AN INTERCONNECT-DRIVEN SYSTEM-ON-CHIP FLOORPLANNING FRAMEWORK

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Patrick Siu-ying Hung
August 2002

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Michael J. Flynn
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Giovanni De Micheli

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John T. Gill III

Approved for the University Committee on Graduate Studies:

Preface

As VLSI technology reaches deep submicron era, *interconnect properties* instead of *gate properties* plays a dominant role in determining processor performance and power consumption. While this relationship has been known for a number of years, most System-On-Chip (SOC) vendors today still follow the traditional design flow, separating architectural design, logic design and physical design into three distinct stages. Chip floorplanning, which affects architectural and logic design and determines the overall system performance, is often performed only in the physical layout stage. This is one of the main reasons why many SOC's have problems surpassing 400MHz, while custom-made, high-end processors have already exceeded 2.5GHz.

The focus of our research is to develop an interconnect-driven floorplanning framework, supporting effective chip floorplanning in the architectural design stage. In order to estimate the effects of interconnect in the early design stages, we examine interconnect models on three different levels. First, we developed a wirelength distribution model within a functional block. Our proposed wirelength distribution model is more flexible and accurate than the previous models. It can be used to estimate wire load, wire delay and interconnect power consumption within a synthesized functional block. Second, we developed a wire congestion model, identifying congestion hotspots among multiple blocks in a floorplan. The model can be used to model the effects of interconnect coupling in a floorplan. Third, we developed an interconnect-driven processor performance model, which generalizes the relationship between processor performance and interconnect overhead.

The three interconnect models, constituting the basis of our research, were integrated with other design tools previously developed in our research group to form a unified floorplanning framework. Our research shows that system-level floorplanning in the architectural design stage is necessary in the deep-submicron era.

Acknowledgments

First, and foremost, I would like to thank my advisor, Professor Michael J. Flynn, for his superb technical guidance and generous financial support in the past few years. Most importantly, he taught me how to conduct independent research, which will continue to be an invaluable asset for me for the rest of my life. I would also like to express my gratitude to Professor De Micheli and Professor John Gill, who carefully read through this dissertation and provided many constructive comments.

There are many other people to whom I owe a debt of appreciation for their assistance and support to the preparation and completion of this work. I would like to thank Susan Gere, who helped maintain a friendly and stable research environment for all graduate students. I would like to thank the current and previous members of my research group, especially Professor Martin Morf, Professor Sang Bang Choi, Professor Neville Harris, James Bennett, Takahiro Nishiyama, Kevin Rudd, Steve Fu, Grant McFarland, Albert Liddicoat, Hossam Fahmy and Andrew Zimmerman. They have spent countless hours with me discussing on my research, and have proved to be my best sounding board. I would like to thank Luc Séméria, who worked with me on the statistical wirelength distribution model.

Last, but definitely not least, I would like to thank my wife for her unfailing moral support during my years at Stanford. Without whom none of this work would have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 On-Chip Interconnect System

On-chip interconnect system is used for distributing clocks and power supplies, and connecting signals among circuits. Today digital integrated circuit (IC) is composed of transistors fabricated epitaxially on a lightly doped silicon substrate. Alternating dielectric and metal layers are grown on top of the transistors. The metal layers are typically made of copper (Cu), aluminum (Al) or tungsten (W), whereby the dielectric layers are made of silicon dioxide (SiO_2). The state-of-the-art silicon chips may consist of eight or more metal layers ($M_1 - M_8$), forming a three-dimensional interconnect structure (Figure 1.1) [Tai, Uni]. To minimize cross couplings between layers, all interconnections usually run in the same direction within a layer and perpendicularly between adjacent layers [She95].

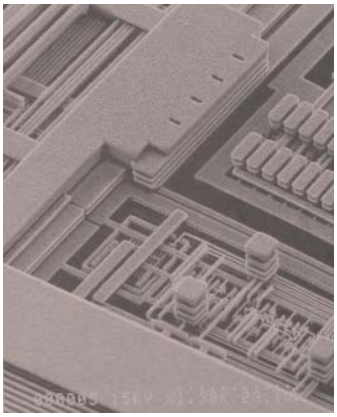


Figure 1.1: Metal interconnect structure of a chip after etching of dielectric layers [ES00]

Figure 1.2 shows the layout diagram of a complementary metal-oxide-semiconductor field-effect transistor (CMOS) inverter. The pMOS transistor is located in an n-doped well, whereas the nMOS transistor is located in a p-doped substrate. In this example, the metal layer M_1 is used for providing power supply and connecting the CMOS inverter input and output signals to the other circuits. Metal contacts, known as *vias*, connects the metal layer to the two transistor gates.

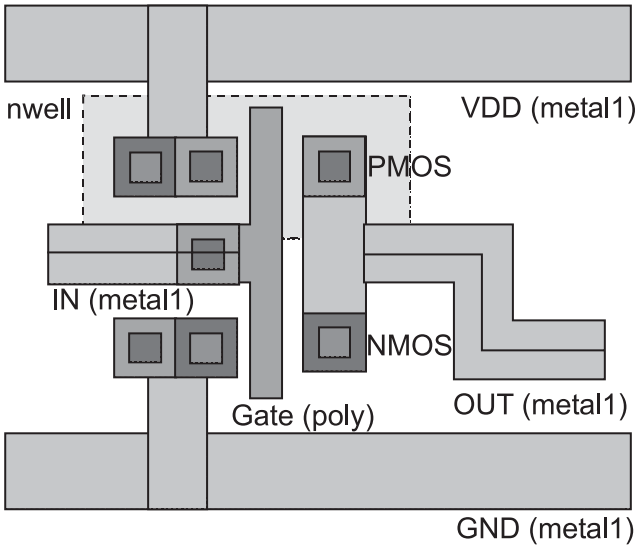


Figure 1.2: Topview of a CMOS Inverter Layout

A three-dimensional (3-D) visualization of the interconnections is shown in Figure 1.3. It is tedious to precisely measure all interconnect characteristics, but researchers have developed approximation models extracting the vital interconnect parameters, such as capacitance and inductance, fairly accurately and efficiently [Bak90].

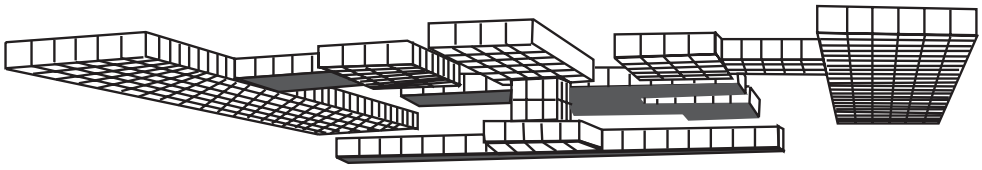


Figure 1.3: 3-D View of the Interconnections in a CMOS Inverter

On-chip interconnections can be broadly divided into the three categories:

- *Local interconnections* are intended for connecting transistors within a functional block. They usually use the minimum pitch to wire up large circuits at local level. These wires have relatively high resistivity and are typically driven by small transistors. Their maximum length scales with feature size and is around $1mm$ at 0.18 μm feature size.

- *Intermediate interconnections* are for spanning large circuit blocks. They normally use a pitch and cross-sectional area greater than the minimum and have a maximum length of around $2 - 5mm$. Unlike local interconnections, these wires do not scale with feature size.

- *Global interconnections* are for distributing power supply and for connecting data busses, control busses and clock signals. They are usually much less dense and have large cross-sectional area to minimize wire delay and power loss. These wires span the entire chip connecting separate functional blocks within the design.

As feature size shrinks, the projected system-on-chip (SOC) die size remains roughly constant in the near future, incorporating more functionalities on a single chip [Ass01]. Thus, global and intermediate interconnection delays are increasingly more dominant relative to gate delays. Usually, the lower metal layers within smaller cross-sectional areas (e.g. $M_1 - M_2$) are used for local interconnections, and the higher metal layers with larger cross-sectional areas (e.g. $M_3 - M_8$) are reserved for intermediate and global interconnections. Figure 1.4 shows the cross-sectional view of these three types of interconnections in an integrated circuit.

1.2 Deep Submicron Interconnect Issues

Deep submicron technology allows designers to build very complicated chips with hundreds of millions of very fast transistors. According to the Semiconductor Industries Association (SIA), the number of transistors and the clock frequencies for high performance microprocessors will continue to grow exponentially in the near future [Ass01] (Figures 1.5 and 1.6).

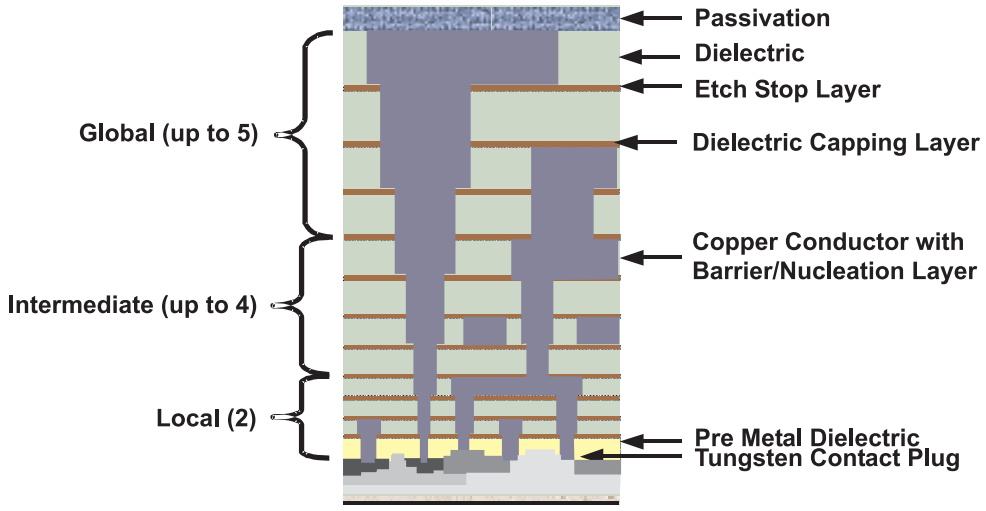


Figure 1.4: Cross-sectional View of the Interconnections in an IC

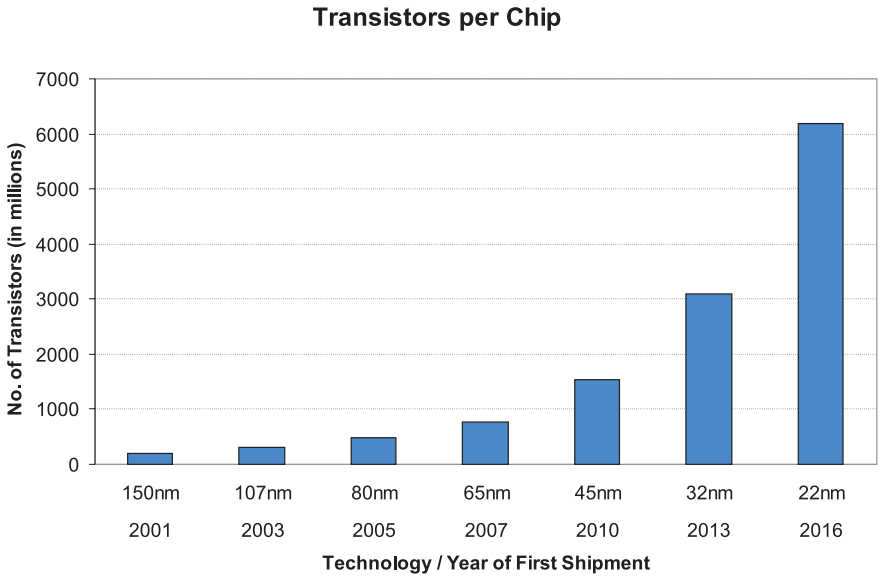


Figure 1.5: Total Transistors per Chip from 2001 to 2016

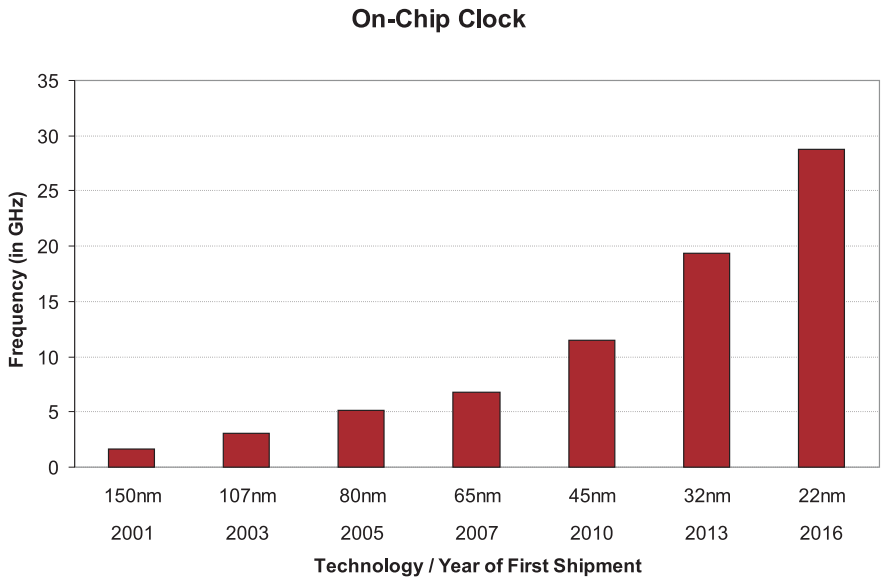


Figure 1.6: On-Chip Clock from 2001 to 2016

However, the interconnect properties do not scale with the feature size, thus limiting the usefulness of the vast number of fast transistors. This section discusses the issues related to the on-chip interconnect system, including performance, power consumption, reliability, manufacturing yield, and signal integrity.

1.2.1 System Performance

Much improvement in chip performance has been due to technology scaling, allowing increased circuit densities at higher clock frequencies. As feature sizes shrink, device area shrinks roughly as the square of the scaling factor, and device delay improves approximately linearly with feature size. Unfortunately, interconnect delay does not scale with the feature size. When all three dimensions of an interconnect wire are scaled down by the same factor, the interconnect delay remains roughly unchanged. Consequently, interconnect delay becomes the performance bottleneck [FHR99].

Systems today are integrated to upward of 10^8 devices per square centimeter. Interconnects — both within a single chip and across chip boundaries — determine the signal latency. Figure 1.7 shows the projections of gate and interconnect delays. While local interconnections scale roughly with the feature size, intermediate and global interconnections

certainly do not. This poses a major challenge in the future system-level design, implying that "interconnect-driven architectures" are becoming crucial.

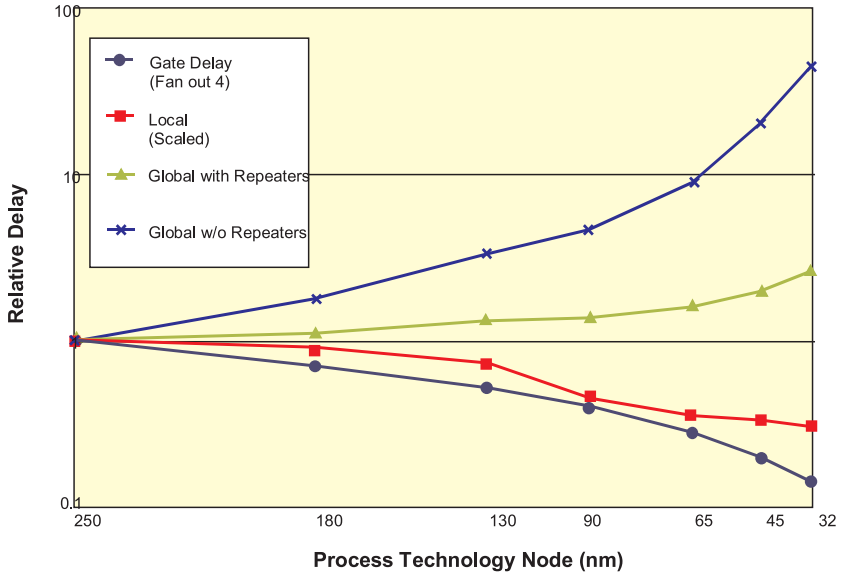


Figure 1.7: Interconnect Delay

A related problem is the fact that the present CAD design tool cannot handle the exponential growth in the number of interconnections. Occasionally the physical synthesis tool cannot meet timing closure due to excessive interconnect delays caused by long wires. This timing closure problem is also known as "wire exceptions." The wire exceptions are a serious problem because they have to be fixed manually by adjusting physical placements or even changing the logic or architectural design. As feature size shrinks, the number of wire exceptions (following the number of wires) is expected to increase exponentially [Ron01], suggesting that a new system-level design methodology is becoming necessary.

1.2.2 Power Dissipation

Power consumption has received much attention recently because of growing demands for wireless and portable electronic appliances. The three main sources of power dissipation in a CMOS circuit (P_{total}) are the switching loss, the direct-path short-circuit current, and

the transistor leakage current, corresponding to the three terms in Equation 1.1 [CSB92].

$$P_{total} \;=\; p_t\,(C_L \cdot V \cdot V_{dd} \cdot f_{clk}) \;+\; I_{sc} \cdot V_{dd} \;+\; I_{leakage} \cdot V_{dd} \tag{1.1}$$

The first term $p_t\,(C_L \cdot V \cdot V_{dd} \cdot f_{clk})$ represents the switching loss, in which C_L is the sum of the interconnect and gate capacitances, V_{dd} is the supply voltage, V is the voltage swing, f_{clk} is the clock frequency, and p_t is the transition probability. In most CMOS design, the voltage swing V and the supply voltage V_{dd} are identical. The second term $I_{sc} \cdot V_{dd}$ represents the short-circuit power loss, in which I_{sc} is the short-circuit current. The third term $I_{leakage} \cdot V_{dd}$ represents the loss caused by leakage current, in which $I_{leakage}$ is the CMOS leakage current.

Among all these terms, interconnect switching loss represents the most important power loss in deep submicron design. In general, the interconnect switching loss may be mitigated by lowering (or dynamically adjusting) supply voltage, switching frequency, or by reducing interconnect capacitance. At the architectural design phase, there is a tradeoff among area, system performance and power consumption [FHR99]. For instance, multiple functional units running in parallel at a sub-threshold voltage level can achieve the same performance as a single functional unit running at a nominal voltage but effectively consume much less power [Bha94]. At the physical design phase, power consumption can be reduced by minimizing interconnect wirelength and wire congestion using different floorplan or different placement and routing algorithms.

1.2.3 Reliability and Manufacturing Yield

Interconnect affects chip reliability and manufacturing yield in a variety of ways, such as process-induced damage, electromigration, and bridging defects caused by particle contamination. As feature size shrinks, long interconnect can create excessive static charge buildup at the gate during the ion and plasma implant processes, causing permanent damages to the gate oxide. This is known as *process-induced damage* [GM92]. For a given manufacturing process, a *process-induced damage rule* (also known as *antenna rule*) is used to define the maximum ratio of interconnect area to transistor gate area.

Electromigration is caused by the excessive transport of the metal atoms when an electric current flows through an on-chip interconnection for a prolonged period of time [Bla69]. Aluminum is especially vulnerable to electromigration due to its low melting point. The

problem can be alleviated by adding alloy elements (e.g. copper), blocking the grain boundary diffusion path. Another related problem is the migration of silicon into aluminum at the metal contacts, known as *contact electromigration*, which arises when the silicon atoms diffuse into aluminum at the interface area where the void created by the migrating silicon is filled by aluminum. In general, both reliability and manufacturing yield decreases with longer interconnection and with higher current density. Recently, researchers have developed models predicting the manufacturing yield based on the interconnect wirelength estimation and layer assignment [CdG01]. The manufacturing yield estimation function may also be used in the CAD tools to drive the physical layout [Isr00].

1.2.4 Signal Integrity

In the past chip designers could tape out designs without bothering with signal integrity closure. However, skipping signal integrity analysis is no longer an option below 0.13-micron process [Pol]. As the design shrinks, the reduced wire spacing results in larger interconnect cross-sectional aspect ratio. Consequently, the cross-capacitance and cross-inductance between neighboring wires would create more coupling noise, significantly undermining the signal integrity. As CMOS gates are voltage driven, when the coupling noise caused by the *aggressor* wire exceeds the noise margin in the *victim* circuit, the entire system becomes unreliable.

Signal integrity can be improved by increasing wire spacing or by adding ground wires between adjacent signals. Nevertheless, these techniques cannot be used in congested routing areas because there is no room to increase wire spacing or to add ground wires. Hence, it is important to spread the wire density evenly as much as possible to avoid congestion hotspots.

1.3 Our Work

1.3.1 Motivation

In the traditional VLSI design flow, there are three separate and distinct optimization phases: architectural optimization, logic optimization and physical optimization (Figure 1.8).

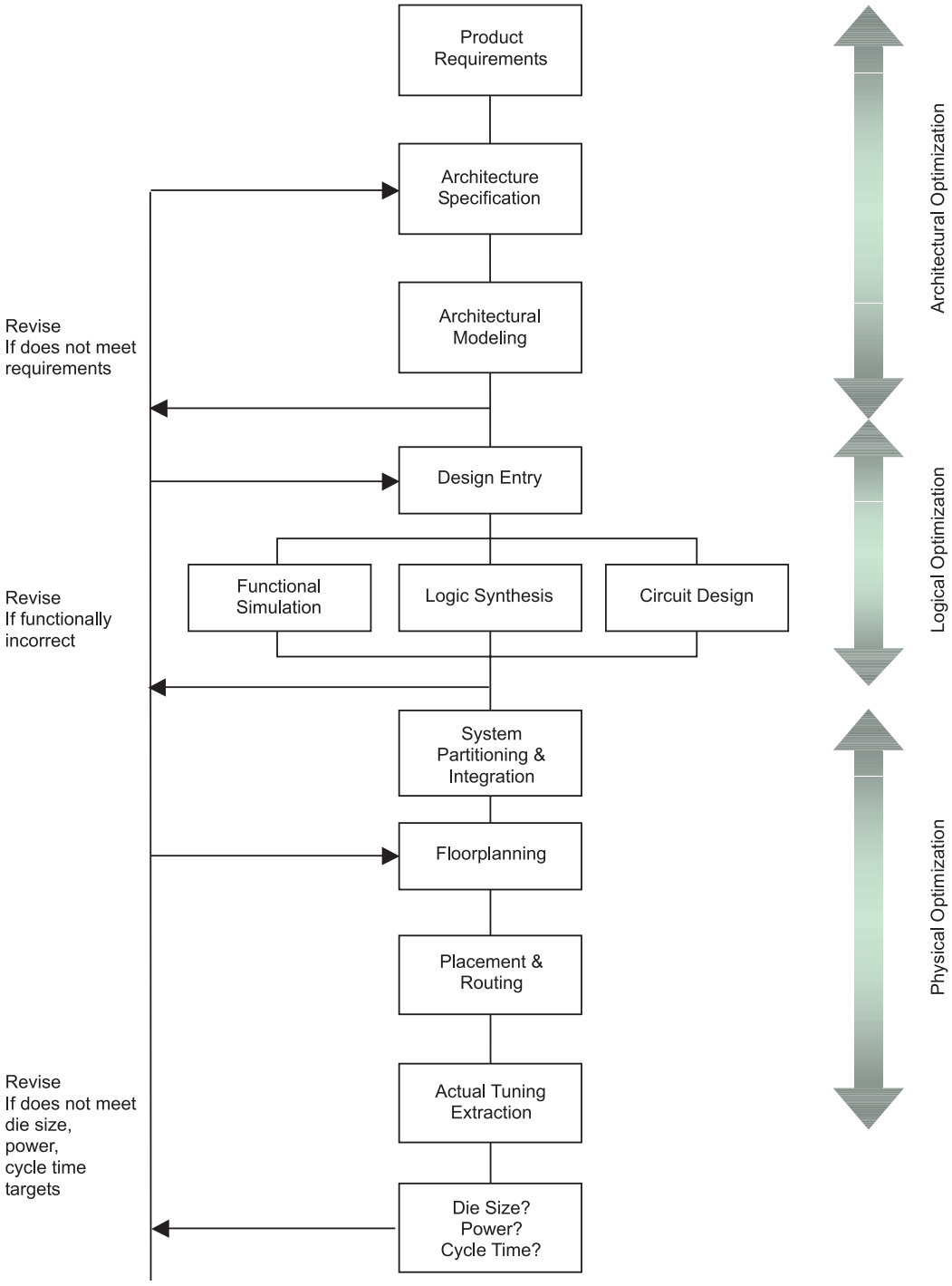


Figure 1.8: Traditional Design Flow

A VLSI design project starts from the architectural design phase, then proceeds to the logic and circuit design phase, and finally to the physical design phase. If design specifications or requirements cannot be met at some point, the project may need to roll back to the previous design phase.

