dummy fill minimization, and therefore, they should be in an appropriate range. For completeness, we will list newblue2 in our tables of results, but the gains on this circuit can be appropriately discounted since the parameters were tuned on it. For YaCMP, we also tune the weights γ_1 in (4.23) for the circuit newblue2 in the similar way. Specifically, we use $\gamma_1 = 5.5$.

[5]In this case, a CMP tile used to compute IPD and EPD may contain more than one routing grids. This explains why the number of tiles shown in Table 4.1 is much smaller than that of grids shown in Table 4.2.

[6]The cost function used in the global routing stage in [69] is in fact the same as that used in [68].

[7]Circuit newblue2 is chosen to tune the weights due to its medium size and least runtime among all the circuits. Generally, tuning weights costs tens of times the runtime of a single run. Also, note that p_1, p_2 and γ_2 take the pre-defined values and are not required to be tuned.

Table 4.3: Comparison of routing results among NoCMP, YaCMP and MaxEPD.

Circuits	Total overflow			Wirelength ($\times 10^5$)			$\mathrm{ran}(z)$ (Å)			Runtime (s)		
	NoCMP	YaCMP	MaxEPD	NoCMP	YaCMP	MaxEPD	NoCMP	YaCMP	MaxEPD	NoCMP	YaCMP	MaxEPD
adaptec1	0	0	0	42.51	44.33	45.41	2145	2038	1817	285	998	2784
adaptec2	0	0	0	40.01	40.48	41.19	2622	2557	2390	61	113	265
adaptec3	0	0	0	108.95	112.97	114.08	2251	2169	1875	305	1080	1229
adaptec4	0	0	0	102.34	102.76	103.13	2138	1950	1806	71	132	195
adaptec5	0	0	0	121.13	122.25	127.15	2628	2582	2522	668	1215	3371
newblue1	3	16	6	32.42	32.65	32.87	2337	2262	2230	227	683	652
newblue2	0	0	0	57.13	57.59	57.89	1839	1717	1603	35	89	132
Summary	3	16	6	1.000	1.017	1.033	1.000	0.955	0.889	1.000	2.592	4.658

Table 4.3 compares the routing results from NoCMP, YaCMP and MaxEPD. The last line "Summary" presents a synopsis of the comparison among the three methods. For each circuit, we show the total overflow, wirelength, as well as the unevenness in the topography after CMP. In the table, "$\mathrm{ran}(z)$" stands for the range/variation of the ILD thickness. The value of $\mathrm{ran}(z) = z_1 \cdot \mathrm{ran}(\rho)$ is calculated according to (4.1), suppose z_1 = 7000 Å [44]. Note that each via is counted as 1 unit of wirelength in the calculation of the total wirelength, according to the rule in ISPD 2008 global routing contest [73]. For all the circuits except newblue1, the maximal overflow is 0. For newblue1, the maximal overflows for the routing solutions obtained by NoCMP, YaCMP and MaxEPD are all equal to 1. Since the guardband factor for newblue1 is 90%, the one or two overflows on an edge can be eliminated in later stage with the reserved capacity, and then these overflows will not cause any problems to the EPD computation and dummy filling.

It can be seen that MaxEPD consistently provides significant improvements in the ILD variations, at the cost of a small increase in the wirelength and overflow (only for the circuit newblue1). Compared with NoCMP, MaxEPD improves the post-CMP ILD variations by 11.1% on average and up to 16.7%; compared with YaCMP, the improvement is 7.0% on average and up to 13.5%.

The runtime of MaxEPD is acceptable, even for the large circuit adaptec5. The ratio of runtime of MaxEPD to that of NoCMP for circuit adaptec1 is the largest among all the circuits. This is likely because the weights which are tuned for circuit newblue2 are not appropriate for circuit adaptec1. Note that it is difficult to route circuit newblue1 without overflow [73], and therefore there is small space to optimize CMP variation. As a result, the improvement in the CMP variation, for both MaxEPD and YaCMP, is small on newblue1.

As a verification step, Table 4.4 shows an evaluation of the quality of our results, using the ranged-variation linear programming (LP) formulation of the dummy fill algorithm in [44]. We set $\epsilon = 0.02$ in (4.4). A commercial LP solver, ILOG CPLEX [74], is used to solve the dummy filling problem on a 64-bit Linux machine with a 2.6 GHz dual-core AMD Opteron™ 2218 processor and 2 GB memory. In the table, "fillWL/minWL" presents the ratio of the equivalent wirelength of total dummy fill to the minimal total wirelength (minWL). The value of minWL is the sum of the minimal wirelength without detours, ignoring congestion constraints. Compared with NoCMP, the MaxEPD approach significantly reduces the total fill by 22.0% on average and up to 41.5%; compared with YaCMP, MaxEPD reduces the fill amount by 14.1% on average and

Table 4.4: Comparison of dummy fill results among NoCMP, YaCMP and MaxEPD. The data in column 2 and 3 are normalized with the basis case (1.0) corresponding to NoCMP.

Circuits	fillWL/minWL		Fill time (s)
	YaCMP	MaxEPD	MaxEPD
adaptec1	0.831	0.585	224
adaptec2	0.956	0.852	495
adaptec3	0.917	0.700	1849
adaptec4	0.849	0.736	2083
adaptec5	0.950	0.873	1848
newblue1	0.944	0.920	206
newblue2	0.890	0.794	2775
Summary	0.905	0.780	

up to 23.6%. The data show the effectiveness of our routing algorithm, especially the proposed cost functions and the strategy of minimizing the maximal EPD. The last column of the table shows the runtime of the dummy fill algorithm of [44] for MaxEPD. The CPU time required by the dummy filling step, applied to the results of NoCMP and YaCMP, is similar and on average within 2% of that of MaxEPD.

Table 4.5: Comparison of IPD gradient $\mathcal{G}$ among NoCMP, YaCMP and MaxEPD

Circuits	$\mathcal{G}$ for horizontal layer			$\mathcal{G}$ for vertical layer		
	NoCMP	YaCMP	MaxEPD	NoCMP	YaCMP	MaxEPD
adaptec1	0.0216	0.0228	0.0236	0.0290	0.0281	0.0281
adaptec2	0.0218	0.0221	0.0215	0.0258	0.0264	0.0267
adaptec3	0.0182	0.0184	0.0184	0.0173	0.0179	0.0179
adaptec4	0.0201	0.0201	0.0202	0.0218	0.0219	0.0219
adaptec5	0.0239	0.0241	0.0234	0.0251	0.0251	0.0243
newblue1	0.0176	0.0183	0.0185	0.0199	0.0205	0.0205
newblue2	0.0147	0.0147	0.0147	0.0217	0.0223	0.0221
Summary	1.000	1.018	1.019	1.000	1.013	1.007

Next we experimentally support the claim in Section 4.1 that IPD gradient is not a good metric for CMP variations. As suggested in [67], IPD gradient of a tile t_{ij} is defined as $d_{ij} - \bar{d_{ij}}$, where $\bar{d_{ij}}$ is the average IPD of tiles adjacent to t_{ij} (including t_{ij}). Then we compute the quadratic mean of IPD gradients of all tiles as the IPD gradient of a layout:

$$\mathcal{G} = \sqrt{\frac{1}{|Q|}\sum_{t_{ij}\in Q}(d_{ij} - \bar{d_{ij}})^2},$$

where Q is the set of all the tiles in the layout. Table 4.5 shows the comparison of IPD gradient, $\mathcal{G}$, among NoCMP, YaCMP and MaxEPD. From the table, we can see that for most circuits, the layouts with the smallest IPD gradients do not have the smallest $\mathrm{ran}(z)$ and D. On average, the layouts obtained by NoCMP have the smallest IPD gradients but the largest $\mathrm{ran}(z)$ and D. In sum, it is clear that the IPD gradient is not a good indicator for $\mathrm{ran}(z)$ and D.

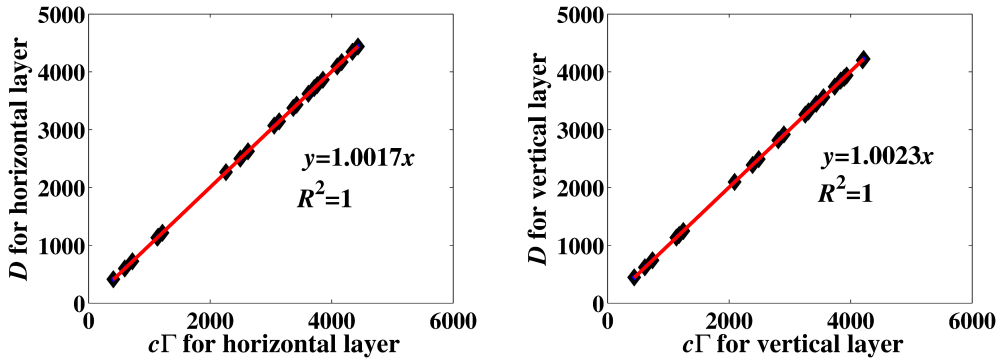


Figure 4.4: Linear fitting results for $c\Gamma$ vs. D.