This style of design methodology is analogous to the waterfall model in software design, where requirements analysis, architectural design, detailed design, coding, testing and integration are separated into distinct phases [Roy70]. The software waterfall model is risky in a large and complicated project because it does not sufficiently examine the design details in the early development cycle. As a result, design problems are sometimes uncovered only in the implementation phase when any major changes are extremely costly [Ber97]. In fact, the waterfall model has already been replaced by the rapid prototyping model and other related models, in which programmers build quick prototypes to minimize design risks [Gom83].

Similarly, it is also much cheaper to make changes in a VLSI design in the early design phases. As the design progresses, the costs and lost revenues associated with a major redesign increases exponentially (Figure 1.9). Sometimes late changes are so costly that the entire market opportunity window is lost. To avoid costly and time-consuming chip redesign, it is becoming crucial to obtain an accurate estimate of performance, cost and power consumption in the early design stages and apply these estimates to avoid expensive redesign later on [FH01].

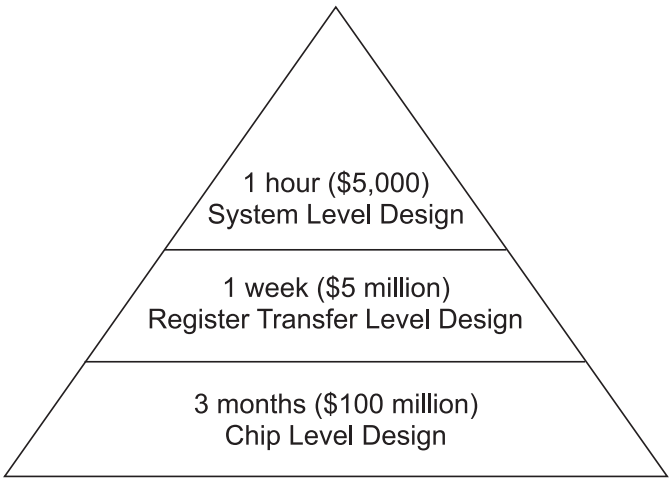


Figure 1.9: Lost Revenue

1.3.2 A Priori Interconnect Models

In this research, we propose an interconnect-driven floorplanning framework, in which designers can build fairly accurate prototypes evaluating system cost, performance, power consumption and other design parameters. As interconnect is becoming the pre-dominant factor in determining all these design parameters, we have developed models to characterize the interconnect properties and examined their architectural impacts.

We have developed three statistical, a priori models — a wirelength distribution model, a wire congestion model and a processor performance model — in our study. Our wirelength distribution model is an extension to the Davis' model [DDM96], whereas our proposed wire congestion model and interconnect-driven processor performance model are derived based on probability theory. The details of these three models are discussed in Chapters 2, 3 and 4, respectively.

An inherent limitation of statistical models is that while these models may accurately predict interconnect properties (e.g. wirelength) for the majority of the wires, they inevitably over-estimate or under-estimate interconnect properties for some of the wires. This is probably unavoidable due to the lack of information in the early design stages. As design progresses, on-line and a posteriori interconnect models can be used in conjunction with a priori interconnect models to improve modeling accuracy [Str99].

1.4 Organization of this Dissertation

- Chapter 1 introduces the effects of interconnect in deep-submicron era and discusses our research objectives.
- Chapter 2 presents a generalized wirelength distribution model.
- Chapter 3 presents an interconnect congestion model and describes a floorplan algorithm that uses the congestion model.
- Chapter 4 presents a processor performance model that takes interconnect overhead into consideration. An upper bound on the optimum issue width (based on interconnect overhead) is described and derived in this chapter.
- Chapter 5 introduces the IPLAN floorplanning framework, which incorporates the interconnect models described in the previous chapters as well as a number of design

tools previously developed in our research group. A case study using Lexra LX4380 is presented to show how the floorplanning framework works.

- Chapter 6 presents the conclusions and discusses about the future work.
- Appendix A shows the derivations of some structural wirelength distributions discussed in Chapter 2.

Chapter 2

Wirelength Distribution Model

Interconnect wirelength is the most important parameter in determining interconnect delay, power loss, reliability and signal integrity. By examining the wirelength distribution of design alternatives, chip designers can more effectively perform architectural tradeoffs, taking the effects of interconnections into consideration [HF00]. This chapter introduces a wirelength distribution model, which is one of the three interconnect models used in the IPLAN floorplanning framework [HSF01].

2.1 Introduction

This introduction section provides the background of the wirelength models developed by other researchers. Sutherland and Oestreicher [SO72] were the first researchers to determine an upper bound of interconnect wirelengths. However, their estimates were overly pessimistic because their model was based on a pure random placement. However, connected blocks are usually placed in close proximity by commercial place-and-route tools. Donath [Don79, Don81] used Rent's Rule [LR71] to derive a tighter upper bound on the expected interconnect wirelength based on a two-dimensional hierarchical placement. Using a similar approach, Masaki and Yamada [MY87] derived the expected interconnect wirelengths for various 2-D and 3-D system packaging structures. Sastry and Parker [SP86] showed that the wirelength distribution in gate arrays follows a Weibull distribution. Later, Gura and Abraham [GA89], Pedram and Preas [PP89], and Stroobandt [Str96] modified and further enhanced the model.

Recently, Davis et al. [DDM98] constructed a wirelength distribution model based on a

non-hierarchical placement. This model is significant because it shows that the wirelength distribution can be derived without explicitly using hierarchical partitioning. Furthermore, the wirelength distribution appears to be more accurate than the previous models for long wires. In our research, we extend Davis' model to investigate the effects of aspect ratio and routing obstacles on interconnect wirelength.

The number of terminals (T) and the number of gates (N) in a VLSI design are related by a simple empirical formula, known as Rent's Rule [LR71].

$$T = kN^p \tag{2.1}$$

In this equation, k is the average number of terminals per gate and p is the Rent's constant. The Rent's constant can be determined empirically and is found to be between 0.5 and 0.75 [Bak90]. Davis et al. investigated a square array of N gates with $\sqrt{N}$ rows and $\sqrt{N}$ columns. The horizontal and vertical gate pitches are both equal to one unit, so the aspect ratio of the entire array is one. A continuous interconnect density function $i(l)$ is defined such that the number of interconnects of length between $l = a$ and $l = b$ is:

$$\int_a^b i(l)dl \tag{2.2}$$

The continuous interconnect density function $i(l)$ consists of three components: a gate pair structural distribution function $M(l)$, a gate pair connection probability $I_{exp}(l)$, and a normalization factor Γ.

$$i(l) = \Gamma M(l) I_{exp}(l) \tag{2.3}$$

The gate pair structural distribution function $M(l)$ can be found by inspection [DDM98], or more easily by using moment generating polynomials [Str96].

$$M(l) = \begin{cases} \frac{l^3}{3} - 2\sqrt{N}l^2 + 2Nl & \text{if } 1 \le l \le \sqrt{N} \\ \frac{(2\sqrt{N}-l)^3}{3} & \text{if } \sqrt{N} \le l \le 2\sqrt{N} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Based on Rent's Rule, Feuer [Feu82] showed that the gate pair connection probability $I_{exp}(l)$ in a two-dimensional array is proportional to l^{2p-4}. The same result was also derived by Davis et al. [DDM98]. According to Donath [Don79], the total number of interconnects

is $\alpha kN(1 - N^{p-1})$. The constant factor α is equal to $f/(f + 1)$, where f is the average fanout of the system. The normalization factor Γ can be found by integrating $i(l)$ over the entire range of wirelength.

$$\Gamma = \frac{\alpha kN(1 - N^{p-1})}{\int_1^{2\sqrt{N}} M(l) l^{2p-4} dl} \tag{2.5}$$

Davis' model assumes a square array of gates, but in practice many blocks cannot be designed and placed with unity aspect ratio. Moreover, there can be many routing obstacles in real designs. It is therefore useful to investigate the effects of aspect ratio and routing obstacles on wirelength distribution. Furthermore, the horizontal and vertical gate pitches can be very different, and it would be useful to examine the effects of the gate pitches.

2.2 Wirelength Distribution Model for Rectilinear Blocks

2.2.1 Effects of Block Aspect Ratio

Consider a rectangular block with m columns and n rows. The total number of gates is $N = m \times n$, and the dimensions of the entire block are X and Y units. Hence, the horizontal gate pitch is $\frac{X}{m}$ units, the vertical gate pitch is $\frac{Y}{n}$ units, and the average gate pitch λ is $\sqrt{\frac{XY}{N}}$ units. Without loss of generality, we assume $X \geq Y$.

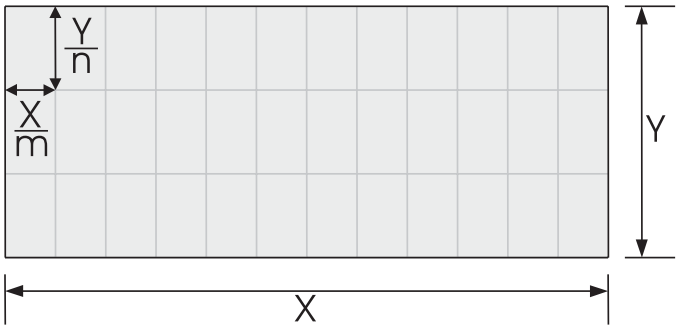


Figure 2.1: Rectangular array of $m \times n$ gates

We follow a similar approach as described in the previous section. We first derive the structural distribution function $M(l)$ and then calculate the gate pair connection probability $I_{exp}(l)$. For each row of gates, we define the horizontal discrete structural distribution

function $M_x(l)$ to be the number of gate pairs with distance l. It can be found by inspection [Str96].

$$M_x(l) = \begin{cases} m & \text{if } l = 0 \\ \frac{2m(X-l)}{X} & \text{if } \frac{X}{m} \leq l \leq X \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

Similarly, for each column of gates, we can define the vertical discrete structural distribution function $M_y(l)$. The expression is similar to $M_x(l)$.

$$M_y(l) = \begin{cases} n & \text{if } l = 0 \\ \frac{2n(Y-l)}{Y} & \text{if } \frac{Y}{n} \leq l \leq Y \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Since m and n are large numbers, we can approximate these discrete structural distribution functions to be continuous. The continuous distribution functions are realistic because the I/O ports may be located anywhere in the gates and the interconnect wirelengths are usually not equal to multiples of the gate pitches. The continuous structural distribution function $M(l)$ is simply the convolution of $M_x(l)$ and $M_y(l)$. To simplify the expression of $M(l)$, we add a constant scaling factor $XY/2N$. This constant is later absorbed in the normalization factor Γ.

$$M(l) = \frac{XY}{2N} \int_0^l M_x(x) M_y(l-x) dx \tag{2.8}$$

Evaluating the definite integral from $x = 0$ to $x = l$, we get the following expression for $M(l)$.

$$M(l) = \begin{cases} \frac{l^3}{3} - (X+Y)l^2 + 2XYl & \text{if } l \leq Y \\ -Y^2 l + XY^2 + \frac{Y^3}{3} & \text{if } Y \leq l \leq X \\ \frac{(X+Y-l)^3}{3} & \text{if } X \leq l \leq X+Y \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

The gate pair connection probability $I_{exp}(l)$ is the expected number of interconnections between a pair of gates that are l units apart. In order to calculate $I_{exp}(l)$, we follow Davis' approach and define block A, block B, and block C, as shown in Figure 2. In general, the

horizontal and vertical gate pitches do not have to be identical, but they are shown to be the same in the diagram. Block A represents a single gate in the rectangular array; block C consists of all the gates with a distance l away from block A; block B consists of all the gates between block A and block C.

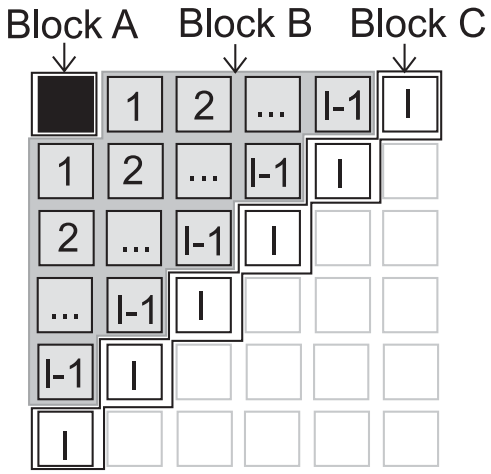


Figure 2.2: Definitions of Blocks A, B, and C

N_A, N_B, and N_C are the number of gates in block A, B, and C, respectively. Using Rent's Rule, $I_{exp}(l)$ can be determined as below [DDM98].

$$I_{exp}(l) \cdot N_C = k((N_A + N_B)^p + (N_B + N_C)^p - {N_B}^p - (N_A + N_B + N_C)^p) \tag{2.10}$$

The number of gates in block A (N_A) is always 1. For short interconnects ($l \leq Y$), N_B is proportional to l^2 and N_C is proportional to l. For longer interconnects ($l \geq Y$), N_B is approximately proportional to l instead of l^2 and N_C is approximately a constant. Using binomial expansion, $I_{exp}(l)$ is proportional to l^{2p-4} for short interconnects and l^{p-2} for longer interconnects. Thus, $i(l)$ can be determined by the following equation.

$$i(l) = \begin{cases} \Gamma M(l) l^{2p-4} & \text{if } l \leq Y \\ \Gamma M(l)(Y \cdot l)^{p-2} & \text{if } Y \leq l \leq X + Y \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

For $Y \leq l \leq X + Y$, a scaling factor Y^{p-2} is added to ensure that $I_{exp}(l)$ is continuous at $l = Y$. As before, the normalization factor Γ can be found by integrating $i(l)$ over the

entire range.

$$\Gamma = \frac{\alpha k N(1 - N^{p-1})}{\int_{\lambda}^{Y} M(l) l^{2p-4} dl + \int_{Y}^{X+Y} M(l)(Y \cdot l)^{p-2} dl} \tag{2.12}$$

Please note that the interconnect density function $i(l)$ does not depend on the gate aspect ratio. Equations 2.9 and 2.11 show that the wire distribution function has three distinct regions: (a) short wires ($l \leq Y$), (b) medium wires ($Y \leq l \leq X$), and (c) long wires ($X \leq l$). If $X = Y = \sqrt{N}$ units (one unit is equal to one gate pitch), there is no medium wire, and our model is similar to Davis' model. The only difference is that $I_{exp}(l)$ is equal to $(Y \cdot l)^{p-2}$ instead of l^{2p-4} for long wires. However, the two models are quite different for non-unity aspect ratios. In particular, the average wirelength estimations are very different. The average wirelength l_{av} is derived in Equation 2.13.

$$\begin{aligned} l_{av} &= \frac{\int_{\lambda}^{X+Y} i(l) \cdot l dl}{\int_{\lambda}^{X+Y} i(l) dl} \\ &= \frac{\int_{\lambda}^{Y} M(l) l^{2p-3} dl + \int_{Y}^{X+Y} M(l)(Y \cdot l)^{p-2} \cdot l dl}{\int_{\lambda}^{Y} M(l) l^{2p-4} dl + \int_{Y}^{X+Y} M(l)(Y \cdot l)^{p-2} dl} \end{aligned} \tag{2.13}$$

If we assume that $I_{exp}(l)$ is equal to l^{2p-4} for all wires (as in Davis' model), the estimated average wirelength estimate decreases with higher aspect ratios. This is incorrect as it contradicts with our experimental results. On the other hand, we set $I_{exp}(l)$ to be l^{2p-4} for short wires and l^{p-2} for medium and long wires in our model. There are relatively more long wires in our model and the average wirelength increases almost linearly with the aspect ratio.

Figure 2.3 compares the average wirelength estimates derived from both models. In this graph, p is 0.67, λ is one unit, and the total area is 10,000 square units. Assume that the area of the rectangular block is fixed. Figure 2.4 shows the theoretical wirelength distributions of the block with various aspect ratios. As before, the Rent's constant p is set to 0.67.

For short wires, the slope of the log-log graph is roughly equal to $2p-3$ (or -1.66) because $M(l)$ is roughly proportional to l (according to Equation 2.9) and $I_{exp}(l)$ is proportional to l^{2p-4}. Similarly for medium wires, the slope is roughly equal to $p - 2$ (or -1.33) because $M(l)$ is roughly constant and $I_{exp}(l)$ is proportional to l^{p-2}. For $p = 0.67$, the two slopes are quite similar and the two regions may not be very distinctive in the graph. The distribution

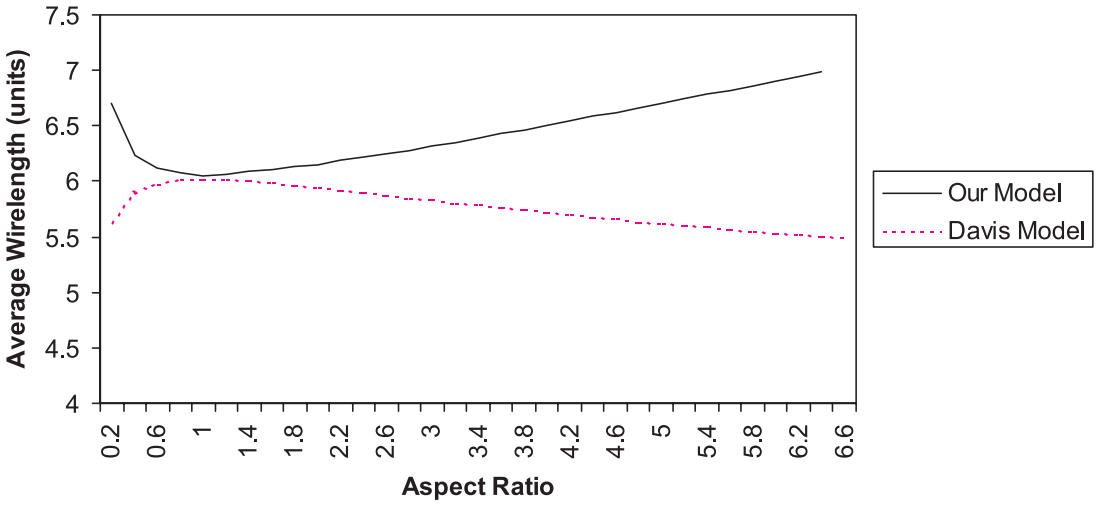


Figure 2.3: Average Wirelength Estimations vs Aspect Ratio

tails off when the wirelength is near $X + Y$.

It is apparent that rectangular blocks with higher aspect ratios contain more long wires, but the wire density function $i(l)$ for short wires remains roughly the same. The number of short wires far exceeds the number of long wires, and the small increase in the number of the long wire does not substantially affect the wire distribution of the short wires.

2.2.2 Effects of Routing Obstacles

The effects of routing obstacles on interconnect wirelength are similar to the effects of aspect ratio. Routing obstacles spread out connected gates and may cause some long interconnections. As in the case of high aspect ratios, obstacles are expected to increase the number of long wires without significantly affecting the number of short wires. Figure 2.5 shows three simple routing obstacle examples. In Figure 2.5(a), the routing obstacle is located at the corner of the block; in Figure 2.5(b), the obstacle is located inside the block; in Figure 2.5(c), the obstacle is located on the edge of the block.

First, we examine the L-shaped block in Figure 2.5(a). The structural distribution function $M(l)$ shown in Table 2.1 may be derived using the moment generating polynomial technique (Appendix A). The gate pair connection probability $I_{exp}(l)$ is derived as shown in the previous section. $I_{exp}(l)$ is proportional to l^{2p-4} for short wires ($l \leq X - Y$) and is

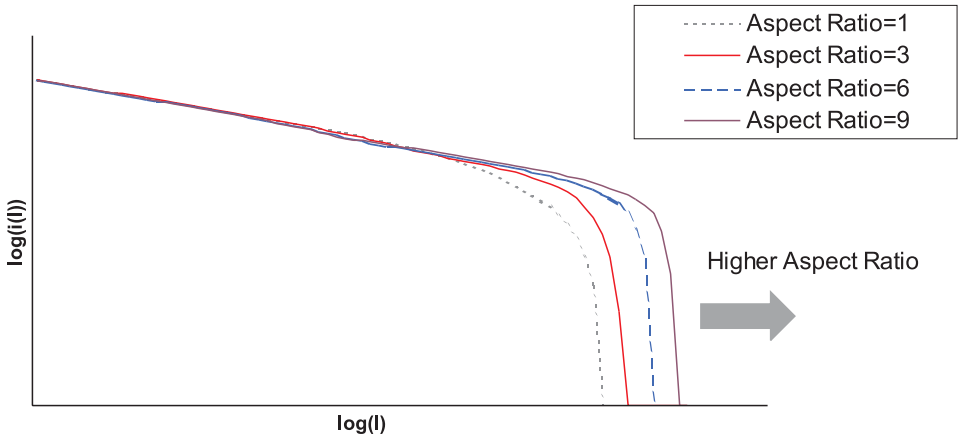


Figure 2.4: Wire Density Function $i(l)$ for Various Aspect Ratios

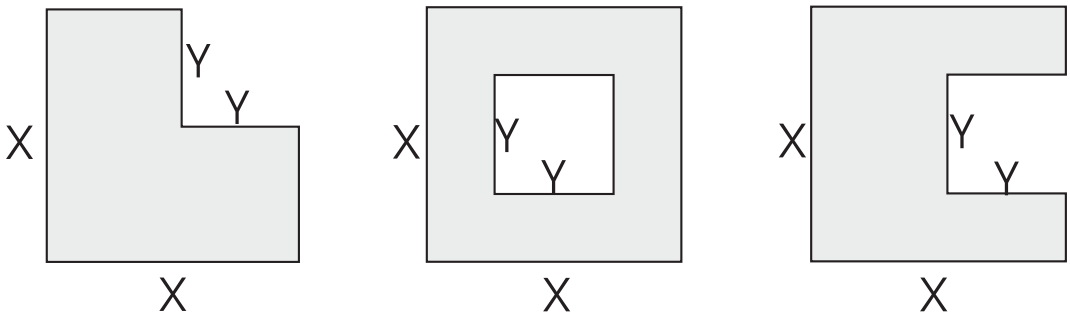
proportional to l^{p-2} for long wires ($l \geq X - Y$). The normalization factor Γ can be found by integrating $i(l)$ from λ to $2X$.

Table 2.1: Structural Length Distribution Function for the L-shaped Block[1]

Wirelength (l)	$M(l)$ for the L-shaped Block
$l \leq X - Y$	$\frac{1}{2}l^3 - 2Xl^2 + \frac{4X^2-4Y^2-1}{2}l$
$X - Y \leq l \leq Y$	$\frac{1}{6}l^3 - (X-Y)l^2 + \frac{6X^2-12XY+6Y^2-1}{6}l + \frac{X^3-X+Y+3X^2Y-9XY^2+5Y^3}{3}$
$Y \leq l \leq 2X - 2Y$	$-\frac{1}{6}l^3 + (2Y-X)l^2 + \frac{6X^2-12XY+1}{6}l + \frac{X^3-X+3X^2Y-9XY^2+6Y^3}{3}$
$2X - 2Y \leq l \leq X$	$-\frac{1}{3}l^3 + Yl^2 + \frac{6XY-3X^2-6Y^2+1}{3}l + \frac{5X^3-2X+Y-9X^2Y+3XY^2+2Y^3}{3}$
$X \leq l \leq 2Y$	$-\frac{1}{3}l^3 + (2X-Y)l^2 + \frac{18XY-15X^2-6Y^2+1}{3}l + \frac{11X^3-2X+Y-15X^2Y+3XY^2+2Y^3}{3}$
$2Y \leq l \leq 2X - Y$	$-\frac{1}{6}l^3 + 2(X-Y)l^2 + \frac{36XY-30X^2+1}{6}l + \frac{11X^3-2X+2Y-15X^2Y+3XY^2-2Y^3}{3}$
$2X - Y \leq l \leq X + Y$	$\frac{1}{6}l^3 - Yl^2 + \frac{12XY-6X^2+6Y^2-1}{6}l + \frac{3X^3+Y-3X^2Y-3XY^2-Y^3}{3}$
$X + Y \leq l \leq 2X$	$-\frac{l^3}{6} + Xl^2 - 2X^2l + \frac{8X^3-2X+1}{6}$

[1]We assume $X \leq 2Y$ in this table.

As the expression of the wire distribution is quite complicated, we would like to approximate the L-shaped block by a rectangular block for interconnect wirelength estimation so that the wirelength distribution can be more easily calculated in software by the CAD tools. First, we examine the wire distribution of an L-shaped block and compare the L-shaped block with a rectangular block of the same area ($X^2 - Y^2$). In order to match the longest



(a) L-shaped Block (b) O-shaped Block (c) C-shaped Block

Figure 2.5: Routing Obstacle Examples

wirelengths in these two blocks ($l_{max} = 2X$), the aspect ratio of the rectangular block is set to $\frac{X+Y}{X-Y}$. Figure 2.6 compares the two wirelength distributions and shows that the two distributions are almost identical. For long wires ($l \approx 2X$), the wire density function $i(l)$ of the rectangular block is twice the wire density function of the L-shaped block. This is because there are two "diagonals" in the rectangular block, and the connections across these two "diagonals" can both constitute long wires. On the other hand, there is only one long "diagonal" in the L-shaped block.

Likewise, we can derive the wirelength distribution functions of the O-shaped block in Figure 2.5(b) and the C-shaped block in Figure 2.5(c). The derivations are shown in Appendix A. For $Y \leq X/2$, the theoretical wirelength distributions of these two blocks are both very similar to the L-shaped block. As in the rectangular block, the numbers of long wires ($l \approx 2X$) in the O-shaped and C-shaped blocks are about twice the number of long wires in the L-shaped block because there are two long "diagonals" in the O-shaped and C-shaped blocks. For $Y > X/2$, the maximum wirelength in C-shaped block is equal to $X + 2Y$, which is longer than the maximum wirelengths in the L-shaped and O-shaped blocks. In general, it is better to place routing obstacles near the corner in order to reduce the number of long wires.

2.3 Experimental Results

Our model was verified with some previously published data and three other design examples. The three design examples are Cordic, Multiplier, and IDCT.

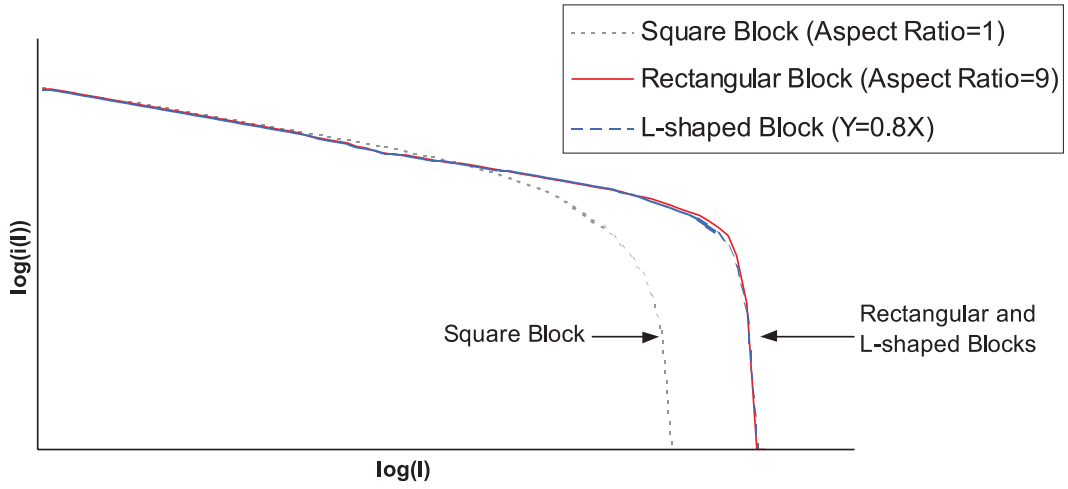


Figure 2.6: Wirelength Distributions of L-shaped and Rectangular Blocks

Table 2.2: Experimental Results on Average Wirelength Estimation

Design	No. of gates	No. of conns	p	Area in mm^2 for aspect ratio=1
Cordic	2151	3384	0.75	0.28
Multiplier	6367	16050	0.72	1.3
IDCT	12666	35193	0.63	4.2

The Multiplier and Cordic designs were synthesized using Synopsys Design Compiler, and the IDCT design was synthesized using Synopsys Behavioral Compiler. The netlists of these designs were generated for TSMC 0.35μm technology. The floorplanning, placement, and routing of the designs were performed by Epoch from Duet Technologies, and the wirelength extractions were done by Cadence Dracula. In this section, we first investigate the average wirelength and then examine the wirelength distribution.

2.3.1 Average Wirelength Model

Our average wirelength estimates were derived from Equation 2.13, and were then compared with Donath's estimates [Don79] using the experimental data cited in his paper. In Donath's experimental data, the aspect ratios of his blocks are all approximately one. The results

are shown in Table 2.3. Although our estimates are still higher than the actual data, they are substantially more accurate than Donath's estimates. For example in Graph B, the estimation error is reduced by about half. Since the aspect ratios are close to one, our estimated average wirelengths are similar to Davis' model. We also compared our average wirelength estimation with Stroobandt's model [Str96] and the two results are comparable. Our estimation is better in Graphs C and F, and worse in Graphs B, D, and E (Graph A is not available in [Str96]).

Table 2.3: Experimental Results on Average Wirelength Estimation

Graph	No. of gates	No. of conns	p	Donath's formula	Stroobandt's estimate	Our estimate	Actual length
A	60	100	0.67	2.76	---	2.28	1.29
B	528	1007	0.59	4.02	2.88	3.09	2.15
C	576	1111	0.75	5.26	4.13	3.92	2.85
D	671	1670	0.57	4.07	2.89	3.13	2.63
E	1239	2687	0.47	3.76	2.45	2.97	2.14
F	2148	7302	0.75	7.37	5.74	5.29	3.50

Next, we compare the theoretical average wirelength with the actual average wirelength in the *Cordic* design for a number of aspect ratios. We calculate the wirelength density function and determine from the slope of the function that the Rent's exponent p is around 0.72. As shown in Figure 2.7, the theoretical results closely match the actual data for a wide range of aspect ratios. The average wirelength increases when the aspect ratio increases, and the slope of the graph is around $2.5\mu m$.

We did the same comparison for *Multiplier*, and the results are shown in Figure 2.8. The theoretical results are also close to the actual data. However, if we assume the total area of the block is fixed and then use the aspect ratio to estimate the average wirelength, the estimates are not as accurate (Figure 2.8). The reason is that the area of *Multiplier* changes considerably with different aspect ratios (Figure 2.9), which is different from the *Cordic* example in which the area remains roughly constant from 0.25 aspect ratio to 6.0 aspect ratio.

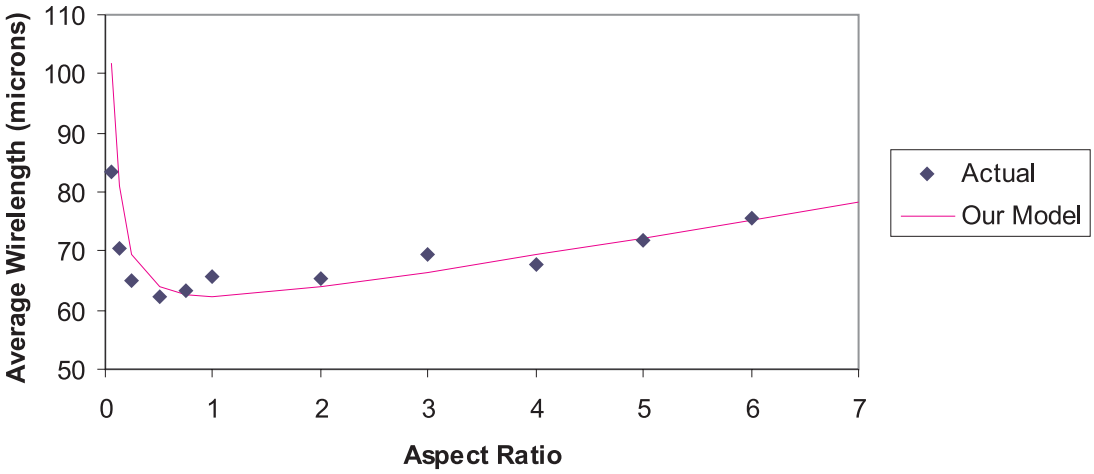


Figure 2.7: Average Wirelength vs Aspect Ratio (Cordic)

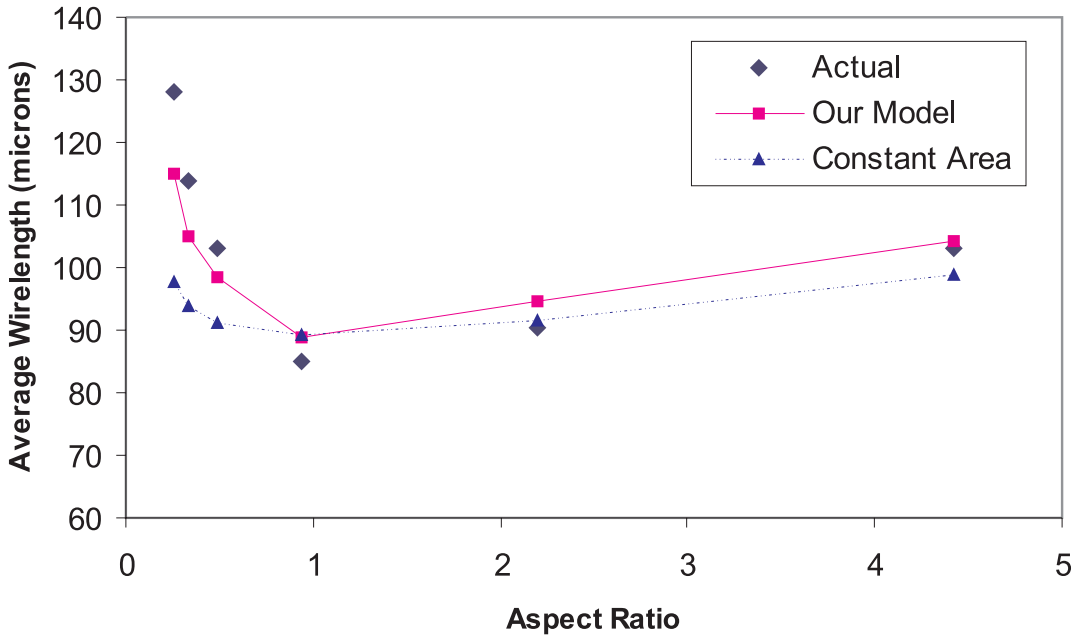


Figure 2.8: Average Wirelength vs Aspect Ratio (Multiplier)

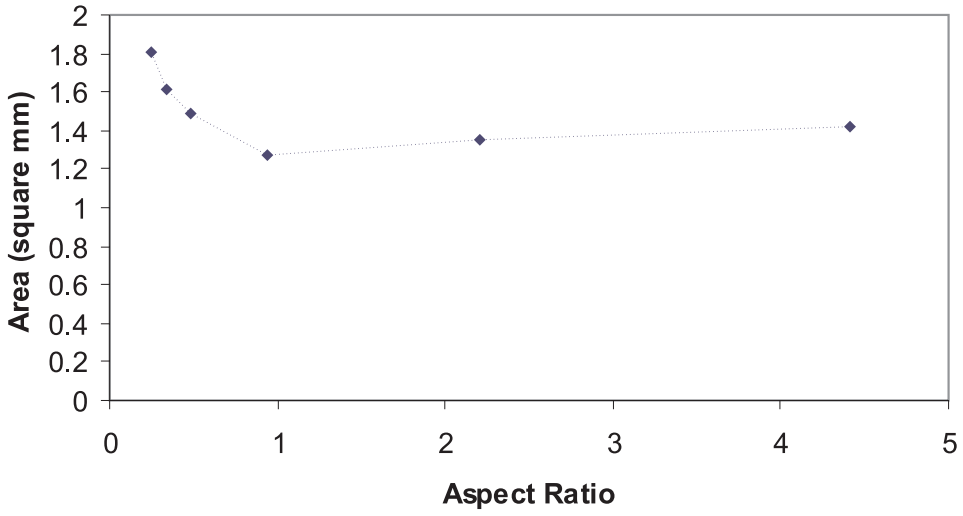


Figure 2.9: Area vs Aspect Ratio (Multiplier)

2.3.2 Wirelength Distribution Model

In this section, we examine the effects of aspect ratio and routing obstacles on wiring distribution. In order to calculate the wirelength density of a design, we need to construct a histogram of all wirelengths. Since the number of short wires is substantially higher than the number of long wires, it is sensible to have larger histogram bins for long wires and smaller histogram bins for short wires. Thus, our histograms are constructed in logarithmic scale. If a certain histogram bin contains no wire, this bin is then merged with an adjacent bin and the wire density is distributed evenly between the two bins. This is to reduce the number of bins and minimize the noise effects when the number of wires in a bin is too small.

Figure 2.10 shows the wirelength distribution of IDCT with unity aspect ratio. Our theoretical wire density is similar to Davis' model, and it describes the actual wire density accurately. We change the aspect ratio of IDCT and measure the new wire density function. The general shape of the graph is same as before, but the number of long wires increases. The wire density is displayed in Figure 2.11. We also show the previous theoretical wire density in the same graph for comparison. To highlight the differences between the two

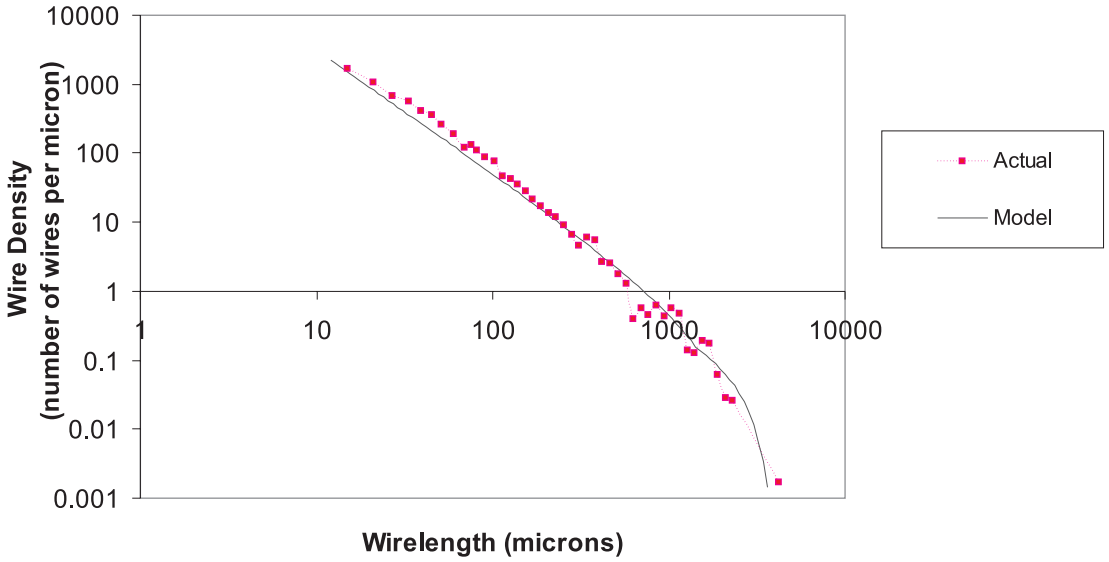


Figure 2.10: Wirelength Distribution with Unity Aspect Ratio (IDCT)

density functions near $1000\mu m$, the graph is drawn in semi-log scale instead of log-log scale.

Similarly, we add a routing obstacle in IDCT to make it an L-shaped block, as in Figure 2.5(a). The X and Y dimensions are approximately $3.2mm$ and $2.3mm$, respectively. As discussed in the previous section, the wire density can be approximated by a rectangular block with aspect ratio ≈ 6. Figure 2.12 shows the wire density of the L-shaped block. We also add the previous two theoretical wire densities (aspect ratio=1, and aspect ratio=6) for comparison. As expected, the wire densities in Figure 2.11 and 2.12 are very similar.

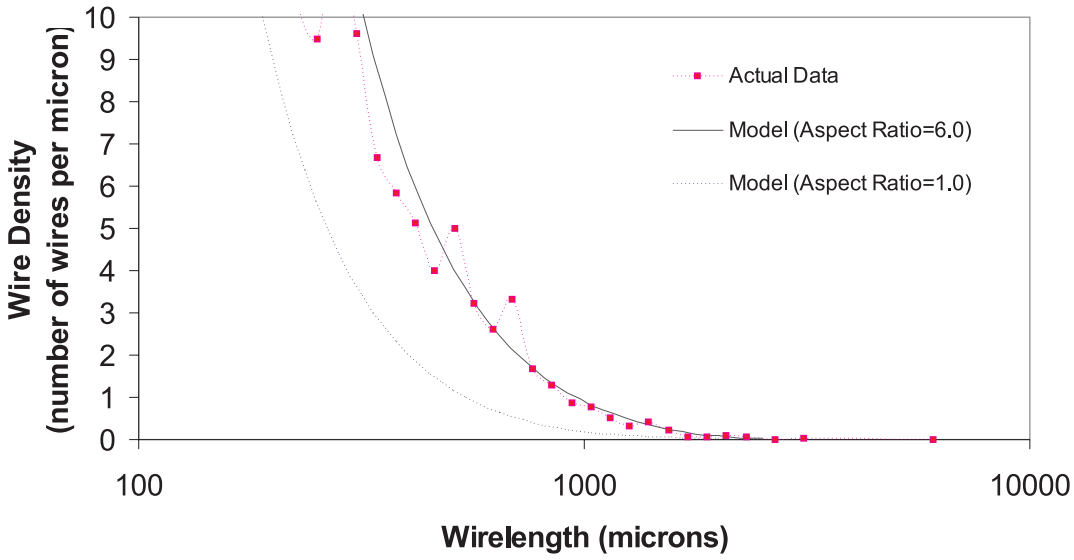


Figure 2.11: Wirelength Distribution with Aspect Ratio = 6.0 (IDCT)

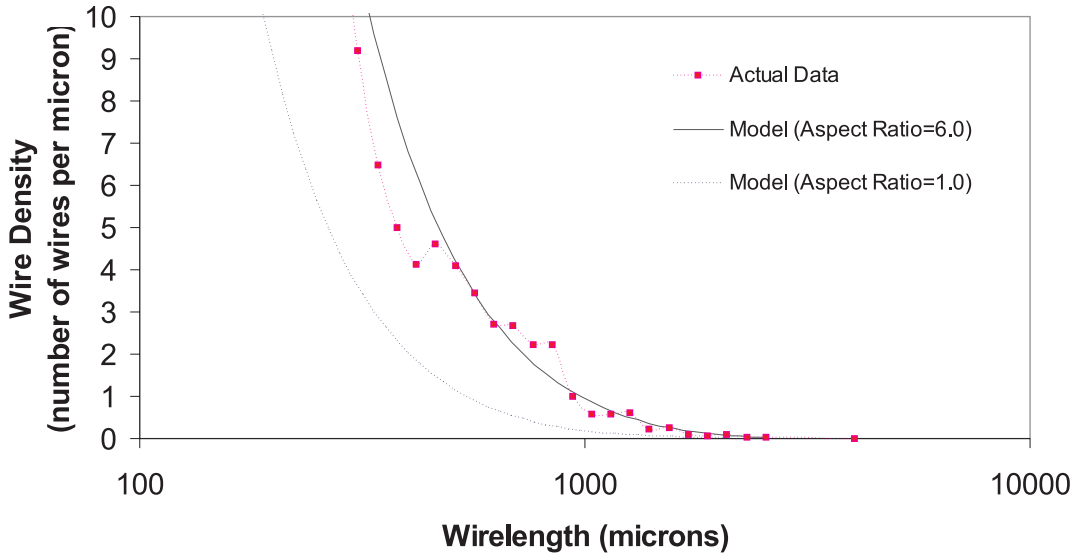


Figure 2.12: Wirelength Distribution with Routing Obstacles (IDCT)

2.4 Summary

This chapter presents a generalized interconnect wirelength distribution model and investigates the effects of aspect ratio and routing obstacles on interconnect wirelength. This model is important because interconnect wirelength is the most important parameter in determining the performance and power consumption of a chip in deep submicron design. The wirelength distribution (Equation 2.11) is divided into three wirelength regions: short wires ($l \leq Y$), medium wires ($Y \leq l \leq X$), and long wires ($X \leq l$) where l is the wirelength, X and Y are the width and height of the rectangular block. As described in Chapter 1, long wires can affect chip performance and cause wire exceptions during physical synthesis process. On the other hand, the power consumption is usually determined by the average wirelength (or total wirelength).

The wirelength distribution depends primarily on the Rent's exponent, the dimensions of the blocks, and the total number of gates in the block. To the first order approximation, the wirelength distribution is not sensitive to the number of rows or columns of gates in the block, or the aspect ratios of the gates. In order to accurately model the wirelength distribution, we determine that it is also important to estimate the effects of aspect ratios on the synthesized area. The area is usually smaller with unity aspect ratio, but the precise effects depend on the type of synthesized block, the synthesis tool and the cell library used. In general, these effects appear to be more serious for a datapath than for a controller block. As shown in the *Multiplier* example, if the changes in the synthesized area are not taken into consideration, the wirelength distribution results can be inaccurate when the aspect ratio is not unity. This is because the size of a datapath can be significantly larger when it is synthesized into an "inefficient" aspect ratio.