Finally, we experimentally support the claim from Section 4.5.1 that (4.11) holds under the *dummy fill assumptions*. For all the dummy filling experiments we perform, we find that μ and ν are negligible compared to Γ. For example, for the dummy filling solution for the horizontal layer for circuit newblue2 with MaxEPD algorithm, $(\mu + \nu) : \Gamma = 1 : 483$. Fig. 4.4 shows the fitting results for dataset $(c\Gamma, D)$ obtained in all the dummy filling experiments, using a linear regression model, "$y = ax$", showing that $D \approx c\Gamma$ with $R^2 = 1$.

4.7 Conclusion

In this chapter, we develop a global routing algorithm optimizing the amount of dummy fill to be inserted. The accurate oxide CMP model in [5] is adopted to make our algorithm be aware of the real EPD distribution and be able to optimize the CMP variation more directly and effectively than previous ones. We propose a metric, Γ, that correlates well linearly with the amount of dummy fill in the routed layout. Then by deduction we find that minimizing the maximal EPD is a good surrogate for minimizing the total amount of dummy fill. Based on this, we set up effective cost functions to optimize the amount of dummy fill. The experimental results show that our algorithm can reduce the amount of dummy fill significantly. It is worth pointing out that our method uses a limited amount of empirical calibration and is built on a foundation of sound theoretical support. Therefore, it can easily be applied to other routers. Our work addresses the problem at the level of global routing, and can be applied in a flow that includes a CMP-conscious detailed router.

Chapter 5

Physical Design Techniques for Optimizing RTA-induced Variations

5.1 Introduction

As introduced in Chapter 1, variations in the semiconductor manufacturing process can cause significant yield loss and performance degradation, and tremendous effort is being expended in addressing these issues. Several manufacturing steps, such as CMP, have received substantial attention from the design community. A less well-studied, but increasingly important, issue is related to the effects of thermal annealing, a major contributor to on-chip variability [1].

RTA is widely used during manufacturing [21], and a critical point at which it impacts chip performance is when it is employed in contemporary ultra-shallow junction (USJ) technologies to activate the dopants after implantation [22]. RTA commonly involves the application of high temperature for a short period to perform dopant activation, while limiting unwanted dopant diffusion. Annealing largely determines the concentration of dopant and the junction depth, which in turn determines the threshold voltage, V_T, and also the sheet resistance of the doped regions such as the source, drain and the polysilicon gate. Since these parameters affect the performance of the circuit significantly, it is critically important to ensure that the annealing process occurs uniformly across the entire die.

The mechanism for RTA involves heating the top side of the wafer, where the active transistor devices are located, by exposure to an energy source such as an array of lamps to rapidly transfer radiative heat to the wafer surface (illustrated in Fig. 1.3(a)). One particular aspect of radiation

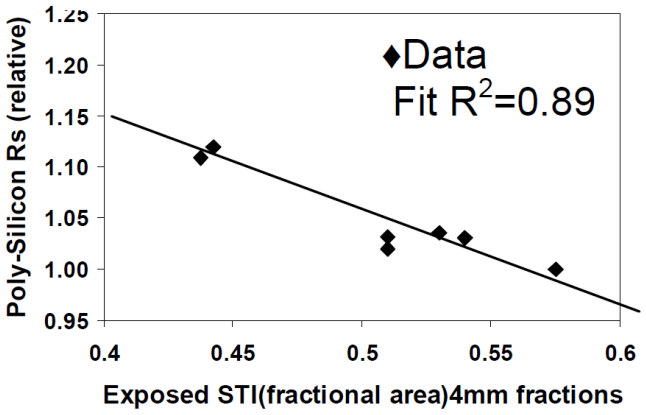


Figure 5.1: The variation in the polysilicon sheet resistance R_s correlates well with the exposed STI density averaged over 4 mm [1].

is that the reflectivity of the surface plays an important role on the amount of heat transferred. The principle is similar to the idea that on a sunny day, shiny or light-colored clothes reflect more light than dark clothes that tend to absorb heat. Across a die, due to differences in the reflectivities of materials and varying pattern densities, different locations will absorb different amount of heat, causing variations in the annealing temperature. This results in within-die and die-to-die variations in the circuit characteristics after RTA.

The resulting variability, often referred to as the pattern effect, has been studied in [1]. Their work investigates the impact of RTA on intra-die variations in the performance, subthreshold leakage, and other parameters, and demonstrates that most of the observed variations, including inverter delay changes, can be accounted for through RTA-driven variations in the transistor extrinsic resistance, R_{EXT}, and V_T. Moreover, it shows that the variations correlate with the calculated reflectivity for the lamp RTA spectrum, and hence depend on the millimeter-scale pattern density of the exposed STI regions, i.e., the STI not covered by the gate polysilicon. Fig. 5.1 illustrates how the variation in the polysilicon sheet resistance, R_s, correlates with the exposed STI density. These observations have been qualitatively confirmed by other groups (e.g., in [7]). The work in [75] performs thermal simulation and uses compact models to analyze within-die variability due to pattern-dependent RTA process. Their simulation shows about 20% variation in the leakage and 3% variation in the frequency due to the non-uniformity of the

pattern density in the layout. Therefore, it is necessary to optimize RTA-induced variations on a die.

A primary mechanism for reducing the large RTA variations is to achieve an even distribution of exposed STI throughout the layout. There are two ways to achieve this:

- *By rearranging blocks in the layout*: In system-on-chip designs, different circuit modules may have different STI densities, and the differences are especially significant if there are many memory and analog IP blocks. Variations in STI density can be inferred from Fig. 15 in [7] and Fig. 5 in [76]. Physical design plays an important role in achieving STI uniformity, and therefore, floorplanning or placement may be used to alter the STI density distribution to achieve uniform reflectivity.

- *By inserting dummy polysilicon fills*: As demonstrated in [7, 76], RTA variations can be mitigated if dummy fill structures are incorporated to improve the uniformity of the exposed STI. In [76], the dummy fill structure consists of dummy active area and dummy polysilicon gates, while in [7], the dummy fill structures are mainly dummy polysilicon. We assume the latter in our work. In particular, we attempt to minimize the inserted fills to minimize their side-effects, such as unwanted parasitics.

This work addresses both of these aspects. We first propose and formulate the RTA-variation-driven floorplanning problem[1], and present a floorplanner that optimizes the uniformity of the STI density, in addition to optimizing conventional objectives such as wirelength and area.

A significant issue to be addressed is whether RTA optimization should be addressed at the floorplanning level, or at more detailed levels. While methods at all levels of abstraction are appropriate, the greatest amount of flexibility is available at the floorplanning stage, albeit with the least detailed information. This is the classic early planning problem that is faced and surmounted in other aspects of physical design, and our solution is based on several facts. First, while the precise STI density of each block may not be known, most design houses perform extensive design reuse, or use similar design styles for similar blocks. Therefore, estimates of block STI densities are realistically available from historical data; in fact, past design data are already often used to predict Rent's parameters or interconnect density. Second, early stage planning for STI density can be made effective throughout the design flow, by propagating the

[1] Although it is possible and desirable to extend this work to RTA-driven placement, we focus on floorplanning in this work as a first step to RTA-driven physical design.

STI densities computed at this early planning stage as constraints at more detailed stages of design. Third, the accuracy and fidelity of the STI model that we use here are appropriate to this level of abstraction: more detailed analysis at finer levels of abstraction may require more detailed heat conduction models. Our experimental results confirm that the amount of improvement available at the floorplanning level is very large (on average, 39% in global variations and 29% in local variations).

In addition to floorplanning optimizations, we present the problem formulation of inserting dummy polysilicon fills that reduce the exposed STI area, thereby helping further improve the RTA variations by ensuring the uniformity in the exposed STI density. We show that a LP (linear programming) method, similar to that in [44], can be used to solve the dummy polysilicon filling problem. Our experimental results demonstrate that the proposed dummy fill algorithm can reduce the RTA variations to negligible amounts. We also demonstrate that our two-step approach can significantly reduce RTA variations with very limited impact on the conventional design objectives.

The rest of this chapter is organized as follows. We begin by presenting background and definitions in Section 5.2. Next, we present a formulation for computing the STI density and our efficient computation techniques in Section 5.3. Then we propose a formulation for the RTA-variation-driven floorplanning problem and our algorithm to solve it in Section 5.4, followed by a description of our LP method for inserting dummy polysilicon fills in Section 5.5. Experimental results and concluding remarks are provided in Sections 5.6 and 5.7, respectively.

5.2 Background

5.2.1 Rapid thermal annealing

During ion implantation, nuclear collisions during high energy implantation cause substrate atoms to be displaced, leading to materials defects. RTA is a subsequent annealing process, carried out at high temperature for a short duration, which is used to solve this problem. During RTA, the wafer is heated to around 1000°C for a brief period and then cooled (see Fig. 1.3(b) for an example). The RTA process thermally vibrates the atoms, re-forms the bonds among them, and activates the dopants [77].

The basic mechanism of RTA is to use radiation to rapidly transfer a large heat flux to the wafer surface; this heat then spreads around the silicon wafer by conduction. Therefore,

the surface temperature is due to both radiation and conduction. However, in today's RTA process, the time duration of heating is so short (less than 1 second [78]) that complete thermal equilibrium between conduction and radiation cannot be achieved, and the surface temperature is primarily determined by the ability of different regions of the wafer to absorb heat from the energy source in RTA.