Chapter 3

Wire Congestion Model

Besides long interconnections, wire congestion is another main contributor of excessive wire delay and power loss. This chapter introduces a generalized statistical wire congestion model and a heuristic floorplanning algorithm that minimizes the wirelength and congestion in a floorplan [HF99]. The wire congestion model is described in Section 3.2, while the floorplanning algorithm is described in Section 3.3.

3.1 Introduction

As feature size shrinks, wires are becoming narrower and are placed closer to one another. At the same time, they are also becoming taller in order to reduce resistivity. The net outcome is that the coupling effects between neighboring wires (due to cross-capacitance and cross-inductance) are rising. To avoid excessive coupling, critical wires have to be placed farther apart or be shielded with ground wires [Naf99]. Nevertheless, this would not be possible when routing channels in a floorplan are already overly congested. Thus, interconnect congestion affects not only routability but also the area, performance, power consumption, and signal integrity of a design.

Congestion may be a global measure, such as excessive nets crossing a cutline between two partitions, or a local measure when the track density is too high within a routing channel. Congestion based algorithms have proved effective in global routing [Nai87, HS90, LSW94] and global placement [PBS98], in reducing total chip area, improving wireability, and predicting total wire length in the early design stages. In a typical placement tool, congestion is modeled by a rat's nest [She95] or similar connectivity visualizations [Leb83].

For simplicity, these congestion visualization schemes do not model *via minimization* [Hsu83, NMN87, RKN89] and routing obstacles. As a result, they may under-estimate routing congestion in some regions and over-estimate routing congestion in some other regions.

3.2 Wire Congestion Model

3.2.1 Problem Formulation

Simplified Congestion Model

In the simplified congestion model, a chip layout is modeled by a grid graph [She95]. The grid graph $G = (V, E)$ contains vertex at each grid point as well as horizontal and vertical edges connecting adjacent vertices. The horizontal and vertical spacings between two grids are denoted by $\triangle_x$ and $\triangle_y$, respectively. The layout may be considered as a collection of $\triangle_x \times \triangle_y$ rectangular tiles arranged in a $w \times h$ array.

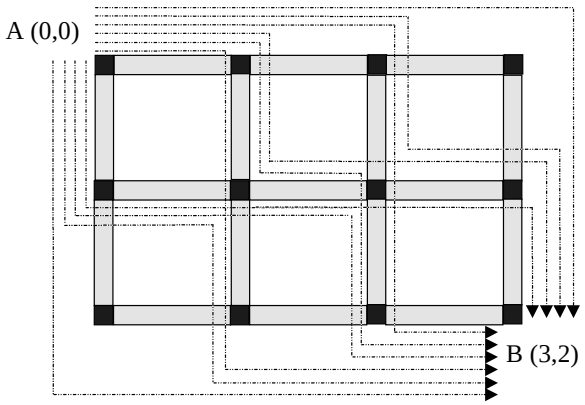


Figure 3.1: Routing between $A(0,0)$ and $B(3, 2)$

Now, consider a route from point A at $(0, 0)$ to point B at (m, n). We assume there is no routing obstacle between A and B in this section. At each grid point, the route can either go horizontally or vertically. The shortest distance between A and B is $m + n$, which is the Manhattan distance between the two points. If the routing is limited to the Manhattan distance, there are $\binom{m+n}{m}$ different ways to route from A to B. The combination is derived from the fact that m tracks out of the $m + n$ tracks are horizontal. As shown in Figure 3.1, there are $\binom{5}{2} = 10$ different ways to route from A to B for $m = 3$ and $n = 2$.

Suppose all possible routes are equally probable. Let P_{xy} be the probability that the

route passes through the point (x, y). Since there are $\binom{x+y}{x}$ different routes from (0,0) to (x,y) and $\binom{m-x+n-y}{n-y}$ different routes from (x,y) to (m,n), we get:

$$P_{xy} = \frac{\binom{x+y}{x} \cdot \binom{m-x+n-y}{n-y}}{\binom{m+n}{m}} \tag{3.1}$$

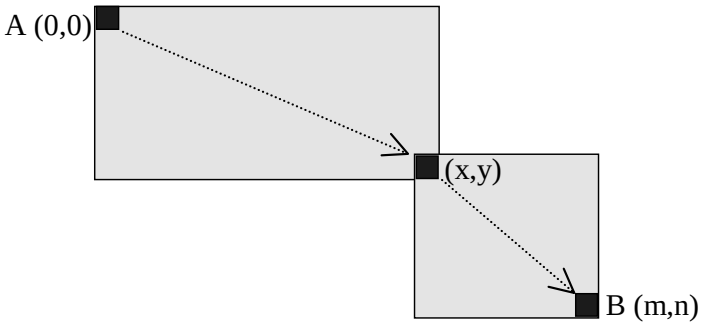


Figure 3.2: Routing Probability through a Point

Similarly, let $P_{xy}{}'$ be the routing probability that the route passes through the horizontal track from (x, y) to $(x + 1, y)$, and $P_{xy}{}''$ be the routing probability that the route passes through the vertical track from (x, y) to $(x, y + 1)$. We get the following equations.

$$P_{xy}{}' = \frac{\binom{x+y}{x} \cdot \binom{m-x-1+n-y}{m-x-1}}{\binom{m+n}{m}} \tag{3.2}$$

$$P_{xy}{}'' = \frac{\binom{x+y}{x} \cdot \binom{m-x+n-y-1}{m-x}}{\binom{m+n}{m}} \tag{3.3}$$

In Figure 3.1, six out of ten possible routes pass through the horizontal track from $(0, 0)$ to $(1, 0)$. Thus, $P_{00}{}'$ is equal to 0.6. The routing probabilities of all the horizontal and vertical tracks are shown in Figure 3.3. Suppose M wires are connected between A and B. The routing densities are defined as the product of the number of wires and the routing probabilities. Let $D_H(x, y)$, $D_V(x, y)$ be the horizontal and vertical routing densities at $(x.y)$. We get the following equations.

$$D_H(x, y) = M \cdot P_{xy}{}' \tag{3.4}$$

$$D_V(x,y) = M \cdot {P_{xy}}'' \tag{3.5}$$

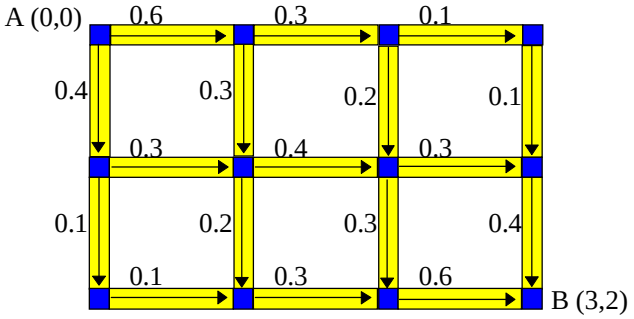


Figure 3.3: Simplified Congestion Model Example

The simplified congestion model ignores many practical considerations. The following sub-sections address the issues of *via minimization*, routing between macro blocks, multi-pin nets, multiple metal layers, and routing obstacles.

Via Minimization

As described in Chapter 1, the interconnections usually all run in the same direction within a layer and perpendicularly between adjacent layers. This methodology supports many compact and efficient routing algorithms, and also serves the purpose of minimizing cross couplings between adjacent layers. To make a 90° turn (or bend) in an interconnect routing, a metal contact (also known as *via*) is needed to connect two perpendicular interconnections from one metal layer to another metal layer. Most routing algorithms minimize the number of vias because vias can cause substantial interconnect delay [Hsu83, NMN87, RKN89]. Vias between metal layers for aluminum wires are made of tungsten, and are fairly resistive. For example in a 0.25-μm process, the resistance of a $M_1 - M_2$ via is about 5Ω and the resistance of a via from M_5 down to the substrate can be more than 20Ω! Considering a 1-μm-wide, 1-mm-long M_5 line has a resistance of only 20Ω, these resistances are very high [Ron01].

In Figure 3.1 there are ten possible ways to route from $(0,0)$ to $(3,2)$ with the shortest path length. Let W_n be the number of routes with n vias (90° bends). In this example, only two routes have a single via. Thus, W_1 is 2 by definition. Similarly, W_2 is 3, W_3 is 4, and W_4 is 3. In order to model *via minimization*, routes with more vias are assumed to be less probable than routes with fewer vias. We use a parameter α between 0 and 1 to model this behavior. Let $Prob\{n\ vias\}$ be the probability that the route has n vias. We define α

as below.

$$\frac{Prob\{n+1\ vias\}}{Prob\{n\ vias\}} = \alpha \cdot \left(\frac{W_{n+1}}{W_n}\right) \tag{3.6}$$

For simplicity, α is assumed to be constant for all n. When α is 1, all routes are equally probable and the model is identical to the simplified congestion model. When α is 0, only routes with the minimum number of vias is allowed. In our experiments, α is set to 0 for busses and very long interconnections, and α is set to 0.5 for local interconnections. These configurations appear to model our designs fairly accurately.

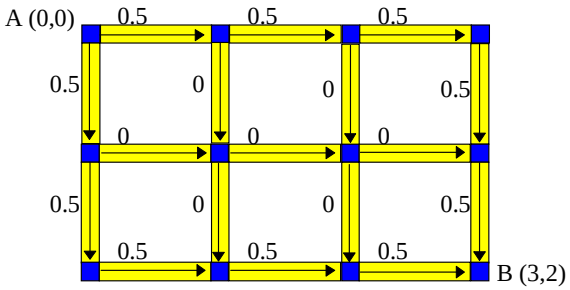
Figure 3.4 shows the routing probabilities when $\alpha = 0$, 0.5, and 1. As α gets smaller, the routing densities near the perimeter of the boundary box get higher. This example illustrates that the simplified model under-estimates the routing congestion near the perimeter of the boundary box, and over-estimates the routing congestion in the middle of the bounding box.

Routing Between Macro Blocks

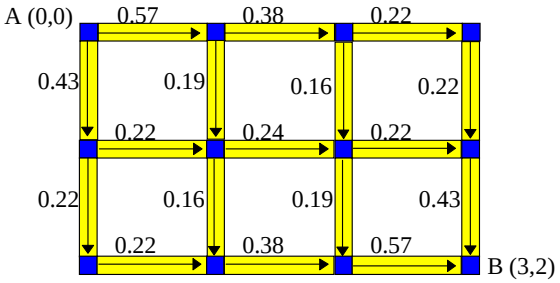
Suppose there are M wires between block **A** and block **B**. Traditional routing algorithms assume the wires come from the centers of the two blocks. This sometimes creates unrealistic high congestion near the centers of the blocks. There are several different ways to model routing densities. In the early design stages, the locations of the pins are unknown and the pins are modeled to be evenly distributed among all the tiles within the macro block. In the later design stages, the pins may be assigned to a certain edge and the pins are modeled to be evenly distributed among all the tiles that cover the edge. Finally, if the exact pin locations are known, the routing densities may be calculated using the congestion model between points.

In Figure 3.5, block **A** is divided into 6 *tiles* (2 grid lines $\times$ 3 grid lines) and block **B** is divided into 8 *tiles* (4 grid lines $\times$ 2 grid lines). Assuming that there are M wires connected between the two blocks. If the end points can be located anywhere within the macro blocks, there can be 6×8 or 48 different combinations of start and end points. Thus, we model each of these combinations carries $M/48$ wires. The routing densities for all these combinations are calculated and then superimposed to give the routing densities.

(a) $\alpha = 0.0$



(b) $\alpha = 0.5$



(c) $\alpha = 1.0$

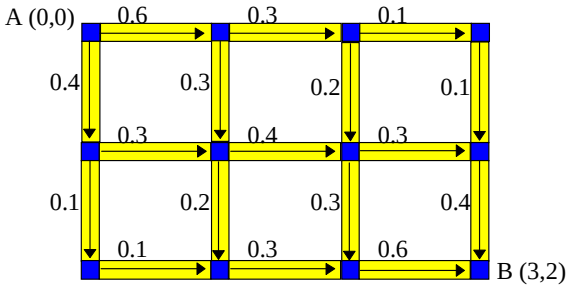


Figure 3.4: Via Minimization Example

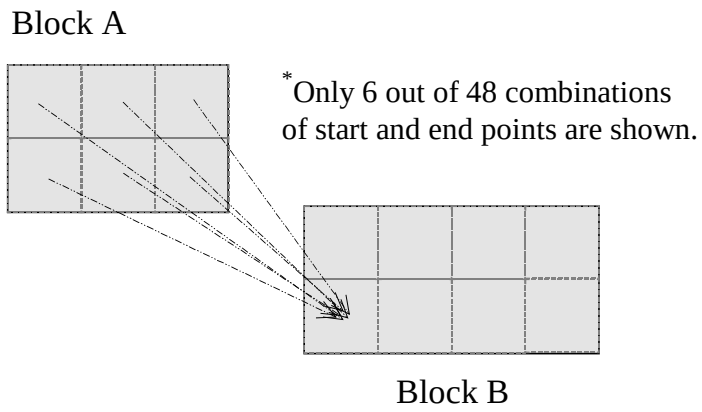


Figure 3.5: Wire Congestion Routing between Blocks Example

Multi-Pin Nets

The earlier part of this section focuses on how to model routing congestion caused by two-pin nets, and this sub-section describes how to model routing congestion caused by multi-pin nets. A simple way to model multi-pin nets is to decompose them to two-pin nets. There are a number of approaches to decompose a multi-pin net. One way is to assume all possible interconnections among the pins are equally possible. Thus, a N-pin net is decomposed into all possible $\frac{N\cdot(N-1)}{2}$ two-pin nets, and each two-pin net carries $\frac{2}{N}$ wires. The number $\frac{2}{N}$ is derived from the fact that only $(N-1)$ out of $\frac{N\cdot(N-1)}{2}$ two-pin nets are required to connect the N points together, so each two-pin net only carries $\frac{2}{N}$ wires.

If the routing algorithm is known, the congestion model can follow the same scheme to better predict the congestion. For example, if the actual router uses minimum spanning trees (MST) to decompose multi-pin nets to two-pin nets [Pri57], the congestion model should follow the same algorithm and use MST in the decomposition.

Multiple Metal Layers

Most modern chips utilize multiple metal layers for global and detailed routing. Typically, each metal layer has different metal width and spacing requirements. Assume every metal layer consists of either all horizontal tracks or all vertical tracks. Now, consider a metal layer with only horizontal tracks. Let $\triangle_x$ and $\triangle_y$ be the horizontal and vertical spacings between 2 grids, W be the metal width, and S be the metal spacing. The number of

horizontal routing channels (R_H) at each grid point is:

$$R_H = \frac{\triangle_y}{(W+S)} \tag{3.7}$$

Similarly, consider a metal layer with only vertical tracks. The number of vertical routing channels R_V at each grid point is:

$$R_V = \frac{\triangle_x}{(W+S)} \tag{3.8}$$

The horizontal and vertical channel supplies of all the metal layers are added together to obtain the horizontal and vertical channel supplies at each grid point. Routing congestion occurs when routing density exceeds channel supply, i.e., $D_H > R_H$ or $D_V > R_H$. When that happens, the routing channel is treated as a routing obstacle. On the other hand, if both horizontal and vertical tracks are permitted in every metal layer, the horizontal and vertical routing channels can be combined. In this case, routing congestion occurs when $D_H + D_V > R_H + R_V$.

3.2.2 Algorithm

This section presents a heuristic algorithm to calculate the routing probabilities based on the statistical congestion model. First, we describe how to calculate the routing probabilities between two points. The algorithm is later extended to handle routing between blocks and with routing obstacles.

Routing between points

The routing probabilities of all the tracks between point A and point B can be found by using the following step-by-step heuristic algorithm. The detailed derivation of the algorithm can be found in [HF97]. Following the derivation in [HF97], set β as in Equation 3.9.

$$\beta = \frac{2\alpha}{1+\alpha} \tag{3.9}$$

Conceptually, the constant β is the conditional probability that via minimization algorithm is not active in a routing. When β is close to 0, the routing always tries to go straight at each point because it "remembers" its previous direction. When β is close to 1, the

routing does not contain any state or memory, resembling the simplified congestion model.

Step 1: Set the probability at point A (P_{00}) to 1.0.

Step 2: Calculate the probabilities at point A.

$$\begin{aligned} P_{00}{}' &:= P_{00} \cdot \tfrac{m}{m+n} \\ P_{00}{}'' &:= P_{00} \cdot \tfrac{n}{m+n} \end{aligned}$$

Step 3: Calculate the probabilities in the first row.

$$\begin{aligned} P_{10} &:= P_{00}{}' \\ P_{10}{}' &:= P_{10} \cdot \beta \cdot \tfrac{m-1}{m+n-1} + P_{00}{}' \cdot (1-\beta) \\ P_{10}{}'' &:= P_{10} \cdot \beta \cdot \tfrac{n}{m+n-1} \\ &\ldots \end{aligned}$$

Step 4: Calculate the probabilities in the second row.

$$\begin{aligned} P_{01} &:= P_{00}{}'' \\ P_{01}{}' &:= P_{01} \cdot \beta \cdot \tfrac{m}{m+n-1} \\ &\ldots \end{aligned}$$

Step 5: Repeat Step 4 for the remaining rows.

Step 6: After all the probabilities have been calculated, do the following to ensure symmetry between A and B.

$$\begin{aligned} P_{xy} &:= \tfrac{1}{2}\left[P_{xy} + P_{(m-x)(n-y)}\right] \\ P_{xy}{}' &:= \tfrac{1}{2}\left[P_{xy}{}' + P_{(m-x-1)(n-y)}{}'\right] \\ P_{xy}{}'' &:= \tfrac{1}{2}\left[P_{xy}{}'' + P_{(m-x)(n-y-1)}{}''\right] \end{aligned}$$

Figure 3.6 shows the routing densities from $(0,0)$ to $(3,2)$ with $\alpha = 0.5$ using this algorithm. The results are almost identical to the exact probabilities shown in Figure 4. When $\alpha = 0$ or 1, the results of this algorithm are exact.

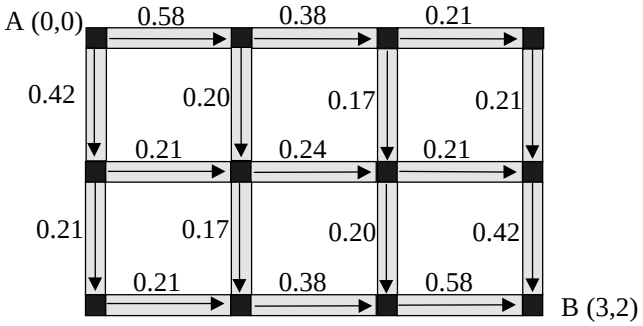


Figure 3.6: Illustration of Wire Congestion Algorithm

Routing between blocks

Suppose there are M wires between block **A** and block **B**; block **A** contains N_A tiles, and block **B** contains N_B tiles. We can run the above algorithm $N_A \cdot N_B$ times to calculate the routing densities, but the computation can be very slow when $N_A \cdot N_B$ is large.

Fortunately, the original algorithm can be modified to calculate the routing densities between two blocks in a single pass. Instead of setting P_{00} to 1.0, P_{ij} is set to $\frac{M}{N_A}$ for all (i, j) inside block **A**. This optimization is valid because routing densities can be superimposed and summed together.

Routing Obstacles

Routing obstacles are common in VLSI designs. For example, some areas in a chip are reserved for local routing and cannot be used for global routing. We can model routing obstacles using the following algorithm.

Step 1: Follow the algorithm described in the previous sub-section but ignore the routing probabilities going into the routing obstacle.

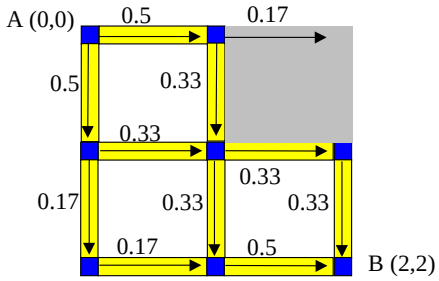
Step 2: Re-route all the routing probabilities going into the routing obstacles back to the origin.

Step 3: Let P_O be the sum of all the routing probabilities going into the routing obstacle. The remaining routing probabilities have to be normalized and divided by $(1 - P_O)$ to compensate for the "missing" probabilities entering the routing obstacles.

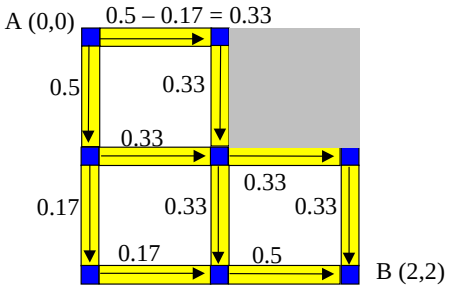
The algorithm is illustrated with an example in Figure 3.7. In Step 1, the routing probabilities are calculated using the previous algorithm. In Step 2, the routing probability

(0.17) going into the obstacle is routed back to the source. In Step 3, all the remaining probabilities are divided by $(1 - 0.17) = 0.83$ to compensate for the probability entering the routing obstacles.

Step 1: Derive the routing probabilities as before.



Step 2: Remove the obstacle probabilities.



Step 3: Compensate for the "missing" probability.

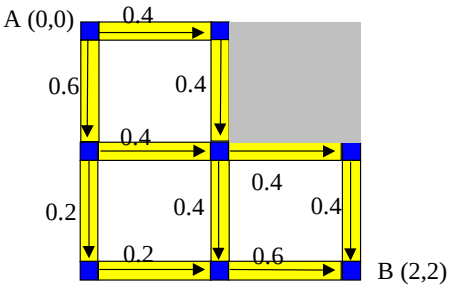


Figure 3.7: Illustration of Wire Congestion Routing Obstacle Algorithm ($\alpha = 1.0$)

3.2.3 Experimental Results

A wire congestion estimator was developed based on the wire congestion model described in this chapter. This congestion estimator is part of the IPLAN floorplanning framework described in Chapter 5. The estimator allows users to create, move, modify macros blocks, and create interconnections among the macro blocks. If the locations of the pins are not defined, it assumes the pins to be distributed evenly throughout the macro block. If the locations are defined, it allows direct pin-to-pin connections.

The wire congestion estimator has been tested with a number of chips, and the congestion estimations appear to be accurate. However, it is important to configure the model correctly, or the results can be incorrect. In this section, we use two real chips to illustrate how the congestion estimator works and how to configure the model.

The first chip is an 8-bit, 8-tap adaptive FIR filter (Figure 3.8), which encompasses eight functional blocks: *Controller*, *Datapath*, *DataShiftReg*, *CoefShiftReg*, *DatapathDriver*, *Drv*, *Clk* and *Counter*. Figure 3.8(a) and 3.8(b) show two congestion maps based on two different configurations, and Figure 3.8(c) shows the $M1$ metal interconnections of the actual physical layout. In this case, the congestion estimator only considers the global interconnections because it reads only the global interconnection netlist. To evaluate the accuracy and effectiveness of the estimator, we should focus on the global interconnections between functional blocks, and ignore the local interconnections within each block.

In the first congestion map (Figure 3.8(a)), the congestion estimation is based purely on the global netlist. The map shows congestion between *Datapath* and *DataShiftReg*, which can also be found in the layout. However, the map also shows some routing congestion between *Controller* and *DatapathDriver*, which does not exist in the actual layout. The discrepancy is due to the fact that the congestion estimator assumes random pin locations within the *Controller* and *DatapathDriver* blocks. Actually, the pins are lined up between *Controller* and *DatapathDriver* boundary. After defining the correct pin locations (Figure 3.8(b)), the second congestion map predicts the congestion more accurately. It is useful to allow users to configure or not configure pin locations in the congestion estimator. Usually, pin locations are not defined in early design stages, and the congestion estimator can provide a rough congestion estimation. As the design progresses, pin and block locations can be defined more precisely, and the congestion estimation becomes more accurate.

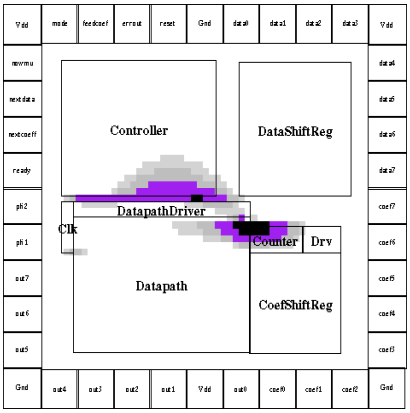
The second chip is a 16-bit pipelined microprocessor (Figure 3.9), which encompasses five functional blocks: *Controller*, *DataPath*, *AddrGen*, *RegFile* and *Pipeline*. Figure 3.9(a)

shows the congestion estimation, whereas Figure 3.9(b) shows the $M1$ interconnection in the actual physical layout. As in the previous example, we should focus on the interconnections among the functional blocks and ignore the interconnections within each block. For the control and data busses, the via minimization parameter (α) should be set as zero because busses typically have very few vias. The congestion map shows routing hot spots between *AddGen* and *RegFile*, matching the actual layout.

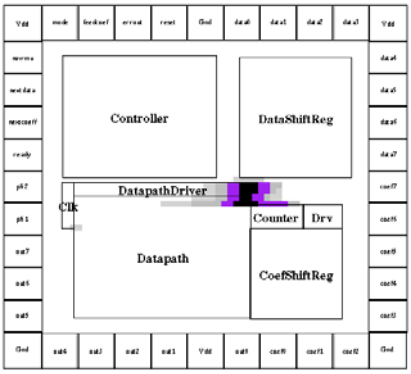
Besides identifying congestion hotspots, the congestion estimation can also be used in the routing algorithm. In Figure 3.9b, the router uses minimum pitches to connect the wires between the *Controller* and *Pipeline* blocks, but the congestion estimation map correctly predicts that the wiring densities between these two blocks are very low. A "congestion conscious router" can use a larger wiring pitch for these interconnections based on the congestion estimation, thus reducing the cross capacitances in these wires.

As described earlier in this chapter, interconnect cross-capacitance is a major factor in determining system performance and power consumption. A "congestion-conscious" router can significantly reduce interconnect delay and power consumption. For example, cross capacitances can be reduced by as much as 60% if wiring pitches are double. In addition to the above two applications, the congestion estimation can also be used in floorplanning and block placement, as described in the next section.

(a) First Congestion Map



(b) Second Congestion Map



(c) Actual Layout

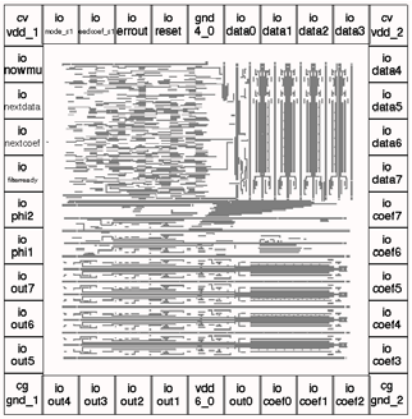
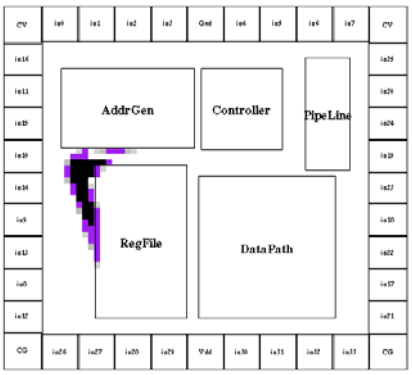


Figure 3.8: Wire Congestion Example: Chip A (Adaptive FIR Filter)

(a) IPlan Result ($\alpha = 0.5$)



(b) Actual Layout

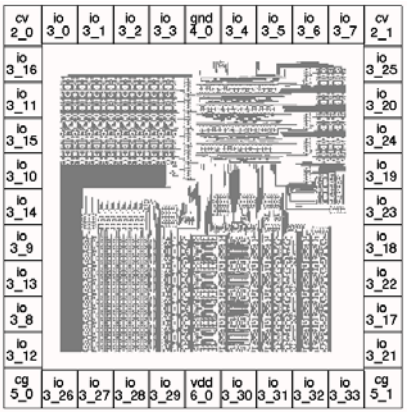


Figure 3.9: Wire Congestion Example: Chip B (16-bit Microprocessor)

3.3 Floorplan Optimizer

3.3.1 Problem Formulation

In the conventional design flow, chip floorplanning is performed only in the physical design stage. This design methodology is becoming problematic, as feature size continues to shrink. As floorplanning can decidedly affect architectural and logic design optimization, we are convinced that floorplanning should become part of the architectural and logic design phases.

The IPLAN floorplanning framework, described in detail in Chapter 5, allows designers to perform chip floorplanning in the architectural and logic design stages. This section describes how the IPLAN floorplanner utilizes the congestion information to optimize a floorplan. As an example, we describe how to optimize the power consumption of a floorplan based on the congestion estimator.

There are two classes of floorplan representations: slicing and non-slicing. A slicing floorplan [Ott82] is obtained by recursively dividing a rectangle into two parts by either vertical or horizontal lines. All other floorplans are classified as non-slicing floorplans. Slicing structure provides an easy way to optimize block orientations [Sto83], define reasonable channels in global routing [MNK95], and order channels in detailed routing [Kaj83]. Figure 3.10 shows a simple slicing floorplan example, consisting of seven rectangular blocks (A, B, C, D, E, F, G).

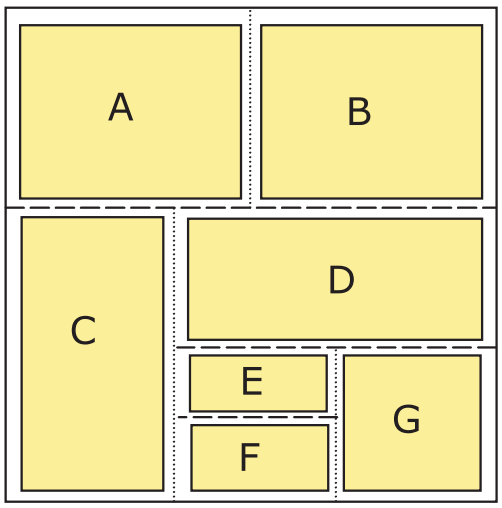


Figure 3.10: A Slicing Floorplan Example

In this example, the floorplan is constructed by first slicing the floorplan horizontally and dividing the floorplan into two halves: {A, B} and {C, D, E, F, G}. The upper half is sliced vertically and in turn divided into two parts: {A} and {B}, while the lower half is also sliced vertically and in turn divided into two parts: {C} and {D, E, F, G}. This slicing process is continued until all blocks are contained in their own partitions. The slicing floorplan structure can be efficiently represented using a binary tree, manipulated easily in software using linked lists. Figure 3.11 shows the binary tree representation of the example in Figure 3.10.

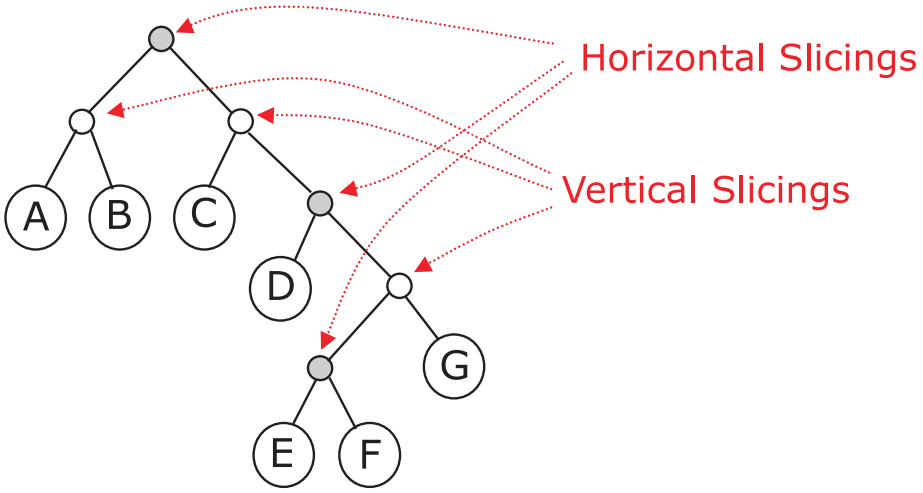


Figure 3.11: Binary Tree Representation of the Slicing Floorplan

The slicing floorplan structure is easy to manipulate, and can be optimized efficiently using divide-and-conquer techniques. On the other hand, the slicing floorplan structure sometimes can be ineffective because it does not represent all compact floorplans. Figure 3.12 shows a floorplan example that cannot be represented by a slicing floorplan. In general, this problem is more serious when the number of blocks is small (e.g. < 20) because it is more difficult to find a compact floorplan following the slicing floorplan structure.

Fortunately, a general floorplan of rectangular blocks can be represented by two other structures: sequence pair structure [MFNK95] and bounding slicing grid structure (BSG) [NFMK96]. Both of these two structures were extended to support convex rectilinear blocks [Mag98a, SNK98, Mag98b, XC98]. The main drawback of non-slicing floorplan structures is that they are computationally expensive to optimize. In fact, the decision whether a given set of fixed-oriented rectangles, having width and heights real numbers, can be packed

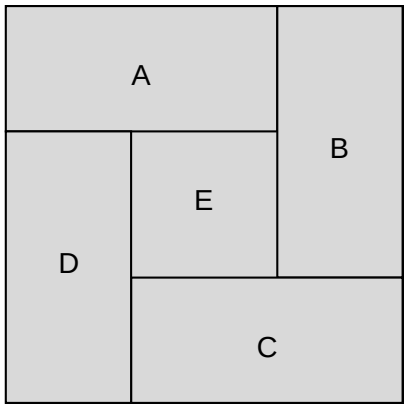


Figure 3.12: A Non-Slicing Floorplan Example

into a chip of known width and height was proven to be NP-complete [BCR90] whereas the problem of finding a minimum area packing is NP-hard. In the IPLAN floorplan framework, the number of functional blocks is assumed to be not too large (e.g. < 40). Thus, we elect to use the sequence-pair representation because non-slicing floorplans are more compact and sequence-pair structures can be handled easily in software.

Sequence Pair Structure

The topological relationships among all the blocks in a floorplan may be expressed by a sequence pair. Murata's paper [MFNK95] describes the sequence pair structure in detail. Assume that there are Q blocks in a floorplan, and each block is represented by a symbol. A sequence pair for a floorplan consists of two sequences of the Q symbols. For example, assume that Q is 5. Given a sequence pair (A D E B C, D C E A B), an oblique grid may be constructed as shown in Figure 3.13.

For every block, the plane is divided into 4 regions by the two crossing slope lines. In this example, block A is in the upper region above block E, which means that A is above E. Similarly, B is to the right of E, C is below E, and D is to the left of E. This sequence pair represents the floorplan shown in Figure 3.12.

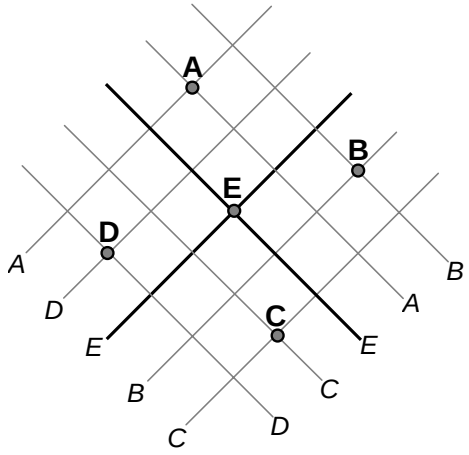


Figure 3.13: Sequence Pair Example

Power Dissipation Model

In the past the main objective of the floorplanning stage was to minimize the chip area. At deep submicron feature sizes, floorplanning plays a vital role in optimizing system performance and power consumption. This section formulates how to optimize power optimization in a floorplan. As described in Chapter 1, the power dissipation in a CMOS circuit is mainly due to the interconnect switching loss ($P_{interconnect}$). In Equation 3.10, $C_{interconnect}$ is the interconnect capacitance, f_{clk} is the clock frequency, V_{dd} is the supply voltage, and p_t is the probability that a transition occurs.

$$P_{interconnect} = p_t \left(C_{interconnect} \cdot V_{dd}^2 \cdot f_{clk} \right) \tag{3.10}$$

In order to evaluate the interconnect power dissipation, we need to examine the interconnect capacitance in more detail. Sakurai [ST83] proposed simple formulas to estimate the line-to-ground capacitance (C_1) and cross capacitances (C_{20}, C_{21}). The formulas are shown in Equations 3.11 and 3.12, where L is the total wire length. The parameters W, H, T, and S are defined in Figure 3.14.

$$C_1 = \epsilon_{ox} \cdot \left[1.15 \left(\frac{W}{H} \right) + 2.8 \left(\frac{T}{H} \right)^{0.222} \right] \cdot L \tag{3.11}$$

$$C_{20} = C_{21} = \epsilon_{ox} \cdot \left(\frac{S}{H}\right)^{-1.34} \cdot L \cdot \left[0.03\left(\frac{W}{H}\right) + 0.83\left(\frac{T}{H}\right) - 0.07\left(\frac{T}{H}\right)^{0.222}\right] \quad (3.12)$$

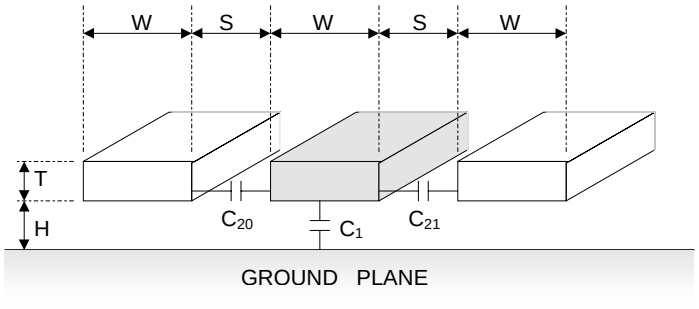


Figure 3.14: Sakurai Wire Capacitance Model

The transition probability (p_t), voltage swing (V), and clock frequency (f_{clk}) are different for every net. In some cases, even the supply voltage (V_{dd}) may vary from one net to another [CP96, RS95]. We define the activity factor w_i for net i as the product $p_t \cdot V \cdot V_{dd} \cdot f_{clk}$. Let P_0 be the power dissipation due to line-to-ground capacitance, and let P_1 be the power dissipation due to cross capacitance. P_1 depends on both the wire length and wire spacing, and P_0 depends only on the wire length. Using Equations 3.11 and 3.12, we can derive:

$$P_0 \propto w_i \cdot L \quad (3.13)$$

$$P_1 \propto w_i \cdot L \cdot S^{-1.34} \quad (3.14)$$

The interpretation of these two equations is quite simple: when the activity factor (w_i) is high, the wire length (L) should be minimized and the wire spacing (S) should not be too small.

Wire Density and Power Dissipation

This section derives the relationships between wire densities and power consumption. Based on the following theorems, the power dissipation can be calculated from the wire densities.

Theorem 1 $\sum_{x,y} D_H(x,y) + D_V(x,y)$ *is proportional to the total wire length in a layout.*

Proof: The total wire length is equal to the sum of all the wire segments in a grid graph. Therefore, the total wire length is equal to $\sum_{x,y} D_H(x,y) + D_V(x,y)$ multiplied by the spacing between adjacent grids.

Theorem 2 *For each net i, w_i is the activity factor as defined in Section 3.1. Assuming that the number of wires in each net is multiplied by w_i, let $D_H{}'(x,y)$ and $D_V{}'(x,y)$ be the new horizontal and vertical wire densities at (x,y). $\sum_{x,y} D_H{}'(x,y) + D_V{}'(x,y)$ is proportional to the total power dissipation due to line-to-ground capacitance (P_0).*

Proof: Using Theorem 1, $\sum_{x,y} D_H{}'(x,y) + D_V{}'(x,y)$ is proportional to to $w_i \cdot$ *total wire length.* As shown in Section 3.1, $w_i \cdot$ *total wire length* is proportional to P_0.

Theorem 3 $\sum_{x,y} D_H{}'(x,y)^{2.34} + D_V{}'(x,y)^{2.34}$ *is roughly proportional to the total power dissipation due to cross capacitance (P_1).*

Proof: The wire density is assumed to be roughly inversely proportional to wire spacing S. The summation $\sum_{x,y} D_H{}'(x,y)^{2.34} + D_V{}'(x,y)^{2.34}$ is equal to $\sum_{x,y} D_H{}'(x,y)^{1.34} \cdot D_H{}'(x,y) + D_V{}'(x,y)^{1.34} \cdot D_V{}'(x,y)$, which is proportional to $S^{-1.34} \cdot w_i \cdot$ total wire length. As shown in Section 3.1, $S^{-1.34} \cdot w_i \cdot$ *total wire length* is proportional to P_1.

3.3.2 Algorithm

As described in Section 3.3.1, optimizing a non-slicing floorplan is a NP-hard problem and it would be impractical to find the optimum solution by "brute-force." We follow Murata's approach to use a heuristic technique – simulated annealing – to search for optimal solutions [MFNK95]. In Murata's paper, the floorplan area is used as the objective function. In this section, we apply different objective functions and examine their impacts on power consumption and area.

Simulated annealing [KGV83] is a generalization of a Monte Carlo simulation [MRR+53]. It is a computational process patterned after the physical process of annealing, in which physical substances such as glass are melted and then gradually cooled until some solid state is reached. The goal of this process is to achieve a minimal-energy final state. By performing enough exploration of the entire space early on, the final solution is relatively insensitive to the starting state. This would minimize the chances of getting caught at a local minimum state.

In the physical annealing process, substances usually move from higher energy configurations to lower ones but there is some probability that the reverse can also occur. This probability p is given by the following equation.

$$p = e^{-\Delta E/kT} \tag{3.15}$$

where ΔE is the positive change in the energy level, T is the temperature, and k is Boltzmann's constant. The simulated annealing algorithm follows the physical annealing process. In this analogous process, ΔE is generalized to represent the change of the value of an objective function. Since the units for E and T are artificial, the Boltzmann's constant k is incorporated into T. Thus, the following revised probability formula is usually used in the simulated annealing algorithm.

$$p' = e^{-\Delta E/T} \tag{3.16}$$

In case of our floorplanner, a sequence pair is created at random initially. The temperature is gradually lowered following an exponential cooling schedule. At each simulated annealing step, the algorithm considers one of the following three perturbations: (a) rotation of a random block, (b) "half exchange" of the sequence pair, (c) "full exchange" of the sequence pair. The "half" and "full" exchanges of a sequence pair are simply mutations in the sequence pair structure, and are described in [MFNK95] in detail.

The floorplanning algorithm may or may not accept each perturbation, depending on three factors: the result of an objective function, the current temperature, and the result of a random number generator. If the objective function of the new solution is better than the present solution, the perturbation is accepted. On the other hand, if the objective function of the new solution is worse than the present solution, Equation 3.16 defines the probability determining whether the new solution is accepted or not. As the temperature decreases, this probability gets smaller and smaller.

How do we select the objective function? Obviously, we can use the power consumption equation as our objective function. However, the resulting floorplan may be too large because the area is not optimized. A different approach is to use a constraint optimization, in which each perturbation is accepted only if the constraints are met. This approach can be useful. For instance, an area constraint may be imposed so that only perturbations smaller than a certain area are accepted. One caveat of this approach is that it may create sharp

edges and non-linearity in the solution space and thus affect the quality of the simulated annealer. In this experiment, we elect to use an objective function (Φ_P) that takes both power and area into consideration (Equation 3.17). This objective function does not create sharp edges in the solution space and appears to give fairly good results.

$$\Phi_P = Area \cdot {P_0}^{\lambda_0} \cdot {P_1}^{\lambda_1} \tag{3.17}$$

In this equation, $Area$ is the total floorplan area, P_0 and P_1 are the power consumption due to line-to-ground and cross capacitances, respectively. λ_0 and λ_1 are parameters determining the relative importance among $Area$, P_0, and P_1. For example, if λ_0 and λ_1 are small, the objective function is insensitive to power consumption. Conversely, if both λ_0 and λ_1 are large, Φ_P is less sensitive to $Area$. Using the theorems in Section 3.3, the objective function can be expressed in terms of the wire densities (Equation 3.18).

$$\Phi_P = Area \cdot \Big[\sum_{x,y} {D_H}'(x,y) + {D_V}'(x,y)\Big]^{\lambda_0} \cdot \Big[\sum_{x,y} {D_H}'(x,y)^{2.34} + {D_V}'(x,y)^{2.34}\Big]^{\lambda_1} \tag{3.18}$$

In our experiment (results described in Section 3.3.3), four different sets of λ_0 and λ_1 are used. Set 1 optimizes only for area, in which both λ_0 and λ_1 are 0.0. Set 2 optimizes for both area and line-to-ground capacitance, in which λ_0 is 1.0 and λ_1 are 0.0. Set 3 and Set 4 optimize for all three design considerations: area, line-to-ground capacitance and cross capacitance; however, Set 4 emphasizes cross capacitance more than Set 3. In Set 3 λ_0 is 1.0 and λ_1 are 0.5, whereby in Set 4 λ_0 is 1.0 and λ_1 are 1.0.

For simplicity, we assume that the objective function Φ_P is smooth at the optimal point. Thus, we can differentiate $ln(\Phi_P)$ at the optimal point and get the following equation.

$$\frac{dArea}{Area} + \lambda_0 \cdot \frac{dP_0}{P_0} + \lambda_1 \cdot \frac{dP_1}{P_1} = 0 \tag{3.19}$$

For example, λ_0 is 1.0 and λ_1 is 0.0 in Set 2, and Equation 3.19 becomes:

$$\frac{dArea}{Area} + \frac{dP_0}{P_0} = 0 \tag{3.20}$$

In Set 2 the objective function treats area and line-to-ground capacitance equally. In other words, the floorplanning algorithm is willing to accept a $p\%$ decrease in line-to-ground capacitance at the expense of $p\%$ increase in the total area, or vice versa. Similarly, the

objective function considers area, line-to-ground capacitance and cross capacitance in Set 3. However, cross capacitance is treated less importantly compared with area and line-to-ground capacitance in this case. The floorplanning algorithm is willing to accept a $p\%$ decrease in cross capacitance at the expense of only $\frac{p}{2}\%$ increase in the total area.

3.3.3 Experimental Results

Our simulated annealing floorplanning algorithm was implemented and tested with the MCNC building block examples (*apte, xerox, hp, ami33*). The numbers of blocks and nets for each example are shown in Figure 3.15.

	MCNC Benchmark			
	apte	xerox	hp	ami33
No. of blocks	9	10	11	33
No. of nets	97	203	83	123

Figure 3.15: MCNC Benchmark Statistics

In our experiment, all the modules are hard modules which means that the dimensions of each block are fixed and inflexible. For simplicity, all terminals are assumed to be located at the centers of the blocks. The multi-terminal pins are decomposed to 2-terminal pins as described in Section 3.2, and the activity factors (w_i) for all the nets are set to 1.0.