As pointed out in [1], the RTA temperature of a point in the layout is highly dependent on the average exposed STI density in the neighborhood of the point. The exposed STI density is averaged over an $l \times l$ region, where l is the heat diffusion length. The parameter, l, is a descriptor of the region over which thermal equilibrium can be reached for the heat diffusion interaction time t, and is given by [1, 23]:

$$l = \sqrt{\frac{\sigma}{c_v \rho} t} \tag{5.1}$$

where σ, c_v and ρ are the thermal conductivity, specific heat, and density of silicon, respectively. The value of l for the process used in [1] was found to be 4 mm.

The reason why the exposed STI density is the most important factor in determining the RTA variability effects is that the reflectivity coefficients of all regions of the wafer are roughly similar, but that of STI is substantially lower, as shown in Table 5.1 [1]. Therefore, STI regions absorb significantly more heat, and transistors that lie in the neighborhood of large STI density regions tend to have lower V_T, because of higher local annealing temperature. Moreover, R_{EXT} in these regions is decreased due to the combination of improved dopant activation and increased gate overlap, both in the source and drain regions [78]. The impact of these effects on circuit-level performance is that the delay and leakage of these regions will differ from other regions with higher reflectivity.

Table 5.1: Calculated reflectivity coefficients of different regions during RTA [1].

Region	Reflectivity coefficient
N+ source/drain	0.57
P+ source/drain	0.57
Gate over isolation	0.54
Gate over Transistor	0.45
STI	0.20

Before proceeding further, we will define several concepts related to the STI density in a region. Essentially, the STI density is the ratio of the area of the exposed STI layer (that is not covered by polysilicon) to the total area of the region.

Definition 5.1 (Local STI density)**.** *The local STI density, $d(x, y)$, at a point (x, y) in the layout is the average exposed STI density defined in a small local square region centered at (x, y). The size of this local region is typically smaller than or in the order of $l/10$ [23], where l is the heat diffusion length, so that the difference of the local anneal temperature in this region is negligible.*

Definition 5.2 (Effective STI density)**.** *The effective STI density, $e(x, y)$, of a point (x, y) in the layout is the average exposed STI density over a window of size $l \times l$ units, called the* gating window*, centered at (x, y).*

5.2.2 Dummy polysilicon filling for RTA

The effective STI density captures the effect of averaging the reflectivity over a region, and the effect of heat transfer by conduction within the wafer, and can be used as a rough predictor of local annealing temperature [78]. To make the annealing temperature uniform, dummy polysilicon fills can be inserted over the STI region to even out the effective STI density, as illustrated in Fig. 5.2, so that the exposed STI density and the reflectivity of different regions become similar.

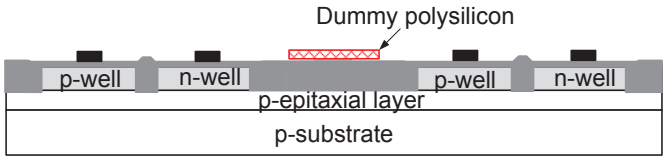


Figure 5.2: Cross-section of a partial wafer, with dummy polysilicon fills inserted before ion implantation and RTA. The dark grey regions are SiO_2 and the dark regions are polysilicon gates.

Dummy polysilicon features are electrically inactive, but can couple capacitively with other lines in neighboring layers. Since these polysilicon lines are left floating, such coupling could cause unpredictable effects on performance, and therefore, it is important to minimize the amount of inserted dummy fills.

There are two classes of methods that are used for dummy fill insertion: rule-based and model-based. Analogous to dummy filling methods for oxide CMP, rule-based methods work

directly on the local STI density, and try to make the local STI density fall within a specified limit after dummy filling. Therefore, in a region with very large STI density, dummy polysilicon fills may be inserted to block off some part of the STI region, in order to bring the local STI density within the desired limit. The problem with rule-based dummy filling is that the limit for the allowable density is typically quite large, so that the prescribed fill density must go through trial-and-error for every design; sometimes, no single value works [44]. Compared to the rule-based approach, model-based dummy polysilicon filling is based on the effective STI density, a more complex metric than the local STI density used in the rule-based method. Model-based methods are analytical and typically more computationally intensive, but are widely accepted to be more effective than rule-based methods.

5.3 The effective STI density

As mentioned before, the effective STI density correlates well with the variations in circuit parameters such as R_s. A key step in our algorithm is the computation of the effective STI density. We first create a formulation for computing the effective STI density, and then describe efficient computational techniques for this purpose.

5.3.1 A formulation for the effective STI density

Given the local STI density of each point in the layout $d(x, y)$, we proceed to calculate the effective STI density. By Definition 5.2, the effective STI density, $e(x, y)$, at a point is the average density in a gating window of size $l \times l$ around the point. Therefore, it can be computed as:

$$e(x, y) = \frac{1}{l \times l} \int_{y-l/2}^{y+l/2} \int_{x-l/2}^{x+l/2} d(u, v)\, du\, dv. \tag{5.2}$$

Let W and H be the width and height of the floorplan, respectively. Since the die is periodically repeated across the wafer, the RTA environment experiences periodicity: in other words, $d(x, y)$ is periodic. Therefore, we have

$$d(x + k_1 W, y + k_2 H) = d(x, y), \tag{5.3}$$

where k_1 and k_2 are two integers such that $(x + k_1 W, y + k_2 H)$ is a valid point on the wafer.

We introduce a gate function $g(x, y)$ defined below:

$$g(x,y) = \begin{cases} \frac{1}{l \times l} & -l/2 \leq x, y \leq l/2, \\ 0 & \text{otherwise}, \end{cases} \tag{5.4}$$

where l is the heat diffusion length for a RTA process.

From (5.2) and (5.4), it can easily be verified that $e(x, y)$ can be calculated by convolving $d(x, y)$ with $g(x, y)$:

$$e(x,y) = d(x,y) \otimes g(x,y), \tag{5.5}$$

where $\otimes$ is the circular convolution operator.

Since the effective STI is a long-range effect, and since the calculation of circular convolution in the continuous field involves computationally-intensive symbolic double integration, we choose to compute it over a discrete field. We tessellate the layout into M rows and N columns of tiles, each with side equal to s. The value of s can be chosen so that it is sufficiently small. As mentioned in Definition 5.1, the difference of the local anneal temperature in a region with size $l/10 \times l/10$ is negligible [23], so we use $s = l/10$ in this work. The heat diffusion length, l, in the continuous space maps on to a length of L tiles in the discrete space.

Let t_{ij} be the tile in the i^{th} row and the j^{th} column of the partitioned layout, $0 \leq i \leq (M-1), 0 \leq j \leq (N-1)$. Since s is small enough, the local STI densities of all the points in t_{ij} are assumed to equal that of the central point of t_{ij}, denoted as $d(i, j)$. We refer to this as the local STI density of the tile, and it is computed as the STI density averaged over the area of the tile. Similarly, we discretize the gate function, $g(i, j)$, as the discrete version of $g(x, y)$, and $e(i, j)$ as the discrete counterpart of $e(x, y)$.

We define the discrete gating window for a tile t_{ij} as a region of $L \times L$ tiles, where L is an odd number, centered around t_{ij}. The discretized version of (5.4) is

$$g(i,j) = \begin{cases} \frac{1}{L \times L} & -\lambda \leq i, j \leq \lambda, \\ 0 & \text{otherwise}, \end{cases} \tag{5.6}$$

where $\lambda = (L-1)/2$. The discretized effective STI function, $e(i, j)$ is the discrete analog of (5.5):

$$e(i,j) = d(i,j) \otimes g(i,j), \tag{5.7}$$

where $\otimes$ now represents a discrete circular convolution.

5.3.2 Efficient computation techniques

We will now discuss three methods for computing $e(i,j)$. The first method uses the definition of circular convolution to calculate $e(i,j)$ in the space domain:

$$\begin{aligned} e(i,j) &= \sum_{i',j' \in G(i,j)} \sum d(i+i', j+j') \times g(i',j') \\ &= \frac{1}{L \times L} \sum_{i',j' \in G(i,j)} \sum d(i+i', j+j'), \end{aligned} \tag{5.8}$$

where $G(i,j)$ is the gating window of $L \times L$ tiles for (i,j). The runtime required to calculate $e(i,j)\ \forall\ i,j$ is $O(nL^2)$, where $n = M \times N$ is the total number of tiles in the layout.

The second method performs the computation in the frequency domain, using the Fast Fourier Transformation (FFT). We first pad the 2-D sequence $g(i,j)$ to the same size as $d(i,j)$, and then compute $e(i,j)$ as:

$$e(i,j) = \mathrm{IFFT}\left(\mathrm{FFT}(d(i,j)) \times \mathrm{FFT}(g(i,j))\right). \tag{5.9}$$

where FFT and IFFT are the FFT operator and inverse FFT operator, respectively. The time complexity of this computation is $O(n \log n)$.

Finally, we introduce the third method, which is the procedure used in our algorithm. Considering a special property of the gating function, we find that we can compute the effective STI density of all the tiles in the layout by an incremental method. The basic idea of the method is to reuse prior computations as much as possible.

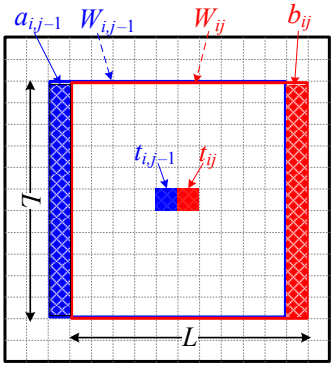


Figure 5.3: An example of incrementally computing $e(i,j)$ according to $e(i,j-1)$.