As described in the previous sub-section, the benchmarks were tested with four different objective functions. The results of the MCNC benchmark are shown in Figure 3.16. The first objective function (Set 1) only optimizes for the total floorplan area. The second objective function (Set 2) optimizes for both area and line-to-ground capacitance. The third and fourth objective functions (Set 3 and Set 4) optimizes for area, line-to-ground capacitance and cross capacitance. The difference between the last two sets is that the fourth objective function (Set 4) puts more emphasis on the cross capacitance than the third objective function (Set 3).

In order to compare the numbers between different sets of data easily, the results are normalized with respect to the first objective function (Set 1). When we compare the results of the second objective function (Set 2) with respective to the first objective function (Set 1), we find that the line-to-ground capacitance P_0 is lowered by 11% to 48% at the expense of roughly 10% increase in *Area*. Depending on the target application, one of these two design points may be more favorable. For example, power consumption may not be an

		MCNC Benchmark			
		apte	xerox	hp	ami33
Set 1	Area	100%	100%	100%	100%
$\lambda_0 = 0.0$	P_0	100%	100%	100%	100%
$\lambda_1 = 0.0$	P_1	100%	100%	100%	100%
Set 2	Area	110%	117%	113%	107%
$\lambda_0 = 1.0$	P_0	52%	51%	69%	89%
$\lambda_1 = 0.0$	P_1	13%	28%	40%	78%
Set 3	Area	104%	115%	107%	104%
$\lambda_0 = 1.0$	P_0	50%	46%	76%	88%
$\lambda_1 = 0.5$	P_1	11%	21%	36%	76%
Set 4	Area	107%	135%	136%	103%
$\lambda_0 = 1.0$	P_0	53%	54%	53%	89%
$\lambda_1 = 1.0$	P_1	9%	21%	29%	75%

* All results are shown in relative percentages with respect to Set 1.

Figure 3.16: MCNC Benchmark Result

important consideration for stationary appliances and designers may opt for a smaller and cheaper design (Set 1).

Similarly, we can perform the same comparisons for the third and fourth objective functions (Set 3 and Set 4). In case of Set 3, the objective function is implemented as $Area^2 \cdot {P_0}^2 \cdot P_1$ to avoid expensive square root operation. For the MCNC benchmarks, the third objective function appears to give fairly good results. In general, the total area is slightly smaller than the second objective function, and the power consumption due to cross capacitances is less than the first two cases. Lastly, the fourth objective function (Set 4) yields the smallest P_1 term (as expected) but the area is significantly larger than the third objective function, especially for the *xerox* and *hp* benchmark.

After examining the results of these four objective functions, it appears that if power consumption is an important criteria, the third objective function (Set 3) should be used. On the other hand, if cost and die size is the only design criteria, obviously the first objective function (Set 1) should be used.

3.4 Summary

This chapter presents a statistical wire congestion model and a floorplanning algorithm using this model. As feature size shrinks, cross coupling between adjacent wires becomes very severe, affecting the performance and power consumption of the system. The wire congestion model is very important, helping designers identify routing hotspots and avoid excessive cross coupling.

The wire congestion model takes many practical considerations (such as via minimization, routing obstacles, multi-pin nets, routing between points and blocks) into account. A simple and efficient algorithm is presented in Section 3.2.2 to calculate the routing densities. Our model has been compared with the actual layouts in some simple designs and the results are positive. One caveat is that in order to obtain a more accurate estimation, the floorplan model has to be configured with care. Specifically, the exact pin locations and routing obstacle positions should be input to the floorplanner if they are available.

The floorplanning algorithm uses a non-slicing floorplan representing the layout of a design. The non-slicing floorplan is internally denoted by a sequence-pair structure. In general, a non-slicing floorplan is more compact than a slicing floorplan but its optimization process is also more computationally expensive. In our floorplanner, a simulated annealer is used to search for optimal solution. The objective function of the annealer is based on the outputs of the congestion model. The relationships between routing densities and interconnect capacitances are derived in this chapter. Based on these derivations, the floorplanner can use the congestion model to optimize the system power consumption. Using the MCNC benchmarks as our example, we show how the floorplanner provides users with design tradeoff information (e.g. between chip area and power consumption).

Chapter 4

Processor Performance Model

While the last two chapters discuss the on-chip interconnections in terms of wirelength distribution and wire congestion, this chapter takes a different perspective and examines the impacts of interconnect overheads on processor performance. Like the previous two interconnect models, this interconnect-driven processor performance model is part of the IPLAN floorplanning framework described in Chapter 5.

In the last fifty years, computer architecture has evolved continuously pursuing higher performance, which usually comes at the expense of increased hardware complexity. Server processors, such as IBM Power4/Regatta [Die00], Compaq Alpha 21264C [Kre01a], Intel Itanium [Gla02] and AMD Athlon MP [Kre01b], are by and large super-pipelined and issue multiple instructions per cycle [MPR02]. On the other hand, the interconnect overhead associated with the additional complexity can limit the benefit of these complicated techniques [FHR99]. As discussed in Chapter 1, processor architectures have to be "interconnect-driven" in deep submicron feature size. This chapter presents a processor performance model that takes interconnect and other hardware complexity overheads into consideration.

4.1 Introduction

Dubey and Flynn showed that pipelining in processor can result in diminishing and even negative returns [DF90]. This is mostly due to clock overheads (e.g. clock skews caused by interconnect delays) and pipeline setup overhead [Phi97]. Assume that the total time to execute an instruction without clock overhead is T, the clock overhead per pipeline stage

is C, and the total time T is segmented into S segments to allow pipelining. Ideally the pipeline completes one instruction per cycle $(T/S + C)$, but there are disruptions such as unexpected branches that result in flushing and restarting the pipeline. Suppose these interruptions occur with frequency b and have the effect of invalidating $S - 1$ instructions. The pipeline throughput G becomes

$$G = \frac{1}{1 + (S-1)b} \cdot \frac{1}{T/S + C} \tag{4.1}$$

Differentiating G with respect to S determines the optimum number of pipeline stages S_{opt} as shown below.

$$S_{opt} = \sqrt{\frac{(1-b)T}{bC}} \tag{4.2}$$

Intuitively, if the cycle time is too large and there are too few stages in the pipeline, we sacrifice overall performance by not overlapping execution sufficiently. On the other hand, if the cycle time is too small, we sacrifice overall performance by incurring too much clock overhead and suffering long pipeline breaks. Figure 4.1 shows the relationship between the performance and the number of pipeline stages.

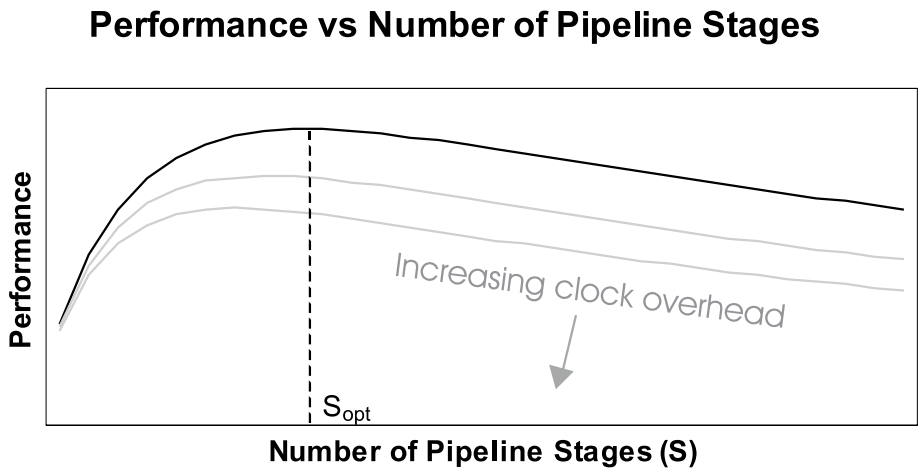


Figure 4.1: Processor Performance vs Number of Pipeline Stages.

There is an analogous behavior in selecting the optimum instruction issue width in a processor. If the issue width is too narrow, we sacrifice overall performance because the

processor cannot fully exploit the available instruction-level parallelism (ILP) in a program. Conversely, if the issue width is too wide, we sacrifice overall performance by incurring excessive interconnect and other hardware overheads to achieve the ILP. While it is well known that increasing the width in a processor can result in diminishing returns [JW89, Phi97], there is no systematic approach to determine the optimum width. In this chapter, we describe a new analytical processor model and show that there is a simple way to optimize instruction issue width. Section 4.2 describes the model and derives a upper bound on the optimum instruction issue width. Section 4.3 presents the simulation results. Section 4.4 discusses some applications of the model.

4.2 Problem Formulation

In this section, we present an analytical model representing the performance of a concurrent processor. A concurrent processor [Fly95], also known as an ILP-processor [SFK97], may be defined as a processor that can fetch, decode, issue, execute and retire multiple instructions per cycle. This is in contrast to a scalar processor that can only fetch, decode, issue, execute and retire one instruction per cycle. The instructions in a concurrent processor may be dynamically scheduled as in a superscalar processor [Joh91], or statically scheduled as in a VLIW (very long instruction word) processor [Fis83]. Our processor performance model consists of two parts: an IPC model and an interconnect / complexity overhead model.

4.2.1 IPC Model

The throughput of a processor may be measured in terms of the average number of instructions executed per cycle (IPC). A concurrent processor often has a larger IPC than a similar scalar processor, and can therefore achieve a better performance than the scalar processor running at the same clock speed. Figure 4.2 shows the instruction issue stage of a 4-way superscalar processor. In this example, the instruction window size is twelve, meaning that the processor can select from twelve decoded instructions to issue in every cycle [Joh91]. In this example, nine out of the twelve instructions in the instruction window cannot be issued due to instruction dependencies or resource conflicts in this clock cycle. Consequently, only three of the four issue slots are occupied.

Let $IPC(N)$ and $IPC(1)$ be the IPC of an N-way concurrent processor and a scalar

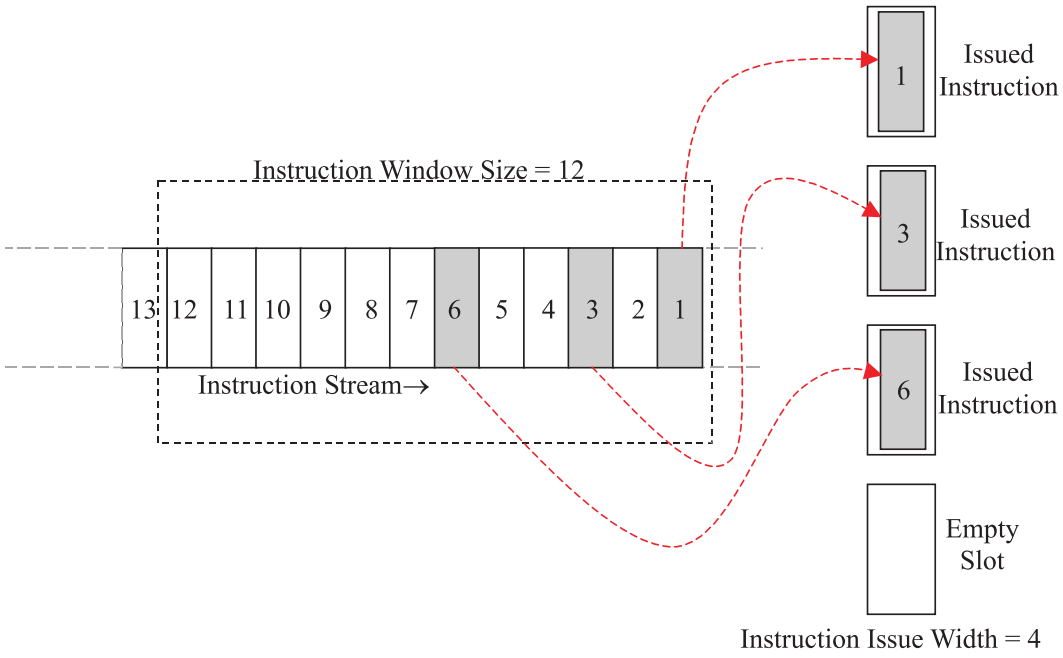


Figure 4.2: Instruction Issue in a 4-Way Superscalar Processor

processor, respectively. The N-way concurrent processor can potentially issue and execute $N-1$ more instructions than the scalar processor. We define a random variable P as the probability that each of the additional $N-1$ issue slots is idle. Thus, the difference between $IPC(N)$ and $IPC(1)$ is determined by Equation 4.3.

$$IPC(N) - IPC(1) = (N-1) \cdot (1-P) \tag{4.3}$$

If the probability P is 0, the concurrent processor can issue $N-1$ additional instructions every cycle. Conversely, if the probability P is 1.0, the concurrent processor is virtually same as a scalar processor. In a useful and effective architecture, the value of P cannot be too large (e.g. > 0.8) because the issue slots should not be idle most of the time. The probability P depends primarily on the processor microarchitecture, including the instruction issue width N, the instruction window size, the number of functional units and the memory and execution latencies, and so on. Increasing the instruction issue width N increases P, while increasing the instruction window size or the number of functional units reduces P.

The next step in the derivation is to examine the number of active instructions (M) in

a processor. An active instruction is defined as an instruction that has been issued but not yet retired. Some instructions, such as "load" or "store", may take many processor cycles (e.g. 10 – 20), whereas an "integer add" instruction typically takes much fewer cycles. As depicted in Figure 4.3, the average number of active instructions (M) is the product of $IPC(N)$ and L.

$$M = IPC(N) \cdot L \tag{4.4}$$

The parameter M is instrumental in determining the IPC. As M gets bigger, the probabilities of instruction dependencies and resource conflicts becomes higher, and the probability P increases. First, we examine the effects of instruction dependency on the probability P. Let d be the probability that any two instructions in the instruction stream are dependent. The probability d is application-dependent and we assume that d is a constant for each application. Figure 4.4 shows the relationship between P and M for $d = 5\%$ to 10%. As expected, the probability P increases with M for a given d and also increases with d for a given M.

Let p be the multiplicative factor between P and M (Equation 4.5). Please note that the factor p is not a constant but a function of M. In fact, p always increases when M increases within meaningful design range from $P = 0$ to $P = 0.8$. In other words, p is an increasing function of M within this range.

$$P = p \cdot M \tag{4.5}$$

While the precise impacts of resource conflicts on P depends largely on the hardware

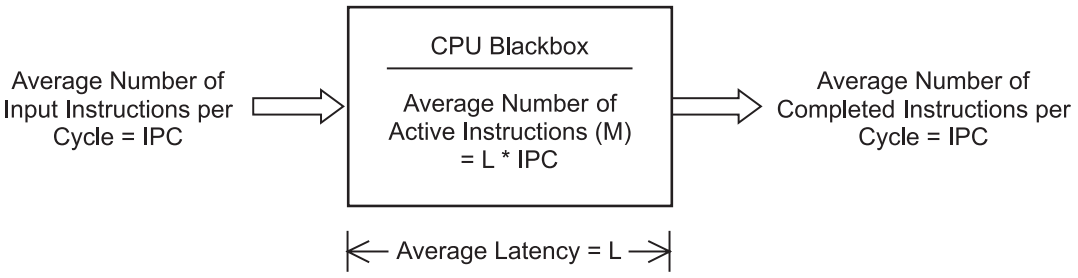


Figure 4.3: Number of Active Instructions in a Processor

implementation, their effects are similar to those of instruction dependencies based on analogous arguments and derivations. Thus, we use the same equation to model both instruction dependencies and resource conflicts.

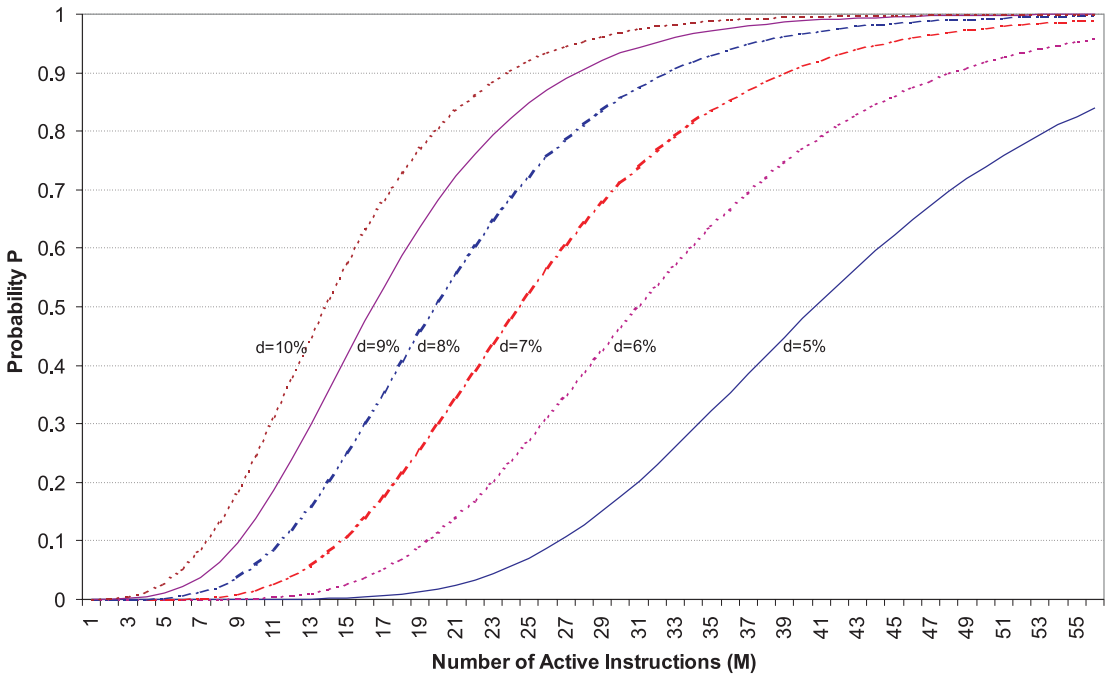


Figure 4.4: Relationship between issue slot idle probability (P) and the number of active instructions (M) for different instruction dependency probabilities (d = 5%, 6%, 7%, 8%, 9%, 10%)

Using Equations 4.3, 4.4, and 4.5, we derive $IPC(N)$ in terms of $IPC(1)$, N, p, and L.

$$IPC(N) = \frac{IPC(1) + (N-1)}{1 + p \cdot L \cdot (N-1)} \tag{4.6}$$

Consider the limiting case when N approaches ∞. As the vast majority number of issue slots are idle, the probability P approaches 1 when N approaches ∞. Thus, the number of active instructions (M) can be approximated with $1/p$ (Equation 4.5). Let $IPC(\infty)$ be the IPC of an ideal processor with an infinite instruction issue width. From Equation 4.4, we

get:

$$IPC(\infty) = \frac{M}{L} = \frac{1}{p \cdot L} \tag{4.7}$$

If we approximate $p \cdot L$ to be a constant, the IPC equation can be rewritten in terms of N, $IPC(1)$, and $IPC(\infty)$ as in Equation 4.8.

$$IPC(N) \approx \frac{IPC(1) + (N-1)}{1 + \frac{(N-1)}{IPC(\infty)}} \tag{4.8}$$

However, $p \cdot L$ is not a constant but an increasing function of N because p is an increasing function of M and M is an increasing function of N, This implies that the actual IPC can be larger than the approximation. Figure 4.5 compares the $IPC(N)$ in Equations 4.6 and 4.8. While the differences are small, the IPC rises and saturates faster in the original equation (Equation 4.6). Thus, we obtain the following expression representing a lower bound on $IPC(N)$.

$$IPC(N) \geq \frac{IPC(1) + (N-1)}{1 + \frac{(N-1)}{IPC(\infty)}} \tag{4.9}$$

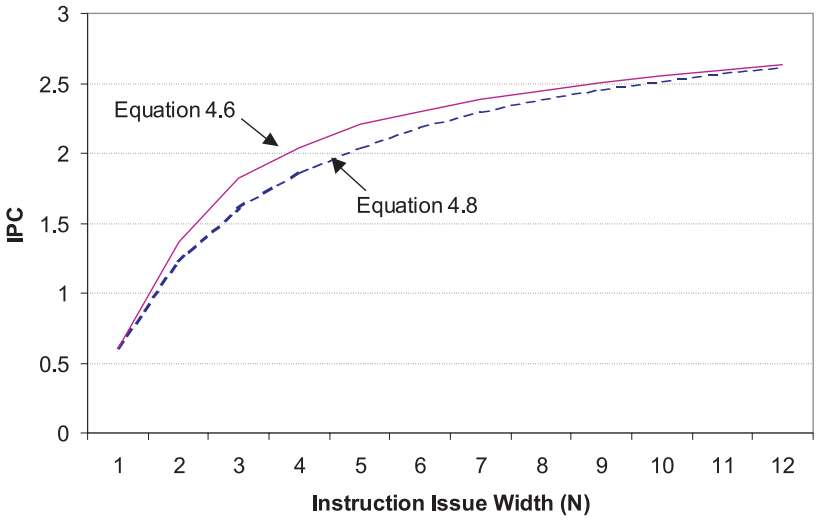


Figure 4.5: IPC vs Instruction Issue Width (N) based on Equations 4.6 and 4.8.

4.2.2 Interconnect/Complexity Overhead Models

Ideally, instruction issue width N does not affect either cycle time or average instruction latency L. However, there are usually extra interconnect and gate delays caused by the added complexity. The extent of the impacts primarily depend on hardware implementation, such as circuit design and physical layout.

In this section, we develop a set of interconnect / complexity overhead models representing these overheads in most implementations. Let L_1 and T_C be the average instruction latency and the cycle time of a scalar processor. Table 4.1 illustrates nine different overhead models. For instance, Model A represents the ideal model where instruction latency clock cycle are constant. The parameter $\mathcal{O}$ is a scaling factor modeling the significance of the overhead. The amount of hardware complexity overhead depends on the overhead model as well as the scaling factor $\mathcal{O}$.

Table 4.1: Various Interconnect and Complexity Overhead Models[1]

Model	Instruction Latency	Cycle Time	Description
A	L_1	T_C	Ideal Model
B	$L_1 \cdot \left(1 + \mathcal{O} \cdot (N-1)^2\right)$	T_C	Non-buffered wire (length $\propto N$)
C	$L_1 \cdot (1 + \mathcal{O} \cdot (N-1))$	T_C	Buffered wire (length $\propto N$)
D	$L_1 \cdot \left(1 + \mathcal{O} \cdot \sqrt{N-1}\right)$	T_C	Buffered wire (length $\propto \sqrt{N}$)
E	$L_1 \cdot (1 + \mathcal{O} \cdot log(N-1))$	T_C	Decoder/Multiplexer
F	L_1	$T_C \cdot \left(1 + \mathcal{O} \cdot (N-1)^2\right)$	Non-buffered wire (length $\propto N$)
G	L_1	$T_C \cdot (1 + \mathcal{O} \cdot (N-1))$	Buffered wire (length $\propto N$)
H	L_1	$T_C \cdot \left(1 + \mathcal{O} \cdot \sqrt{N-1}\right)$	Buffered wire (length $\propto \sqrt{N}$)
I	L_1	$T_C \cdot (1 + \mathcal{O} \cdot log(N-1))$	Decoder/Multiplexer

[1] N is the instruction issue width; $\mathcal{O}$ is the proportionality constant for the overhead factor; L_1 and T_C are the average instruction latency and the cycle time for the scalar processor.

Models B, C, D, F, G, H represent six different interconnect overhead models. In general, interconnect delay can affect either cycle time or average instruction latency (in number of processor cycles). Models B, C, D correspond to an increase in instruction latency, while Models F, G, H correspond to an increase in cycle time. In a linear physical layout, interconnect delay increases quadratically with N without repeaters (Models B and F) but increases linearly with N with repeaters (Models C and G). Similarly in a two-dimensional layout, interconnect delay increases linearly with N without repeaters (Models C and G) but increases linearly with $\sqrt{N}$ with repeaters (Model D and H). Finally, Models E and I are

overhead models related to gate delays. In decoder and multiplexer design, gate delays are proportional to $log(N)$. Model E corresponds to an increase in instruction latency, which Model I corresponds to an increase in cycle time. In deep submicron designs, interconnect overheads are more significant than gate overheads in most designs. Thus, Models E and I are becoming less important.

4.2.3 Overhead Examples

In this section, we analyze two critical pipeline structures (register file access time and instruction wakeup logic time), which often determine the processor clock cycle time.

Register File

The register file of a pipelined processor typically completes read and write operations within a single cycle. This process is atomic and indivisible in most processor designs. As a result, register file access time often limits the maximum clock speed for a processor. In order to investigate the effect of issue width N on register file access time, we examine the relationship between the issue width N and the number of read and write ports required of the register file.

Based on the register file delay model described in [FJC95] and the CMOS scaling model proposed by McFarland [McF97], we derive the register file delay as a function of issue width N and feature size. Using the register file access time, the appropriate overhead scheme and the overhead scale factor for the register file can be determined. Generally speaking, the number of read and write operands per instruction depends on the instruction set architecture (ISA). Typically, there are two or three source operands and one destination operand in a RISC ISA. The upper bound on the number of read and write ports in a register file are therefore three or four times the instruction issue width; however, the number of register ports required is usually smaller. This is because not all instructions use the maximum number of source and destination operands, so it is possible to arbitrate for the register ports [Joh91]. Besides port arbitration, there are other effective ways to reduce the number of register ports, including duplicating and partitioning register file [Kel96, FCJV97].

If N is the instruction issue width, the number of register ports required is about $2N$ for commercial processors. We follow the approach in [FJC95] and use the CMOS scaling model to calculate the register file access time for a 128-entry register file at 0.18 μm technology.

We assume that the register file has $2N$ ports for an issue width N of 1 through 8. The register file delay model includes the decoder, wordline, bitline, sense amplifier, output driver, and precharge delay times. Figure 4.6 shows the register file access time as a function of issue width N; the register file access time is normalized with respect to the access time for $N = 1$. The sense amplifier delay is constant while all other delay components scale linearly with N. If we assume that the register file access time determines the clock cycle time, the cycle time increases linearly with N (Figure 4.6). Thus, the overhead can be modeled using Model G in Table 4.1. The overhead scale factor $\mathcal{O}$ is found to be around 17% from the graph.

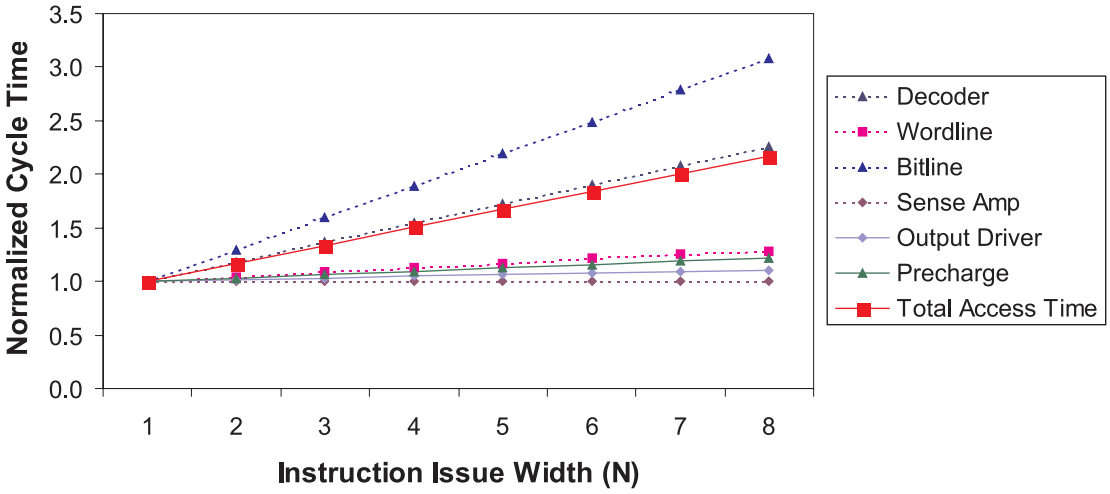


Figure 4.6: Register File Access Time vs Instruction Issue Width (N).

Wakeup Logic

The instruction window is an important piece of processor hardware component required to support out-of-order execution. After instructions are decoded, they are inserted to the instruction window waiting to be issued. In some high-performance processor designs the registers used in the instructions are renamed to eliminate output- and anti-dependencies. When an instruction retires, its destination register number is broadcasted to all the stalled instructions in the instruction window. The tag broadcast and associative comparison is known as instruction wakeup logic.

Palacharla et al. [PJS96] identified the instruction wakeup as one of the timing critical components in superscalar processors. The wakeup logic delay consists of three components: tag drive, tag match, and match-or delays. Figure 4.7 shows the normalized delays of the wakeup logic and its three components. The wakeup delay scales roughly linearly with the issue width N. In fact, the general shape of these graphs are quite similar to the register file access time in the previous section. As N increases, designers can choose to increase the cycle time or the number of pipeline stages for instruction issue, corresponding to Model G and Model C in Table 4.1. In both cases, the overhead scale factor $\mathcal{O}$ is approximately 13%. Please note that Model C cannot be used for register file access time because it is difficult to pipeline a register file.

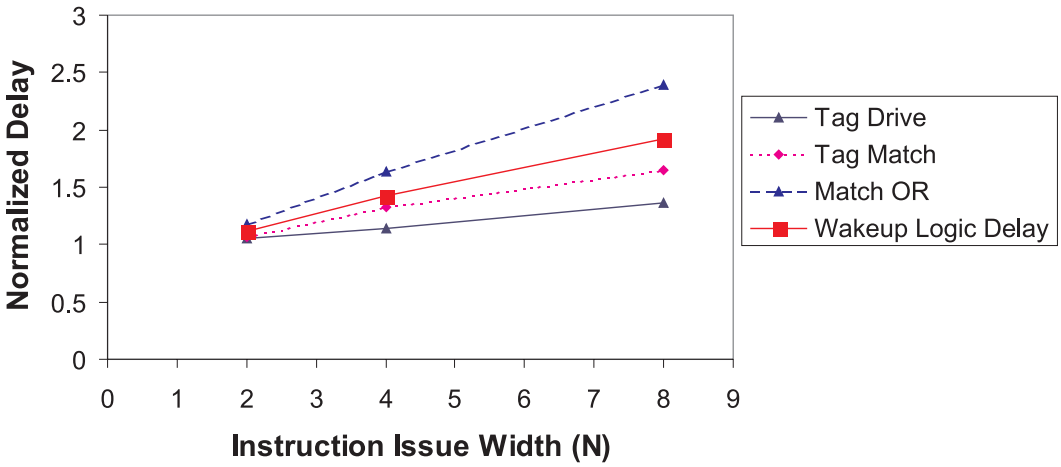


Figure 4.7: Wakeup Logic Delay vs Instruction Issue Width (N).

4.2.4 Upper Bound on Optimum Instruction Issue Width

The performance of a processor is proportional to its IPC and inversely proportional to its cycle time. Using the IPC lower bound expression in (4.9), the performance $G(N)$ can be expressed as follows:

$$G(N) \geq \frac{IPC(1) + (N-1)}{(1 + \frac{(N-1)}{IPC(\infty)}) \cdot CycleTime} \tag{4.10}$$

In this section, we use Model G as an example to illustrate how to determine an upper bound on the optimum instruction issue width (N_{opt}). Substituting $CycleTime$ for Model G into (4.10), we get:

$$G(N) \geq \frac{IPC(1) + (N-1)}{(1 + \frac{(N-1)}{IPC(\infty)}) \cdot T_C(1 + \mathcal{O} \cdot (N-1))} \tag{4.11}$$

Figure 4.8 compares the LHS and RHS of the above inequality. As shown in the figure, the peak of the RHS may serve as an upper bound on the optimum instruction issue width N_{opt}. To find the peak of the RHS, we simply differentiate the RHS with respect to N and

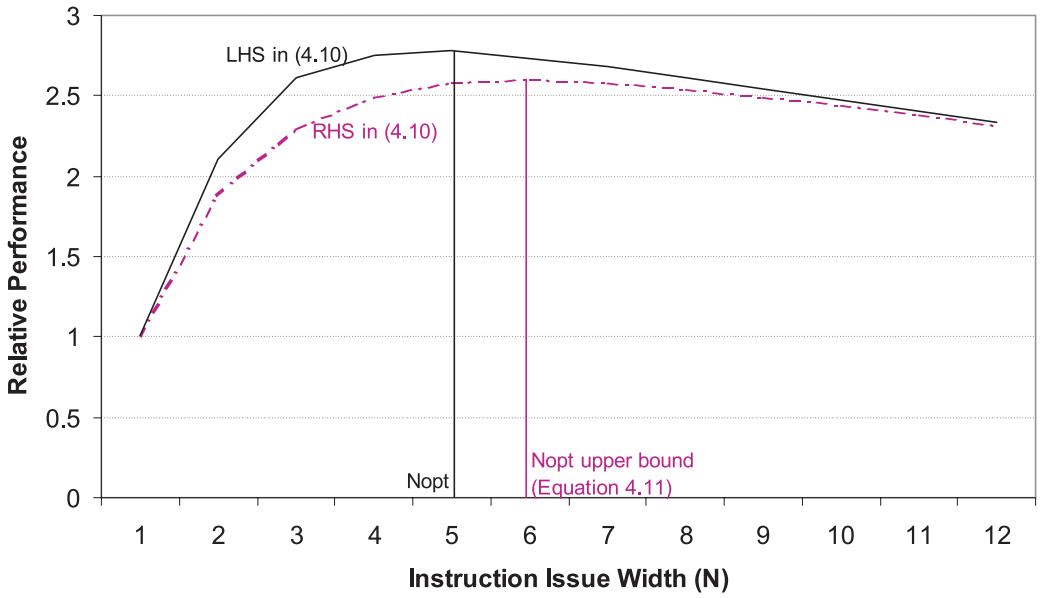


Figure 4.8: Relative Performance vs Instruction Issue Width N.

set the expression to 0. Thus, we get:

$$N_{opt} \leq 1 - IPC(1) + \sqrt{IPC(1)^2 + \frac{IPC(\infty)}{\mathcal{O}} - IPC(1) \cdot (\frac{1}{\mathcal{O}} + IPC(\infty))} \tag{4.12}$$

If we assume that $IPC(1)$ is close to 1 and $\mathcal{O} \ll 1$, (4.12) can further be simplified as shown in (4.13).

$$N_{opt} \leq \sqrt{\frac{IPC(\infty) - IPC(1)}{\mathcal{O}}} = \sqrt{\frac{\Delta IPC}{\mathcal{O}}} \tag{4.13}$$

We can derive the upper bounds on the optimum instruction issue width N_{opt} for the other overhead models in a similar fashion. Table 4.2 summarizes the upper bound expressions for the different overhead models. If the overhead scale factor $\mathcal{O}$ is very small, N_{opt} can be written as shown in (4.14) where $\frac{1}{3} \leq \alpha \leq 1$. The value of α depends on the overhead scheme used. For example, α is equal to $\frac{1}{3}$ for Models B and F, $\frac{1}{2}$ for Models C and G, $\frac{2}{3}$ for Models D and H, and 1 for Models E and I.

$$N_{opt} \leq \left(\frac{\Delta IPC}{\mathcal{O}}\right)^{\alpha} \tag{4.14}$$

Table 4.2: Upper Bounds on N_{opt} for Various Interconnect/Complexity Overhead Models

Model	Instruction Latency	Cycle Time	Upper Bound on N_{opt}
A	L_1	T_C	∞
B	$L_1 \cdot \left(1 + \mathcal{O} \cdot (N-1)^2\right)$	T_C	$1 - \frac{IPC(1)}{2} + \frac{1}{\sqrt[3]{2}} \cdot \sqrt[3]{\frac{\Delta IPC}{\mathcal{O}}}$
C	$L_1 \cdot (1 + \mathcal{O} \cdot (N-1))$	T_C	$1 - IPC(1) + \sqrt{\frac{\Delta IPC}{\mathcal{O}}}$
D	$L_1 \cdot \left(1 + \mathcal{O} \cdot \sqrt{N-1}\right)$	T_C	$1 + \sqrt[1.5]{\frac{2}{3}} \cdot \sqrt[1.5]{\frac{\Delta IPC}{\mathcal{O}}}$
E	$L_1 \cdot (1 + \mathcal{O} \cdot log(N-1))$	T_C	$1 + \frac{\Delta IPC}{\mathcal{O}}$
F	L_1	$T_C \cdot \left(1 + \mathcal{O} \cdot (N-1)^2\right)$	$1 - \frac{IPC(1)}{2} - \frac{p \cdot L_0}{6} + \frac{1}{\sqrt[3]{2}} \cdot \sqrt[3]{\frac{\Delta IPC}{\mathcal{O}}}$
G	L_1	$T_C \cdot (1 + \mathcal{O} \cdot (N-1))$	$1 - IPC(1) + \sqrt{\frac{\Delta IPC}{\mathcal{O}}}$
H	L_1	$T_C \cdot \left(1 + \mathcal{O} \cdot \sqrt{N-1}\right)$	$1 + \sqrt[1.5]{\frac{2}{3}} \cdot \sqrt[1.5]{\frac{\Delta IPC}{\mathcal{O}}}$
I	L_1	$T_C \cdot (1 + \mathcal{O} \cdot log(N-1))$	$1 + \frac{\Delta IPC}{\mathcal{O}}$

4.3 Experimental Results

In this section, we verify the IPC model and the optimum instruction issue width expression described in Section 4.2 using the MXS superscalar simulator, the performance simulator

used in the IPLAN floorplanning framework. The details of the MXS simulator is discussed in Chapter 5 and also in [Ben98].

Our baseline processor model resembles a state-of-the-art superscalar processor. It encompasses a load/store buffer, a reorder buffer and an instruction window of 72 entries. The 64KB L1 cache is 2-way set associative with a line size of 32 bytes and a latency of 3 cycles. The 512KB L2 cache has the same line size and a read/write hit latency of 10/11 cycles. The latencies of the integer and floating-point units are shown in Table 4.3. The

Table 4.3: Functional Unit Latencies

Functional Unit	Latency (cycles)	Functional Unit	Latency (cycles)
Logic	1	FP Add/Subtract	4
Integer Add/Subtract	1	FP Multiply	4
Integer Multiply	4	FP Divide	17
Integer Divide	30	FP Square-Root	35

branch latencies are 1, 1, and 10 cycles for taken, fallen through, and mispredicted branch respectively. A memory read or write takes 36 processor cycles with a 64-bit bus.

We select a number of benchmarks from the SPEC benchmark suites, plus the one-dimensional FFT implementation, and the Linpack benchmark to evaluate the optimum issue width. Table 4.4 briefly describes each of these benchmarks.

Table 4.4: Benchmarks Simulated Using the MXS Simulation

Benchmark	Description	No. of Instructions
008.espresso	Logic optimization	35.6M
013.spice2g6	Circuit simulation	88.5M
022.li	LISP interpreter	12.4M
026.compress	File compression	2.5M
039.wave5	Electromagnetic wave analysis	30.8M
072.sc	Spreadsheet	23.9M
099.go	Game	80.0M
164.gzip	GNU compression utility	39.3M
256.bzip2	Compression utility	131.4M
FFT	Fast Fourier transform	1.1M
Linpacks	Linear algebra routines	67.3M

In this investigation, we examine the range of issue width from $N = 1$ to 8. In order to verify the processor IPC model, we first determine $IPC(\infty)$. Since the current MXS simulator does not model an infinite-width processor, we use $IPC(1)$, $IPC(8)$ and Equation 4.8 to estimate $IPC(\infty)$. The MXS simulation results are then compared with the IPC lower bound expression in (4.9). A typical example (039.wave5) is shown in Figure 4.9. As expected, the MXS simulation rises faster and saturates sooner than the IPC lower bound model. In fact, these graphs look very similar to Figure 4.4 in Section 4.2.1.

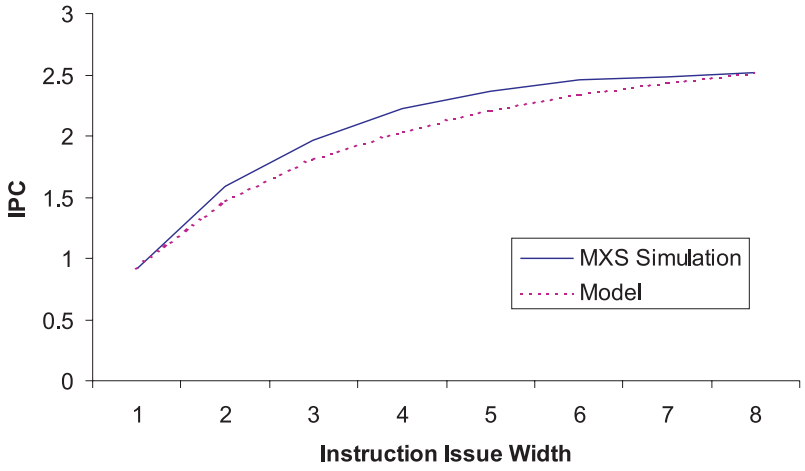


Figure 4.9: Comparison of the modeled and simulated instructions per cycle (IPC) vs instruction issue width (039.wave5).

The gaps between the MXS simulation and IPC lower bound model are small in some benchmarks and somewhat larger in other benchmarks. The percentage difference between the MXS simulation and IPC lower bound model is calculated by taking the largest difference between the two across the range of N studied. For the benchmarks used in this study, the percentage difference varies from about 2% to 12% (Figure 4.10). For instance, the percentage difference in 039.wave5 is around 9%.

Next, we consider the effects of interconnect and other complexity overheads. As described in the previous section, the overheads can affect either cycle time or instruction latency depending on the implementation. However, we have examined both effects and found that the results are similar. For brevity, we only show the results whereby the instruction issue width N affects only the cycle time. This corresponds to Models A, F, G,

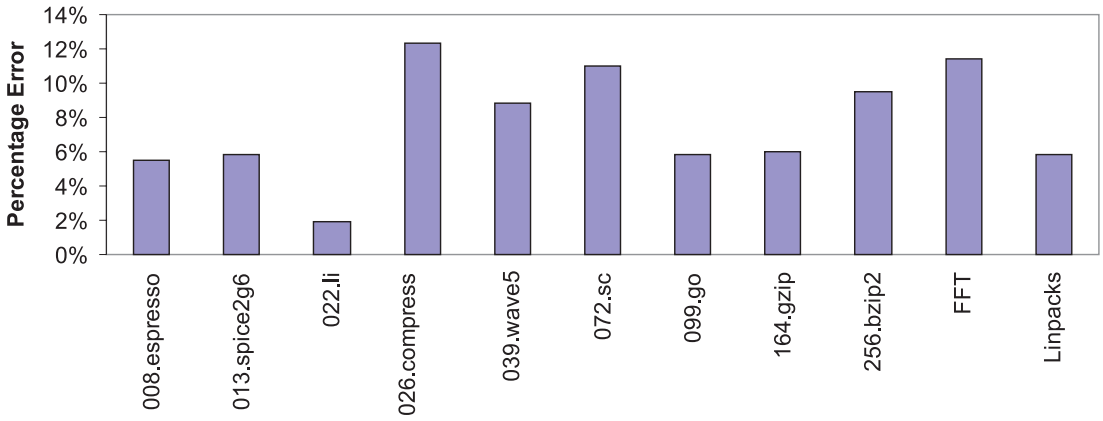


Figure 4.10: IPC percentage errors between the model and the MXS simulation.

H and I in Table 4.2.

Figure 4.11 shows the normalized processor performance as a function of instruction issue width for the 039.wave5 benchmark. The overhead scaling factor $\mathcal{O}$ for each overhead model is selected such that the normalized performance is between 100% to 200% for $N = 1$ to 4. The graphs on the LHS correspond to the performance based on the MXS simulation, while the graphs on the RHS correspond to the performance expression in (4.10). As expected, the graphs in the LHS and RHS look very similar. But more importantly, we should compare the optimum issue widths on both sides. For Models F and G, the optimum issue widths are identical on both sides (all equals 3). For Model H, the optimum issue width is 4 on the LHS and is 5 on the RHS. For Model I, the optimum issue width is 5 on the LHS and is 8 on the RHS. As derived in the previous section, the maxima in the performance expression (RHS) yields an upper bound on the optimum instruction width (LHS). In case of Models H and I, the graphs are quite flat from $N = 4$ to 8 so the LHS and RHS can more easily pick a different optimum point.

Similarly, Figure 4.11 shows the relative performance as a function of instruction issue width for the 022.li benchmark. In this case, the optimum points in the LHS and RHS are identical for all overhead models. While each benchmark may have a very different IPC at a given issue width, the expression for the upper bound on the optimum instruction issue appears to be fairly reliable for all the benchmarks. This upper bound is found to be very close (or identical) to the actual optimum width if the performance graph has a distinct

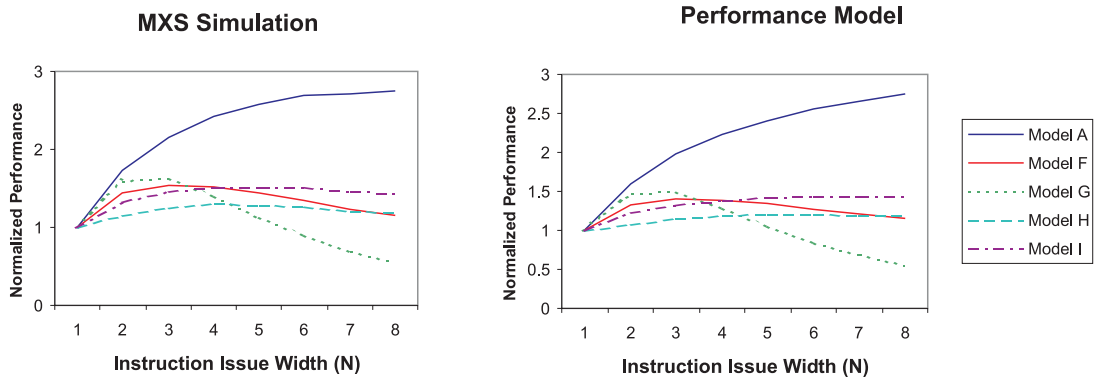


Figure 4.11: Comparison of the relative performance computed using various overhead models for the simulated IPC and modeled IPC using the 039.wave5 benchmark.

maxima point. Conversely, this upper bound is less tight if the performance graph is flat.