Let W_{ij} be the set of tiles in the gating window for t_{ij}. We will assume that the origin, $(i, j) = (0, 0)$, is at the bottom left corner of the layout. As compared to $W_{i,j-1}$, the elements of W_{ij} are largely identical, except that the leftmost column of $W_{i,j-1}$ is removed, and the a new column is added to the right. In other words, the computation of $e(i, j)$ can largely reuse the computation for $e(i, j-1)$, except that the contribution from one column must be removed, and that of another one added. An example for this is shown in Fig. 5.3.

Let $a_{i,j-1}$ be the leftmost column of $W_{i,j-1}$ and b_{ij} be the rightmost column of W_{ij}. Let $M_a(i, j-1) = \sum_{t_{pq} \in a_{i,j-1}} d(p, q)$ and $M_b(i, j) = \sum_{t_{pq} \in b_{ij}} d(p, q)$. Then, it is easily seen that $e(i, j) = e(i, j-1) + (M_b(i, j) - M_a(i, j-1))/L^2$.

The contents of $M_a(i, j-1)$ can be incrementally computed in a similar manner: note that $M_a(i-1, j-1)$ and $M_a(i, j-1)$ have most terms in common, except that the latter adds a new row on top and removes the one at the bottom.

The pseudocode for this incremental method to compute $e(i, j)$ is shown in Algorithm 1. It can be shown that the computational complexity of this method is $O(MN + (M + N)L)$. If $L < \min(M, N)$, as is typically the case since the size of die is usually larger than L, the runtime is $O(n)$.

5.3.3 Finding the discretized local STI density

In this section, we will show how to compute the local STI density for every tile in the layout: this is used in the computation of effective STI density.

It is well known that the STI density of a floorplan block can vary significantly over the area of the block, unless the block is a very regular structure, such as a memory module. To address the issue, we allow the specification of different STI densities to different subblocks of a block if its size is larger than $s \times s$, where $s = l/10$, since local annealing temperature variations in an $s \times s$ window are negligible.

Then the discretized local STI density function, $d(i, j)$, can be computed based on the floorplan information, which provides the local STI densities of each subblock (as explained in Section 5.1). For a tile, t_{ij}, that overlaps with one or more subblocks, each with (possibly) different average STI densities, and/or with dead space, the value of $d(i, j)$ should be the STI density averaged over the area of t_{ij}. Dead space on the chip is assumed to have a user-specified STI density of η; typically, $\eta = 1$.

Algorithm 1 The incremental method for computing the effective STI densities for all tiles in the layout. In the algorithm, v_r is a vector, $\lambda = (L-1)/2$, M_c is a $M \times N$ matrix, and $\psi(i,j)$ is a function to transform any (i,j) to (i',j'), where $i' = i \bmod M, j' = j \bmod N$.

for i = 0 to $M-1$ **do**
  $v_r(i) \leftarrow \sum_{j=-\lambda}^{\lambda} d(\psi(i,j))$
end for
for j = 0 to $N-1$ **do**
  $M_c(0,j) \leftarrow \sum_{i=-\lambda}^{\lambda} d(\psi(i,j))$
end for
for i = 1 to $M-1$ **do**
  for j = 0 to $N-1$ **do**
    $M_c(i,j) \leftarrow M_c(i-1,j) + d(\psi(i+\lambda,j)) - d(\psi(i-\lambda-1,j))$
  end for
end for
$e(0,0) \leftarrow \sum_{j=-\lambda}^{j=\lambda} M_c(\psi(0,j))$
// Compute $e(i,j)$ for the 0^{th} column tiles
for i = 1 to $M-1$ **do**
  $e(i,0) \leftarrow e(i-1,0) + v_r(\psi(i+\lambda)) - v_r(\psi(i-\lambda-1))$
end for
// Compute $e(i,j)$ for the rest of the tiles
for i = 0 to $M-1$ **do**
  for j = 1 to $N-1$ **do**
    $e(i,j) \leftarrow e(i,j-1) + M_c(\psi(i,j+\lambda)) - M_c(\psi(i,j-\lambda-1))$
  end for
end for
for each $e(i,j)$, $e(i,j) \leftarrow e(i,j)/L^2$

5.4 RTA-driven floorplanning

The RTA-variation-driven floorplanning problem is stated as follows:
Given a set of circuit blocks, along with their widths, heights, pin positions, and local STI densities of all the subblocks, and a netlist that captures the connections between the blocks, find a legal floorplan such that a weighted cost function of area, wirelength, and the variations in the effective STI density is minimized.

Our floorplanner is an adaptation of the Parquet package [79,80], which is based on simulated annealing (SA) and the sequence pair representation [81]. The FAST-SP algorithm [82] is used to evaluate a sequence pair within the algorithm. We suggest new objective functions to drive the SA process, introduced in Section 5.4.1. Moreover, we use a heuristic to improve the performance of the algorithm, as described in Section 5.4.2.

5.4.1 Cost function

Our goal is to optimize the uniformity of the distribution of the effective STI density. We use the weighted sum

$$\mathcal{U} = \mathcal{R} + \gamma\mathcal{G} \tag{5.10}$$

as the metric to represent this uniformity, where $\mathcal{R}$ is the global variation, i.e., the difference between the maximum and the minimum, of the effective STI density in the floorplan, $\mathcal{G}$ is the sum of the absolute values of the gradients of effective STI density, and γ is a user-defined parameter. This first component, $\mathcal{R}$, is directed towards reducing global variations of the effective STI density, while $\mathcal{G}$ serves to reduce local variations. The utility of the latter is, for example, in reducing the variation in the clock skew between electrically adjacent registers which are also (usually) spatially adjacent. However, minimizing $\mathcal{G}$ does not obviate the need for the metric $\mathcal{R}$, since even a small level of local variation can accumulate into a large undesirable global variation.

To further define $\mathcal{G}$, we define the horizontal gradient, h_{ij}, and vertical gradient, v_{ij}, of the effective STI density of a tile t_{ij} as:

$$h_{ij} = \begin{cases} 0 & \text{if } j = N-1, \\ e(i,j+1) - e(i,j) & \text{otherwise,} \end{cases} \tag{5.11}$$

$$v_{ij} = \begin{cases} 0 & \text{if } i = M-1, \\ e(i+1,j) - e(i,j) & \text{otherwise.} \end{cases} \tag{5.12}$$

We can then compute $\mathcal{G}$ as:

$$\mathcal{G} = \sum_{\text{all } t_{ij}} (|h_{ij}| + |v_{ij}|). \tag{5.13}$$

In practice, a larger weight should be assigned to $\mathcal{G}$, which means that γ should be larger than 1, because the local variation between two adjacent tiles tends to have a larger impact on circuit design compared with the variation between two tiles far from each other.

In optimizing STI uniformity through floorplanning, it is important to also factor in conventional floorplanning metrics. Therefore, the following cost function is used in the floorplanner:

$$\mathcal{C} = \overline{\mathcal{A}} + \alpha\overline{\mathcal{W}} + \beta\left(\overline{\mathcal{R}} + \gamma\overline{\mathcal{G}}\right), \tag{5.14}$$

where $\mathcal{C}$ is the total cost; $\overline{\mathcal{A}}$ is the normalized version for the area of the floorplan ($\mathcal{A}$); $\overline{\mathcal{W}}$ is the normalized version for the total wirelength of the floorplan ($\mathcal{W}$); $\overline{\mathcal{R}}$ and $\overline{\mathcal{G}}$ are the normalized versions of $\mathcal{R}$ and $\mathcal{G}$ from (5.10), respectively, and serve as metrics of the uniformity of effective STI density in the floorplan; and α, β and γ are user-defined parameters. Note that during SA, we compute the change in the cost, Δ, which is used in the Metropolis rule, using the relative change of each component:

$$\Delta = \frac{\Delta\mathcal{A}}{\mathcal{A}_{blocks}} + \alpha\frac{\Delta\mathcal{W}}{\mathcal{W}_{old}} + \beta\frac{\Delta\mathcal{R}}{\mathcal{R}_{old}} + \beta\gamma\frac{\Delta\mathcal{G}}{\mathcal{G}_{old}}. \tag{5.15}$$

Here, $\Delta\mathcal{A}$, $\Delta\mathcal{W}$, $\Delta\mathcal{R}$ and $\Delta\mathcal{G}$ are the change in $\mathcal{A}$, $\mathcal{W}$, $\mathcal{R}$ and $\mathcal{G}$, respectively. This is consistent with normalized objective function in (5.14), with the normalization carried out relative to $\mathcal{A}_{blocks}$, the total area of all the blocks, and $\mathcal{W}_{old}$, $\mathcal{R}_{old}$, and $\mathcal{G}_{old}$, the old values of $\mathcal{W}$, $\mathcal{R}$ and $\mathcal{G}$, respectively. This is also the way used in Parquet.

5.4.2 Heuristics

Our implementation is based on the Parquet package [79, 80]. We make the following major modifications: first, we add our proposed algorithms to make Parquet capable of optimizing RTA variations; second, when optimizing RTA variations, we change the algorithm to use a two-stage SA scheme.

In the first stage, a conventional SA is performed, and only the area and wirelength are optimized. After the first stage ends, we start the second SA stage from a low temperature (100

in our algorithm), using the best solution obtained in the first stage as the initial solution. In this stage, the cost function (5.14) is used to evaluate a floorplan solution. Since the start temperature is low, the second stage serves more like a local search with certain hill-climbing ability.

In order to improve the performance of our algorithm, we introduce an effective heuristic in the second SA stage to handle the effect of the $\mathcal{U}$ term in (5.10). A penalty function is introduced for this term in the cost function to add barriers for accepting a floorplan candidate with better STI uniformity but much worse area. We observe that for a given benchmark, a floorplan solution with larger area contains more area of dead space, which is assumed to have a high STI density, η, and tends to mitigate the variations in the effective STI density of different blocks in the layout. Therefore, a floorplan solution with larger area tends to have a smaller $\mathcal{U}$ term (i.e., the $\mathcal{R}$ and $\mathcal{G}$ terms) in (5.14). However, in practice, a circuit with large area of dead space is not desirable, and therefore, we apply a penalty function $\theta = \phi_c/\phi_t$ for the $\mathcal{U}$ term, where ϕ_c is the dead space ratio for the current floorplan solution and ϕ_t is the targeted dead space ratio that the designer wishes to achieve after floorplanning. With this penalty function, when calculating Δ in (5.15), each term of $\mathcal{R}$, $\mathcal{R}_{old}$, $\mathcal{G}$ and $\mathcal{G}_{old}$ is scaled by its own value of θ (for the current solution or the old solution). Using this penalty function, when the current floorplan solution has a large area and then a large ϕ_c, a large penalty factor will be assigned to $\mathcal{U}$, which makes this solution less likely to be accepted in SA. In our experiments, $\phi_t = 0.1$.

5.5 Inserting dummy polysilicon fills

After the floorplan has been optimized, we then insert dummy polysilicon fills to further improve the uniformity of effective STI density. Since the problem structure is somewhat similar to that of oxide CMP fill, our method is based on an adaptation of the oxide CMP fill solution in [44]. The approach in [44] has two steps: global dummy fill assignment using LP, followed by local dummy fill insertion where the details of the fills at the local level are determined. In this work, we focus on the first step, since the methods for detailed local dummy fill insertion are very similar to that in CMP dummy fill, and the reader is referred to [44] for details on issues such as the determination of the shapes and locations of dummy fills.

Next, we present the LP formulation for the dummy polysilicon filling problem. Compared to the method in [44], additional constraints to control the gradient of the effective STI density are introduced.

As before, the layout is partitioned to M rows and N columns of tiles, so that the total number of tiles is given by $n = MN$. In this chapter, the amount of dummy fill refers to the ratio of the area of the dummy fills in a tile to the total area of the tile. We define x_{ij} to be the amount of dummy fill inserted into tile t_{ij}, x^a_{ij} to be the upper bound of dummy fill that can be inserted into t_{ij}, and d^0_{ij} to be the initial local STI density in t_{ij}. The local STI density of t_{ij} after dummy filling can be calculated as:

$$d(i,j) = d^0(i,j) - x_{ij}. \tag{5.16}$$

Note that after inserting dummy polysilicon fills, the local STI density is *reduced*, and therefore $d(i,j)$ will be no larger than $d^0(i,j)$. The effective STI density can be calculated using Algorithm 1.

We formulate the problem to ensure that the global variation and the gradients of the effective STI density are within a small range. Let ε_1 be the required bound for the global variation of the effective STI density, and ε_2 be the required bound for the gradient of the effective STI density of any two adjacent tiles. The cost of inserting dummy fills reflects the side-effects of dummy fills, and may be different in different tiles. Therefore, a weight c_{ij} could be assigned to the dummy fills inserted into tile t_{ij}, and this can be calculated using the cost zone method in [44]. The cost of the dummy fills inserted in t_{ij} can then be written as $c_{ij} \cdot x_{ij}$.

To minimize the total cost of the dummy fills, the dummy polysilicon filling problem is formulated as the following LP problem, with $O(n)$ variables and $O(n)$ constraints:

$$\text{minimize} \sum_{\text{all } ij} c_{ij} \cdot x_{ij} \tag{5.17}$$

subject to:

$$0 \le x_{ij} \le x^a_{ij}, \forall\, i,j, \tag{5.18}$$

$$e_L \le e(i,j) \le e_H, \forall\, i,j, \tag{5.19}$$

$$e_H - e_L \le \varepsilon_1, \tag{5.20}$$

$$|e(i+1,j) - e(i,j)| \le \varepsilon_2,\ 0 \le i \le M-2, \forall\, j, \tag{5.21}$$

$$|e(i,j+1) - e(i,j)| \le \varepsilon_2,\ 0 \le j \le N-2, \forall\, i, \tag{5.22}$$

where e_H and e_L are auxiliary variables that determine the maximum and minimum STI density, respectively. In this formulation, the RTA variations are controlled by constraints (5.19)–(5.22). The inputs are d^0_{ij}, x^a_{ij}, c_{ij}, ε_1 and ε_2; the output is x_{ij}.

5.6 Experimental results

5.6.1 RTA-aware floorplanning

We have implemented and tested our algorithm on a 64-bit Linux machine with a 2.6 GHz dual-core AMD Opteron™ 2218 processor and 2 GB memory. We call our implementation as pRTA, and we compare its results with those from the original Parquet, with only area and wirelength optimized.

In our experiments, the RTA profile for 65 nm technology is used, and we take $l = 4$ mm, in accordance with [1]. The side of a tile, s, equals $l/10$. Two groups of standard benchmarks from the MCNC and GSRC suites are used to test the programs, and basic information about the circuits is listed in Table 5.2. Since some of the benchmarks are much smaller than $l \times l$, we scale their size by a ratio, S, as listed in the table. The areas of the scaled circuits vary from about 7 mm×7 mm to 15 mm×15 mm.

Table 5.2: Benchmark characteristics: here, S is area scaling factor, and A_S is area of blocks, after scaling, in mm^2.

Circuit	apte	xerox	hp	ami33	ami49	n100	n200	n300
# blocks	9	10	11	33	49	100	200	300
A_S	47	77	79	116	142	173	191	230
S	1	2	3	10	2	31	33	29

As mentioned in Section 5.3.3, large blocks are likely to have different local STI densities in different subblocks. Since data on STI densities are not readily available for these benchmarks, we generate these data pseudorandomly in the following way[2]. First, we select the average STI density, g_i, of each block B_i to be a random number between 0.25 and 0.75. Next, we set a variation range, v_i, for the local STI densities of all of the subblocks within a block: in our experiments, this value is set to 0.30. Then the local STI density of a subblock on B_i will be chosen randomly between $\max(g_i - v_i, g_{min})$ and $\min(g_i + v_i, g_{max})$, where $g_{min} = 0.15$ and $g_{max} = 0.85$ are, respectively, assumed as the possible minimal and maximal STI density for a logic subblock[3] in practice. Note that these steps are performed after scaling the benchmarks, so that the distribution of STI remains realistic in spite of the artificial scaling operation. All the

[2]In real designs, the STI density information can be usually obtained, as explained in Section 5.1.

[3]Note that for a tile containing some dead space, its local STI density can be larger than 0.85.

random numbers generated here are based on a uniform distribution, using a function from the standard Unix C/C++ library. Further, we set the local STI density of the dead space η to 1 in our experiments, and also assume that the space between the dies on the wafer is negligible.

For a fair comparison, we use all of the default parameters of Parquet (which have been tuned by the authors of the package): using the notation in (5.14), we choose

$$\alpha = 1.00, \beta = 0, \tag{5.23}$$

so that the optimization objectives are $\mathcal{A}$ and $\mathcal{W}$. For pRTA, we tune the weights for the circuit ami49, and then use these weights for all the circuits: specifically, we use

$$\alpha = 1.43, \beta = 0.43, \tag{5.24}$$

so that $\mathcal{A}$, $\mathcal{W}$, $\mathcal{R}$, and $\mathcal{G}$ are all optimized. To place more importance on local RTA variations, we use $\gamma = 10.00$. For completeness, we will list ami49 in our table of results, but the gains on this circuit can be appropriately discounted since the parameters were tuned on it.

Since Parquet and pRTA are based on a stochastic algorithm, the results from different runs can be different. We run each algorithm for 10 runs, and list the best result over the 10 runs in Table 5.3.

To demonstrate the impact of our approach on the RTA variations directly, we determine the relationship between the polysilicon sheet resistance R_s and the effective STI density in Fig. 5.1, based on measured data from [1], as:

$$y = -0.9267x + 1.5223, \tag{5.25}$$

where y denotes R_s in relative units, and x is the effective STI density.

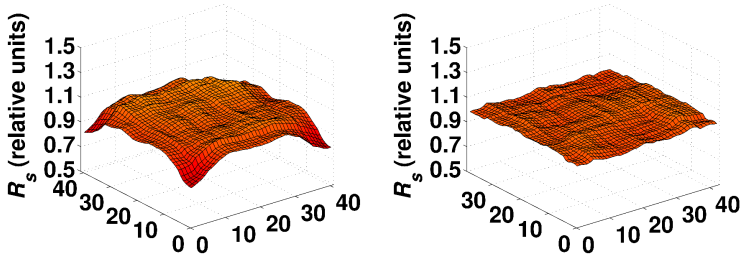
Using (5.25), from the variation in the effective STI density in a floorplan, we can calculate and report the variations in R_s for the floorplan. Recall that in Section 5.4.1, we had defined $\mathcal{R}$ and $\mathcal{G}$ as, respectively, the global and local variation in the effective STI density. We define $\mathcal{R}_{R_s}$, the global variation of R_s, as the difference between the maximum and minimum value of R_s over the layout. The local variation, $\mathcal{G}_{R_s}$, of R_s is defined analogously to (5.11), (5.12), and (5.13), with $e(.,.)$ in each equation substituted by $R_s(.,.)$, and represents the sum of the absolute values of the gradients of R_s over all tiles. We also report the maximum gradient, σ_{R_s}, computed as the maximum, over all tiles, of the absolute values of h_{ij} or v_{ij}, corresponding to the gradient of R_s.

Table 5.3: Comparison of Parquet and pRTA.

Circuit	Area usage(%)		Wirelength (mm)		$\mathcal{R}_{R_s}$		$\mathcal{G}_{R_s}$		σ_{R_s}		Total time(s)	
	Parquet	pRTA	Parquet	pRTA	Parquet	pRTA	Parquet	pRTA	Parquet	pRTA	Parquet	pRTA
apte	96.09	97.90	518.07	519.34	0.15	0.08	6.08	4.32	0.044	0.032	1.2	13.1
xerox	94.11	94.11	749.85	778.20	0.21	0.21	10.69	10.34	0.035	0.044	3.0	26.2
hp	89.14	89.07	652.07	618.76	0.27	0.24	13.69	11.67	0.055	0.043	1.5	28.8
ami33	88.89	92.44	694.65	715.77	0.24	0.18	16.90	13.95	0.049	0.050	8.3	120.4
ami49	88.18	92.52	1409.23	1482.13	0.30	0.20	22.36	16.96	0.054	0.051	22.0	228.4
n100	90.64	90.29	9813.00	10109.40	0.18	0.08	19.69	10.87	0.044	0.039	90.4	682.2
n200	88.39	87.80	18790.10	20083.00	0.21	0.05	21.68	8.31	0.043	0.016	393.6	2312.8
n300	87.90	85.93	19884.80	20414.30	0.30	0.10	28.62	18.27	0.050	0.033	813.7	5120.8
Compare	1.00	1.01	1.00	1.02	1.00	0.61	1.00	0.71	1.00	0.83	1.0	10.4

A comparison of the results of pRTA and Parquet is shown in Table 5.3, for the benchmark circuits listed in Table 5.2. We show comparisons of the percentage area usage, the total wirelength, the global variation, $\mathcal{R}_{R_s}$, the local variation, $\mathcal{G}_{R_s}$, the maximum gradient, σ_{R_s}, and the total runtime for 10 runs. From the table, we can see that with tolerable overheads in area and wirelength, the proposed algorithm is effective in reducing the variations in the values of circuit parameters affected by RTA variations. Although our results are shown for variations in R_s, variations in R_{EXT} and V_T are similar in nature since they both have been shown to have excellent linear correlations with R_s [1]. Compared with a floorplanner only optimizing traditional objectives, on average, our algorithm can reduce global variation in R_s by 39%, local variations by 29% and maximum gradient by 17%, with almost the same area and wirelength. Note that in fact, our floorplanner obtains a little better area usage than Parquet on average (a *larger* value for area usage is better), which is probably because our two-stage SA scheme is more effective on area optimization than Parquet for some circuits.

The runtime of our algorithm is acceptable even for the largest circuit in the benchmarks, which is about 1.4 hours for 10 runs. The major reason for the increase of runtime of our algorithm is that the evaluation of the effective STI density for a floorplan solution incurs a significant computational expense, about 8$\times$ longer than the evaluation of area and wirelength. This is mainly due to the large number of tiles in a layout and subblocks in a block. For example, circuit ami49 has 49 blocks and 408 nets, but about 1000 tiles and 1000 subblocks in total. Therefore, though the proposed algorithm to compute the effective STI density has linear runtime complexity, its runtime is still noticeably longer compared with the evaluation of area and wirelength. In spite of this, the runtime is very reasonable in absolute terms.



(a) (b)

Figure 5.4: Topographies of R_s in the layouts obtained by Parquet and pRTA for circuit n300: (a) without RTA optimization (Parquet), and (b) with RTA optimization (pRTA).

To present the improvement of our algorithm in the RTA variations more intuitively, the topographies of R_s in the layouts obtained by Parquet and pRTA for circuit n300 are shown in Fig. 5.4. The x and y axes represent the physical coordinates with one unit equal to 0.4 mm, and the z axis shows the R_s value at each location in relative units. From the plots, we can clearly see that our algorithm reduces both the global and local variations in R_s induced by RTA significantly.

One phenomenon shown in Fig. 5.4(a) is that the corners of the layout tend to have noticeably smaller R_s values than other locations in the layout. This is probably because that dead space, whose STI density η equals 1, tends to form around the corners of the layout in traditional floorplanners such as Parquet, causing the effective STI density around corners to be larger than other locations in the layout, and thus smaller R_s around corners, since R_s is a monotonically decreasing function of the effective STI density by (5.25).

5.6.2 Dummy polysilicon filling

In the second step of our optimization framework, we test whether the variations in circuit parameters such as R_s are still larger than the required bounds after floorplanning. If so, we insert dummy polysilicon fills into the layout to further reduce the variations, using the formulation presented in Section 5.5.

The settings for the dummy fill procedure are as follows. In a tile t_{ij}, we assume that the lower bound of $d(i, j)$ after dummy filling is 0.15, which is reserved to consider the requirements of design rules, e.g., minimal space between dummy fills and polysilicon gates, and to ensure

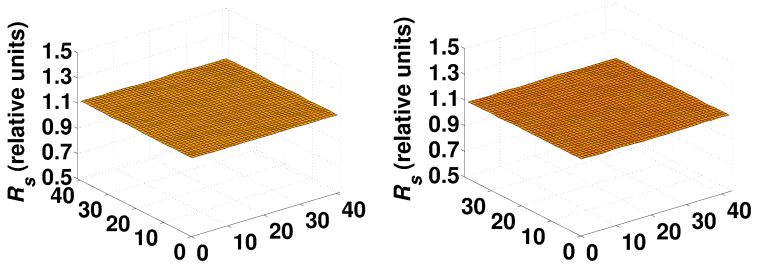
the manufacturability. Therefore $x_{ij}^a = \max(d^0(i,j) - 0.15, 0)$, where max operation is used to ensure x_{ij}^a not negative. For simplicity, the weight c_{ij} for the cost of the dummy fills in each tile is set to 1 in our experiments. The parameters of (5.20)–(5.22) are chosen as $\varepsilon_1 = 0.0108$ and $\varepsilon_2 = 0.0022$. Based on (5.25), these values correspond to a global variation of 1% in R_s, and a maximum gradient for R_s of 0.2%. A commercial LP solver ILOG CPLEX [74] is used to solve the dummy polysilicon filling problem. The hardware used is the same as that used to test floorplanners in Section 5.6.1.

Table 5.4: Dummy filling results for the floorplans obtained by Parquet and pRTA. The column "Ratio" denotes the ratio of the total area of the dummy fills to the total STI area in the layout.

Circuit	Area of fill (mm^2)		Ratio (%)		Fill time (s)	
	Parquet	pRTA	Parquet	pRTA	Parquet	pRTA
apte	7.29	5.60	24.95	20.40	0.8	0.9
xerox	14.71	12.02	30.05	24.56	2.2	2.2
hp	14.83	12.91	28.80	27.09	1.9	2.4
ami33	21.15	17.89	28.55	24.79	5.6	4.9
ami49	30.66	24.00	33.28	29.08	7.3	4.7
n100	25.18	16.30	23.90	15.35	8.9	9.1
n200	24.52	15.93	19.39	13.02	12.7	16.0
n300	31.84	23.43	20.80	15.37	13.2	13.5
Compare	1.00	0.76	1.00	0.80	1.0	1.0

The results are listed in Table 5.4, and they show that to achieve the same requirements of uniformity in the effective STI density, the amount of dummy fills inserted can be reduced by 24%, on average, when using the floorplan solutions obtained by pRTA instead of those obtained by Parquet. Therefore, we can see that our floorplanning algorithm has two benefits in improving the RTA variations: first, it provides a floorplan solution with reduced RTA variations directly, which is useful when dummy polysilicon filling is not suitable; second, when inserting dummy polysilicon, fewer dummy fills are inserted, which can reduce the coupling effects induced by dummy fills. Specifically, take the largest circuit n300 in the benchmarks as an example. By using our two-step approach, the global variation in R_s is reduced from 0.30 to 0.01, while the maximum gradient of R_s is reduced from 0.050 to 0.002, with 24% fewer dummy fills inserted.

The runtime of the dummy filling method is quite short, and is no more than 16 seconds for all the benchmarks.



(a) (b)

Figure 5.5: Topographies of R_s in the layouts obtained by Parquet and pRTA for circuit n300 after dummy polysilicon filling: (a) without RTA optimization (Parquet), and (b) with RTA optimization (pRTA). Our dummy filling algorithm can produce even profiles for both layouts, but as shown in Table 5.4, the overhead of (a) is significantly larger than that of (b).

The topographies of R_s for circuit n300 after dummy filling are shown in Fig. 5.5 to demonstrate the effectiveness of the dummy filling algorithm. The x, y and z axes have the same meanings and units as those in Fig. 5.4. The figures show that after dummy filling, both the global and the local variations are negligible.

5.7 Conclusion

RTA-induced variations are an emerging challenge to integrated circuit designs, at 65 nm technology and below. This chapter has addressed this challenge using a two-step approach. First, the STI density distribution in the circuit layout is smoothed by floorplanning. Next, dummy polysilicon fills are inserted into layout to further reduce the RTA variations. These methods can be easily integrated into the design flow to reduce the RTA-induced variations and then improve the yield and performance of a circuit.

Chapter 6

Conclusion

With continuous technology scaling, new challenges have emerged in the field of physical design. This thesis has proposed techniques to address some important problems in physical design including routability evaluation, buffer area reduction and optimization of the variations induced by CMP and RTA processes.

In Chapter 2, we have addressed two important aspects of routability evaluation: the accuracy of congestion estimation and the metric to evaluate the routability of a design. We have explored the problems in the existing mainstream congestion analysis tools: lack of local congestion modeling, existence of noise hot spots and ineffective congestion metrics. Motivated by these, we have proposed models for local routing resources, a smoothing technique to reduce noise hot spots and a new ACE metric. We have conducted extensive experiments to validate our techniques using industrial circuits. Experimental results have shown that the proposed local resource model can improve the accuracy and fidelity of congestion analysis, and better predict detailed routing issues such as opens and shorts. The good scalability of our model enables designers to use large g-cells to accelerate the process of congestion analysis, thereby speeding design closure. Furthermore, we have shown that the proposed smoothing technique can help to obtain accurate routability evaluation. In addition, we have demonstrated that the ACE metric can represent a congestion plot with higher fidelity, and predict the routability more accurately, compared with conventional metrics. Finally, we have showed that with ACE metric, a routability-driven placer can perform better and can improve the routability of a design significantly.

To overcome the buffer explosion trend with technology scaling, we have proposed an algorithm to improve the layer directive planning in Chapter 3. We have first revealed the

problems in existing physical design tools that have not fully utilize the timing benefits of the thick metal layers, and then extended an industrial physical synthesis flow by adding an early layer directive planning stage, where the core algorithm, CATALYST, assigns as many nets as possible to thick metal layers to maximize the timing benefits including reducing the buffer area. An important feature of our algorithm is that it exploits the embedded routing engine along with heuristics to well control the congestion. Experiments have demonstrated that our flow with proposed algorithm can significantly reduce the buffer area.

Chapter 4 has discussed the routing techniques for oxide CMP variation optimization. We have pointed out that the ultimate goal to deal with oxide CMP variation is minimization of the amount of dummy fills required for CMP instead of minimization of the variation in IPD or EPD after routing, which was commonly used in previous works. Then by theoretical and experimental analyses, we have determined that a good surrogate of minimizing dummy fill is minimizing the maximal EPD. Based on these findings, we have elaborated cost functions to minimize the maximal EPD in routing process. Moreover, we have also added a postprocessing stage to further reduce the maximal EPD. Finally, we have run experiments on a set of standard benchmarks to verify our theoretical analyses and to demonstrate the effectiveness of our new routing algorithm for dummy fill optimization.

Finally, in Chapter 5, we have presented the first known attempt to address RTA-induced variations using physical design techniques. In this work, we have proposed a two-step approach to reduce the impact of RTA-induced variations. We have defined a concept of effective STI density similar to the concept of EPD in CMP and have proposed an efficient incremental method to compute the effective STI density for the whole circuit. Furthermore, we have adapted a conventional floorplanner to handle the new objective of minimizing the variations in effective STI density, and proposed a two-stage simulated annealing heuristic to improve its quality. As the second step of our efforts, we have formulated the dummy polysilicon filling problem for minimizing the RTA-induced variations as a linear programming problem, inspired by a dummy filling formulation for oxide CMP. Experimental results have demonstrated that our floorplanner can reduce the global and local RTA variations significantly, and requires much fewer dummy fills to achieve the same RTA variations than a traditional floorplanner. Moreover, the proposed dummy filling algorithm has been shown to be very effective and can further reduce the RTA variations to negligible amounts.

References

[1] I. Ahsan, N. Zamdmer, O. G. O, R. Logan, E. Nowak, H. Kimura, J. Zimmerman, G. Berg, J. Herman, E. Maciejewski, A. Chan, A. Azuma, S. Deshpande, B. Dirahoui, G. Freeman, A. Gabor, M. Gribelyuk, S. Huang, M. Kumar, K. Miyamoto, D. Mocuta, A. Mahorowala, E. Leobandung, H. Utomo, and B. Walsh. RTA-driven intra-die variations in stage delay, and parametric sensitivities for 65nm technology. In *Proceedings of the IEEE Symposium on VLSI Technology*, pages 170–171, 2006.

[2] C. Alpert and G. Tellez. The importance of routing congestion analysis. *DAC Knowledge Center Online Article*, 2010. http://www.dac.com/back_end+topics.aspx?article=47&topic=2. Accessed: 02/07/2013.

[3] A. B. Kahng and K. Samadi. CMP fill synthesis: A survey of recent studies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):3–19, 2008.

[4] Y. Li, A. Farshidi, L. Behjat, and W. Swartz. High performance post-placement length estimation techniques. *International Journal of Information and Computer Science*, 1(6):144–152, September 2012.

[5] D. O. Ouma, D. S. Boning, J. E. Chung, W. G. Easter, V. Saxena, S. Misra, and A. Crevasse. Characterization and modeling of oxide chemical-mechanical polishing using planarization length and pattern density concepts. *IEEE Transactions on Semiconductor Manufacturing*, 15(2):232–244, 2002.

[6] The history of the integrated circuit. http://www.nobelprize.org/educational/physics/integrated_circuit/history/. Accessed: 01/14/2013.

[7] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W. Shih, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki. Managing process variation in Intel's 45nm CMOS technology. *Intel Technology Journal*, 12(2):93–109, 2008.

[8] D. Hathaway, L. Stok, D. Chinnery, and K. Keutzer. Design flows. In L. Scheffer, L. Lavagno, and G. Martin, editors, *EDA for IC Implementation, Circuit Design, and Process Technology*, pages 1-1–1-15. CRC Press, Boca Raton, FL, 2011.

[9] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar. *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications, Boca Raton, FL, 2008.

[10] Y. Xu, Y. Zhang, and C. Chu. FastRoute 4.0: Global router with efficient via minimization. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 576–581, 2009.

[11] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang. NTHU-Route 2.0: a fast and stable global router. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 338–343, 2008.

[12] M. Cho, K. Lu, K. Yuan, and D. Z. Pan. BoxRouter 2.0: Architecture and implementation of a hybrid and robust global router. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 503–508, 2007.

[13] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang. High-performance global routing with fast overflow reduction. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 582–587, 2009.

[14] T.-H. Wu, A. Davoodi, and J. T. Linderoth. A parallel integer programming approach to global routing. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 194–199, 2010.

[15] M. Pan and C. Chu. IPR: An integrated placement and routing algorithm. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 59–62, 2007.

[16] J. A. Roy, N. Viswanathan, G.-J. Nam, C. J. Alpert, and I. L. Markov. CRISP: Congestion reduction by iterated spreading during placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 357–362, 2009.

[17] H. Shojaei, A. Davoodi, and J. T. Linderoth. Congestion analysis for global routing via integer programming. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 256–262, 2011.

[18] Z. Li, C. J. Alpert, S. Hu, T. Muhmud, S. T. Quay, and P. G. Villarrubia. Fast interconnect synthesis with layer assignment. In *Proceedings of the ACM International Symposium on Physical Design*, pages 71–77, 2008.

[19] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State Circuits*, 37(2):183–190, 2002.

[20] S. S. Sapatnekar. Variability and statistical design. *IPSJ Transactions on System LSI Design Methodology*, 1:18–32, 2008.

[21] S. A. Campbell. *The Science and Engineering of Microelectronic Fabrication*. Oxford University Press, USA, New York, NY, 2nd edition, 2001.

[22] P. J. Timans, W. Lerch, J. Niess, S. Paul, N. Acharya, and Z. Nenyei. Challenges for ulta-shallow junction formation technologies beyond the 90nm node. In *Proceedings of the International Conference on Advanced Thermal Processing of Semiconductors*, pages 17–33, 2003.

[23] A. Colin, P. Morin, F. Cacho, H. Bono, R. Beneyton, M. Bidaud, D. Mathiot, and E. Fogarassy. Simulation of the sub-melt laser anneal process in 45 CMOS technology—Application to the thermal pattern effects. *Materials Science and Engineering: B*, 154–155:31–34, 2008.

[24] J. Lou, S. Thakur, S. Krishnamoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):32–41, 2002.

[25] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *Proceedings of the ACM International Symposium on Physical Design*, pages 204–209, 2004.

[26] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The DAC 2012 routability-driven placement contest and benchmark suite. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 774–782, 2012.

[27] M. Pan and C. Chu. FastRoute: a step to integrate global routing into placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 464–471, 2006.

[28] T. Taghavi, C. Alpert, A. Huber, Z. Li, G.-J. Nam, and S. Ramji. New placement prediction and mitigation techniques for local routing congestion. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 621–624, 2010.

[29] U. Brenner and A. Rohe. An effective congestion driven placement framework. In *Proceedings of the ACM International Symposium on Physical Design*, pages 6–11, 2002.

[30] M. Hsu, S. Chou, T. Lin, and Y. Chang. Routability-driven analytical placement for mixed-size circuit designs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 80–84, 2011.

[31] M. Cho, D. Pan, H. Xiang, and R. Puri. Wire density driven global routing for CMP variation and timing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 487–492, 2006.

[32] H.-Y. Chen, S.-J. Chou, S.-L. Wang, and Y.-W. Chang. Novel wire density driven full-chip routing for CMP variation control. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 831–838, 2007.

[33] Y. Zhang and C. Chu. GDRouter: interleaved global routing and detailed routing for ultimate routability. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 597–602, 2012.

[34] Y. Zhang and C. Chu. RegularRoute: an efficient detailed router with regular routing patterns. In *Proceedings of the ACM International Symposium on Physical Design*, pages 45–52, 2011.

[35] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen. Algorithms and data structures for fast and good VLSI routing. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, pages 459–464, 2012.

[36] M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, and J. Vygen. BonnRoute: Algorithms and data structures for fast and good VLSI routing. *Accepted for publication in the ACM Transactions on Design Automation of Electronic Systems*, 2013.

[37] C. Chu and Y.-C. Wong. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):70–83, 2008.

[38] W. C. Naylor, R. Donelly, and L. Sha. Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer, 2003. U.S. Patent 6671859 B1.

[39] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):734–747, 2005.

[40] T. Chen, Z. Jiang, T. Hsu, H. Chen, and Y. Chang. NTUplace3: an analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, July 2008.

[41] T. Chen, A. Chakraborty, and D. Z. Pan. An integrated nonlinear placement framework with congestion and porosity aware buffer planning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 702–707, 2008.

[42] Z. Jiang, B. Su, and Y. Chang. Routability-driven analytical placement by net overlapping removal for large-scale mixed-size designs. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 167–172, 2008.

[43] T. Chan, J. Cong, and K. Sze. Multilevel generalized force-directed method for circuit placement. In *Proceedings of the ACM International Symposium on Physical Design*, pages 185–192, 2005.

[44] R. Tian, D. F. Wong, and R. Boone. Model-based dummy feature placement for oxide chemical-mechanical polishing manufacturability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(7):902–910, 2001.

[45] Y. Wei and S. S. Sapatnekar. Dummy fill optimization for enhanced manufacturability. In *Proceedings of the ACM International Symposium on Physical Design*, pages 97–104, 2010.

[46] H. Chen, S. Chou, and Y. Chang. Density gradient minimization with coupling-constrained dummy fill for CMP control. In *Proceedings of the ACM International Symposium on Physical Design*, pages 105–111, 2010.

[47] C. J. Alpert, Z. Li, M. D. Moffitt, G. J. Nam, J. A. Roy, and G. Tellez. What makes a design difficult to route. In *Proceedings of the ACM International Symposium on Physical Design*, pages 7–12, 2010.

[48] M. D. Moffitt. MaizeRouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(11):2017–2026, 2008.

[49] J. A. Roy and I. L. Markov. High-performance routing at the nanometer scale. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 496–502, 2007.

[50] P. Saxena and C. L. Liu. Optimization of the maximum delay of global interconnects during layer assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(4):503–515, 2001.

[51] Y. Jia, Y. Cai, and X. Hong. Timing driven layer assignment considering via resistance and coupling capacitance. In *Proceedings of the International Conference on Communications, Circuits and Systems*, pages 1172–1176, 2007.

[52] S. Hu, Z. Li, and C. J. Alpert. A polynomial time approximation scheme for timing constrained minimum cost layer assignment. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 112–115, 2008.

[53] S. Hu, Z. Li, and C. J. Alpert. A fully polynomial-time approximation scheme for timing-constrained minimum cost layer assignment. *IEEE Transactions on Circuits and Systems II*, 56(7):580–584, 2009.

[54] Y. Chang, T. Lee, and T. Wang. GLADE: a modern global router considering layer directives. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 319–323, 2010.

[55] T. Lee, Y. Chang, and T. Wang. An enhanced global router with consideration of general layer directives. In *Proceedings of the ACM International Symposium on Physical Design*, pages 53–60, 2011.

[56] M. D. Moffitt and C. N. Sze. Wire synthesizable global routing for timing closure. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 545–550, 2011.

[57] L. Trevillyan, D. Kung, R. Puri, L. N. Reddy, and M. A. Kazda. An integrated environment for technology closure of deep-submicron IC designs. *IEEE Design & Test of Computers*, 21(1):14–22, 2004.

[58] C. J. Alpert, S. K. Karandikar, Z. Li, G. Nam, S. T. Quay, H. Ren, C. N. Sze, P. G. Villarrubia, and M. C. Yildiz. Techniques for fast physical synthesis. *Proceedings of the IEEE*, 95(3):573–599, 2007.

[59] C. L. Ratzlaff and L. T. Pillage. RICE: rapid interconnect circuit evaluation using AWE. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(6):763–776, 1994.

[60] J. Cong and D. Z. Pan. Interconnect delay estimation models for synthesis and design planning. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 97–100. 1999.

[61] C. J. Alpert, J. Hu, S. S. Sapatnekar, and C. N. Sze. Accurate estimation of global buffer delay within a floorplan. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1140–1145, 2006.

[62] D. Papa, T. Luo, M. Moffitt, C. Sze, Z. Li, G. Nam, C. Alpert, and I. Markov. RUMBLE: an incremental timing-driven physical-synthesis optimization algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(12):2156–2168, 2008.

[63] S. S. Sapatnekar. *Timing*. Kluwer Academic Publishers, Boston, MA, 2004.

[64] T. H. Lee and T. C. Wang. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(9):1643–1656, 2008.

[65] A. Kahng, G. Robins, A. Singh, H. Wang, and A. Zelikovsky. Filling and slotting: analysis and algorithms. In *Proceedings of the ACM International Symposium on Physical Design*, pages 95–102, 1998.

[66] K. Li, C. Lee, Y. Chang, C. Su, and J. Chen. Multilevel full-chip routing with testability and yield enhancement. In *Proceedings of the International Workshop on System Level Interconnect Prediction*, pages 29–36, 2005.

[67] H. Chen, S. Chou, S. Wang, and Y. Chang. A novel wire-density-driven full-chip routing system for CMP variation control. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(2):193–206, 2009.

[68] H. Yao, Y. Cai, and X. Hong. CMP-aware maze routing algorithm for yield enhancement. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 239–244, 2007.

[69] Y. Jia, Y. Cai, and X. Hong. Full-chip routing system for reducing Cu CMP & ECP variation. In *Proceedings of the Annual Symposium on Integrated Circuits and System Design*, pages 10–15, 2008.

[70] C. Chu and Y. C. Wong. Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design. In *Proceedings of the ACM International Symposium on Physical Design*, pages 28–35, 2005.

[71] ISPD 2007 Global Routing Contest. http://www.sigda.org/ispd2007/rcontest/. Accessed: 07/27/2008.

[72] G. Nam, C. Sze, and M. Yildiz. The ISPD global routing benchmark suite. In *Proceedings of the ACM International Symposium on Physical Design*, pages 156–159, 2008.

[73] ISPD 2008 Global Routing Contest. http://www.ispd.cc/slides/ispd2008-files/S7-3.pdf. Accessed: 02/07/2013.

[74] ILOG CPLEX 11.210. http://www.ilog.com/products/cplex/. Accessed: 05/03/2009.

[75] Y. Ye, F. Liu, M. Chen, and Y. Cao. Variability analysis under layout pattern-dependent rapid-thermal annealing process. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 551–556, 2009.

[76] J. C. Scott, O. Gluschenkov, B. Goplen, H. Landis, E. Nowak, F. Clougherty, A. Mocuta, T. Hook, N. Zamdmer, C. Lai, M. Eller, D. Chidambarrao, J. Yu, P. Chang, J. Ferris, S. Deshpande, Y. Li, H. Shang, G. Hefferon, R. Divakaruni, E. Crabbe, and X. Chen. Reduction of RTA-driven intra-die variation via model-based layout optimization. In *Proceedings of the IEEE Symposium on VLSI Technology*, pages 152–153, 2009.

[77] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2003.

[78] S. K. Springer, S. Lee, N. Lu, E. J. Nowak, J.-O. Plouchart, J. S. Watts, R. Q. Williams, and N. Zamdmer. Modeling of variation in submicrometer CMOS ULSI technologies. *IEEE Transactions on Electron Devices*, 53(9):2168–2178, 2006.

[79] S. N. Adya and I. L. Markov. Fixed-outline floorplanning through better local search. In *Proceedings of the IEEE International Conference on Computer Design*, pages 328–334, 2001.

[80] S. Adya, H. H. Chan, and I. L. Markov. http://vlsicad.eecs.umich.edu/BK/parquet/. Accessed: 12/22/2008.

[81] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing based module placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 472–479, 1995.

[82] X. Tang and D. F. Wong. FAST-SP: A fast algorithm for block placement based on sequence pair. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 521–526, 2001.