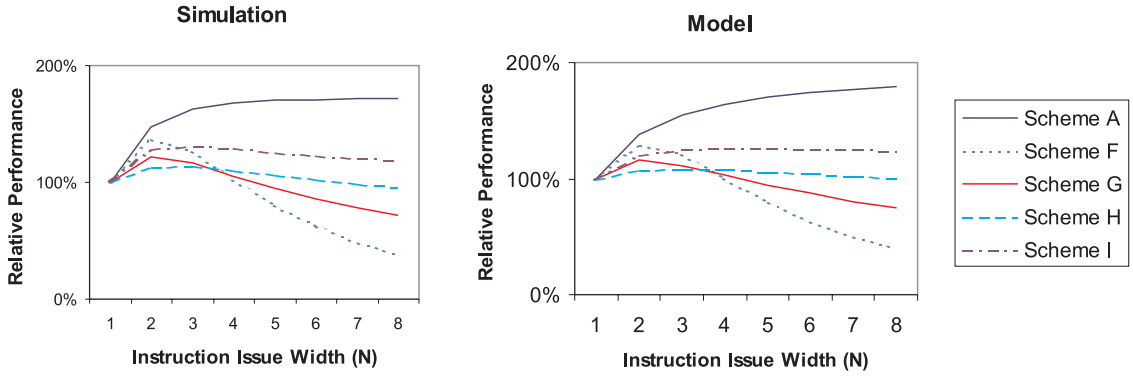


Figure 4.12: Comparison of the relative performance computed using various overhead models for the simulated IPC and modeled IPC based on 008.espresso benchmark.

Now, we have already validated the upper bound expression on the optimum instruction issue width. We use this expression to investigate the differences between benchmarks and the effects of the overhead scale factor $\mathcal{O}$. For simplicity, we assume that the upper bounds are very tight so we can approximate the actual optimum issue width with these upper bound expressions. As shown in Figure 4.13, the optimum instruction issue widths varies across the benchmarks (from 2 to 9 when $\prime = 10\%$). Some benchmarks (e.g. 256.bzip2, FFT, Linpacks) can more effectively take advantage of additional issue widths than other

benchmarks (e.g. 022.li, 013.spice2g6).

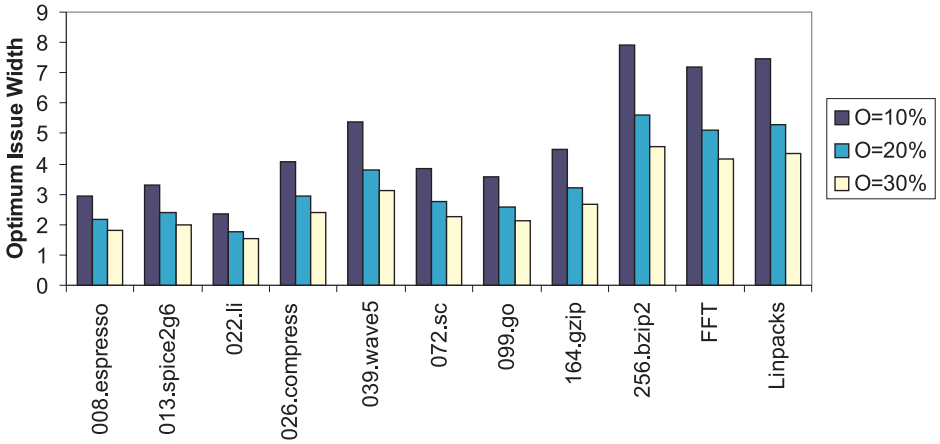


Figure 4.13: Optimum instruction issue width for overhead scale factor $\mathcal{O}$ = 10%, 20%, 30%.

Finally, Figures 4.14 and 4.15 show the relationship between the optimum issue width N_{opt} and the overhead scale factor $\mathcal{O}$ with different overhead models. Overhead models with a stronger dependence on the issue width N (Models F and G) are less susceptible to changes in the overhead scale factor $\mathcal{O}$. Above a certain value of $\mathcal{O}$, the optimum issue width is not very sensitive to any further increase in $\mathcal{O}$. For a wide range of $\mathcal{O}$ (20% to 30%), the optimum issue width is around 5 for 256.bzip2 and around 3 for 013.spice2g6.

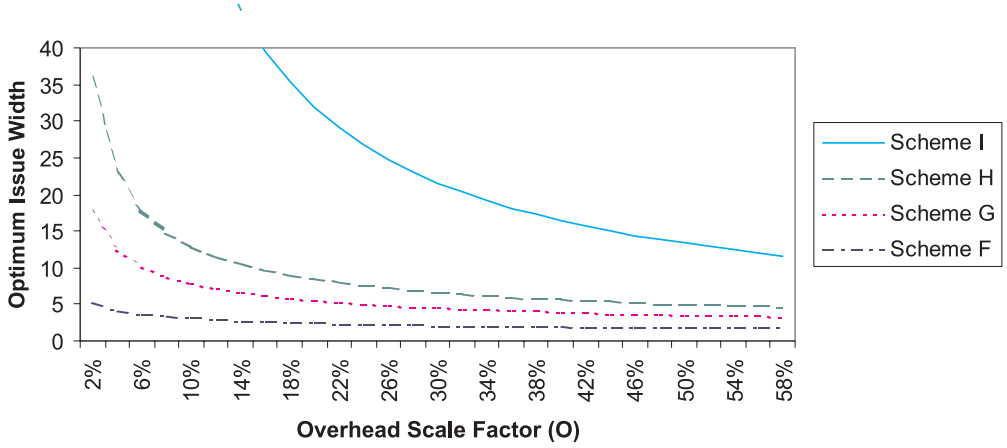


Figure 4.14: Optimum instruction issue width N_{opt} vs overhead scale factor $\mathcal{O}$ based on 256.bzip2 benchmark.

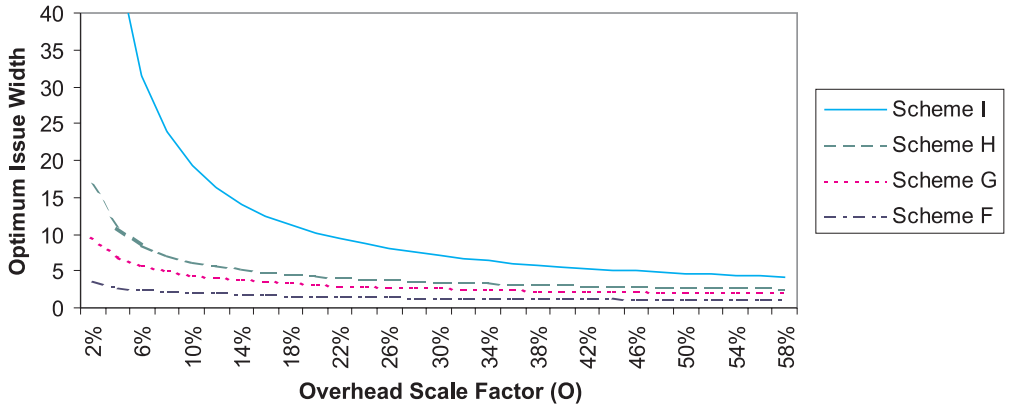


Figure 4.15: Optimum instruction issue width N_{opt} vs overhead scale factor $\mathcal{O}$ based on 013.spice2g6 benchmark.

4.4 Discussion

4.4.1 Approximation of N_{opt}

In the beginning of this chapter (Section 4.2.4), we derive a set of upper bound expressions on the optimum instruction issue width N_{opt} for a variety of overhead models. Can we use these expressions to approximate N_{opt}? As described in Section 4.3, these upper bounds are mostly identical or very close to the actual optimum issue width. The only exception is when the performance is flat near its peak. However, when the performance graph is very flat, it may not be necessary to identify the precise optimum point.

Hence, the upper bound expressions can also serve as an estimation of the optimum instruction issue width. Thus, we can rewrite (4.14) as (4.15). As before, α depends on the overhead model and is between $\frac{1}{3}$ and 1.

$$N_{opt} \approx \left(\frac{\Delta IPC}{\mathcal{O}}\right)^{\alpha} \tag{4.15}$$

4.4.2 Effects of the Processor Organization

The processor organization plays an important role in determining the optimum instruction issue width N_{opt}. Parameters such as the instruction window size, cache size, memory latency, and functional units affect the available instruction level parallelism, which in turn affects the optimum instruction issue width (N_{opt}).

For a fixed instruction issue width the IPC increases asymptotically to a maximum value with increasing instruction window size. Therefore, the performance effect of increasing the window size quickly saturates. Rudd [Rud99] determined that for VLIW processors the instruction window size should be at least four times the instruction issue size. Figure 4.16 illustrates that when the window size is less than four times the issue width, ΔIPC is smaller and the optimum instruction issue width (N_{opt}) should be smaller.

Similarly, cache size as well as memory and execution latencies affects the average instruction latency. For example, if the cache size is small, memory latency becomes the processor bottleneck and the optimum instruction issue width is smaller. Figure 4.17 shows that for a smaller cache size, ΔIPC is smaller and the optimum issue width is also smaller. On the other hand, techniques such as prediction caches [Ben98] can be used to hide the memory latency and can therefore increase the optimum issue width.

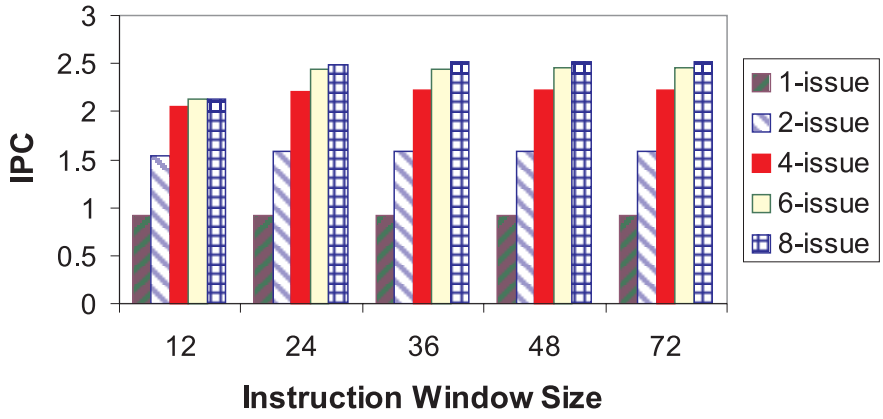


Figure 4.16: The instructions per cycle (IPC) as a function of window size and instruction issue width.

4.4.3 Technology Trends

As semiconductor process technology changes, the appropriate overhead models and their effects on the optimum instruction issue width also change. The appropriate overhead model and the overhead scale factor $\mathcal{O}$ depend on the feature size.

In general, interconnect delay is increasingly more dominant than gate delay. For example in 0.8 μm process technology, total delay is primarily determined by gate delay, which usually grows logarithmically or linearly with the issue width N. The reverse is true in 0.18 μm process technology, when the total delay is driven by the interconnect delay, which grows quadratically with interconnect length [Naf99]. To further examine the effects of process technology, we again use the register file access time as our example.

As described in Section 4.2.3, a linear overhead model is used to model the effects of instruction issue width. Now, we can apply the scaling model and calculate the overhead scale factor $\mathcal{O}$ as a function of feature size. Figure 4.18 shows the relationship between $\mathcal{O}$ and feature size (drawn in semilog scale). The increase in $\mathcal{O}$ for future technologies demonstrates that the effects of overhead associated with the instruction issue width are becoming more important. If we assume ΔIPC to remain unchanged in the future, the optimum instruction issue width N_{opt} appears to decrease at smaller feature sizes.

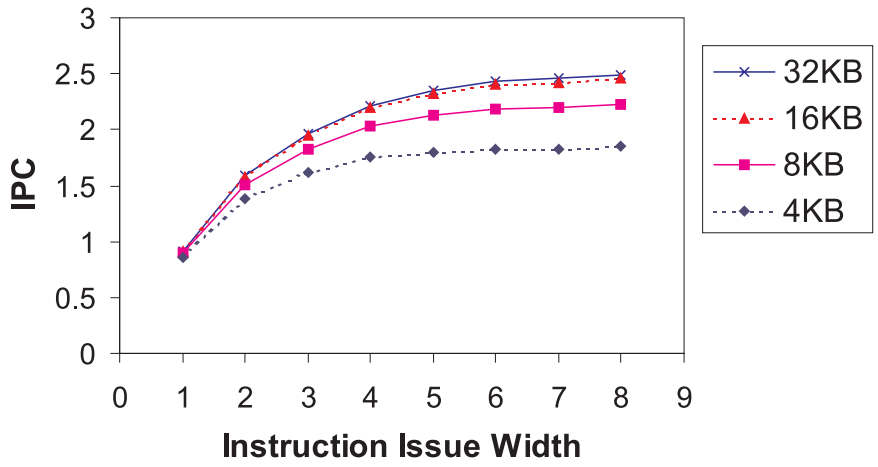


Figure 4.17: The instructions per cycle (IPC) as a function of cache size and instruction issue width.

4.5 Summary

We present a few concluding remarks in this section.

- We derive a set of upper bound expressions on the optimum instruction issue width based on different types of complexity overheads. In general, $N \leq (\frac{\Delta IPC}{O})^{\alpha}$ where $\frac{1}{3} \leq \alpha \leq 1$. This upper bounds are found to be fairly tightly, and therefore can also serve as an approximation. We examine a number of critical pipeline structures including register file and wakeup logic and determine that $N_{opt} \approx \sqrt{\frac{\Delta IPC}{O}}$ in most cases.

- For a target application, the inherent instruction level parallelism (ILP) can be found by compiler and software simulations. Some processor architects may tend to design the instruction issue width based on the available ILP of the application. However, it is important to realize that the overhead (dominated by interconnect delay) can limit the overall performance.

- Simply matching issue width to the available ILP does not always achieve the optimum performance. For instance, assume that $IPC(\infty) = 8$ and $IPC(1) = 0.9$. Using the overhead scale factor $\mathcal{O} = 17\%$ (Section 4.2.3), the optimum instruction issue width

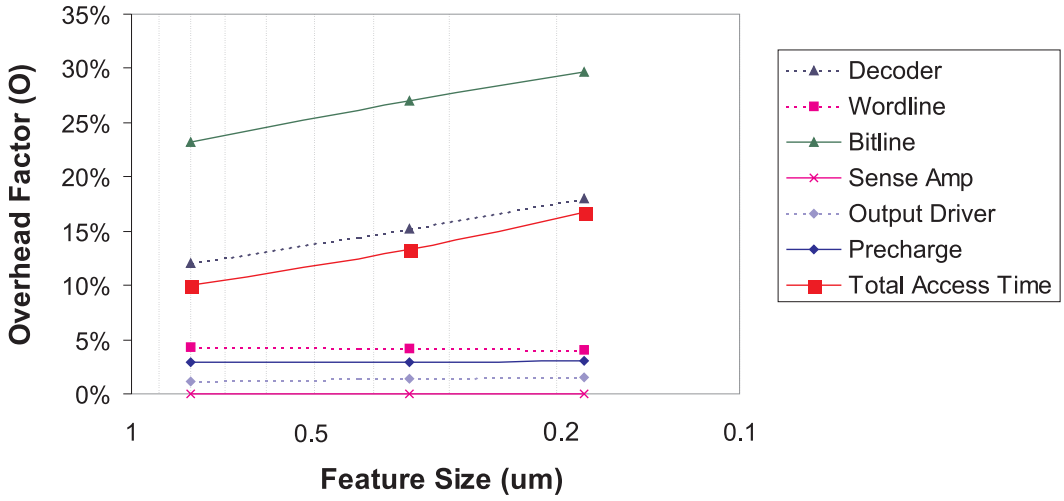


Figure 4.18: Overhead scale factor $\mathcal{O}$ vs feature size.

N_{opt} is 6. Consequently, an issue width of 6 is more cost effective and can deliver better performance than an issue width of 8.

- Alternatively, if $\mathcal{O}$ is low it may be desirable to have an issue width greater than the available ILP. For instance, if $IPC(\infty) = 4$, $IPC(1) = 0.9$, and $\mathcal{O} = 10\%$, N_{opt} is equal to 5. This implies that if cost and power consumption are not an issue, it is possible to achieve better performance by allowing a higher issue rate than the available ILP of the target program. Actually, the high-end superscalar and VLIW processors probably fall in this category.

- In general, the overhead scale factor $\mathcal{O}$ for VLIW processors is less than that for dynamically scheduled superscalar processors because the issue logic is simpler. VLIW machines are thus expected to issue more instructions per cycle. However, it is important to note that even if the overhead scale factor $\mathcal{O}$ in a VLIW processor is half of that in a superscalar processor, N_{opt} is only $\sqrt{2} \approx 1.4$ times its counterpart.

- Advanced compiler techniques [WFW+94] have been used to improve the ILP in a program. These techniques are attractive because they do not incur any hardware cost or overhead. However, as in the case of VLIW, if the IPC is improved by 100%, N_{opt} is only increased by around 40%.

- The goal of Multicluster [FCJV97], Multiscalar [SBV95], and Multiflow [LFK+93] architectures is to minimize the inherent overhead scale factor $\mathcal{O}$ and thus improve the overall performance. However, the partitioning of resources in these architectures may have a negative impact on IPC. It is apparent from the N_{opt} equation that if the reduction in $\mathcal{O}$ is more significant than the drop in ΔIPC, these architectures can issue instructions more effectively than traditional superscalar architecture.

- Simultaneously multithreaded (SMT) [TEE+96] processors are capable of executing multiple sequences of instructions with minimum hardware overhead. The challenges facing SMT architectures are related to cache pollution and memory contention, which are not carefully studied in our model. The SMT architecture has a very small overhead scale factor $\mathcal{O}$, and therefore appears to be an interesting research area.

Chapter 5

IPLAN Floorplanning Framework

5.1 Introduction

As VLSI technology advances, designers are incorporating multiple processors, memory subsystem and special hardware on a single die, forming a system-on-chip (SOC) [FHR99, Hen99]. The Texas Instrument's Open Multimedia Applications Platform (OMAP) [Tex], a typical SOC shipped today, encompasses a Texas Instrument's DSP core, an ARM's RISC processor core [ARM] and an integrated modem.

The SOC design complexity has reached a point where it is essential to integrate predesigned, synthesizable IP's from multiple sources. While industry-wide IP standards ensures signal compatibility among different sources [Bri01, Vir], SOC designers find it increasingly difficult to meet all design criteria, such as clock speed and power consumption, within a very tight design cycle [Hen99]. The conventional design flow, which separates system-level design from physical floorplanning, becomes ineffective in the deep submicron era. This is one of the key reasons why most SOC's today can hardly exceed 400MHz, while custom-made Intel's Pentium IV can run faster than 2.5GHz [Mic02].

Our research addresses this important problem by developing an interconnect-driven floorplanning framework, performing system-level design tradeoffs and chip floorplanning in the early design stages. Figure 5.1 shows a simplified block diagram of the IPLAN floorplanning framework. The floorplanning framework is composed of the MXS performance simulator [Ben98], the area/delay estimator [FHP00], the interconnect estimator, and the floorplan optimizer [HF99]. The MXS performance estimator calculates the number of clock

cycles required to run a benchmark. It can be used in conjunction with the interconnect-driven processor performance model (described in Chapter 4) to optimize instruction issue width. The area/delay estimator predicts the size and latency of a functional block. The interconnect estimator calculates the interconnect delay and power consumption using the estimated interconnect wirelength and congestion. The results of the three estimators are sent to the floorplan optimizer, which determines an optimal floorplan based on the user's constraints and requirements. In the next section, these four components are discussed in more detail.

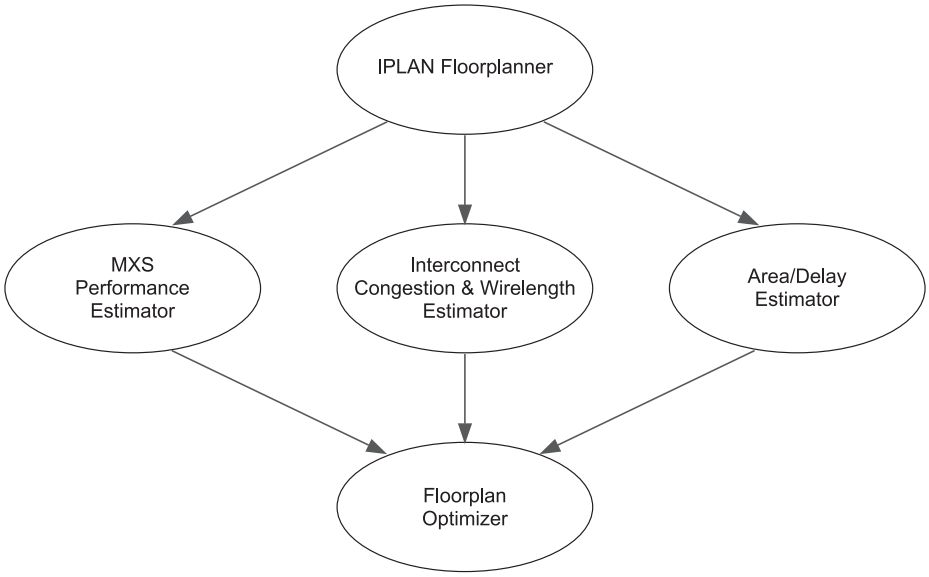


Figure 5.1: Simplified Block Diagram of IPLAN Floorplanning Framework

In this chapter a commercial synthesizable processor core (Lexra's LX4380) is used as an example to illustrate the functionalities of the IPLAN components and to show how these components work together in a unified framework. The Lexra's LX4380 is chosen because it is a fairly popular SOC processor core. Its architecture and floorplan were described in the literature [Sny01, HC01]. The Lexra's LX4380 is a 32-bit, seven-stage pipelined scalar processor, with a three-port register file, and a Harvard bus structure. It executes the MIPS-I instruction set and runs at 266MHz in a 0.18-micron technology. The processor core can further be extended with coprocessors and custom logic. Since the detailed microarchitecture specifications are not publicly available, we have made some general assumptions, including the number of interconnections among the functional blocks

and the timing constraint within each functional block.

5.2 Components in IPLAN Framework

This section describes the four key components in IPLAN floorplanning framework: the MXS performance simulator, the area/delay estimator, the interconnect estimator, and the floorplan optimizer.

5.2.1 MXS Performance Simulator

There is a tradeoff between simulation accuracy and simulation performance. In general, a more accurate simulation usually takes a longer simulation time. There is a wide range of techniques available for performance evaluation. These techniques may be broadly classified into four categories: analytical models, trace-driven simulation, execution-driven simulation and hardware-driven simulation.

Analytical models are the fastest but the least detailed method among all the simulation styles. The processor performance model discussed in Chapter 4 is an analytical model, in which the IPC of a processor may be estimated based on a few input parameters. While the analytical models are very straightforward, they do not encapsulate all implementation details, resulting in estimation errors. In trace-driven simulation, the output of an application is used to drive the simulation. As such, the simulation result is more accurate than analytical models. A survey of different trace-driven simulation techniques by Uhlig and Mudge may be found in [UM97].

In execution-driven simulation, the application drives the simulation directly. In this style of simulation, the simulator reads the application into memory and emulate every operation in an instruction. As a result, the simulation accuracy is better than the previous two techniques at the expense of slower simulation performance. Finally, hardware-driven simulation employs hardware techniques to collect simulation statistics. For instance, some processors are equipped with hardware event counters, collecting vital processor information. While the hardware-driven simulation is very accurate, they tend to be the least flexible. Generally speaking, hardware-driven simulation techniques are used for verifying the hardware behaviors but they are not suitable for evaluating tradeoffs in the early design stages. In the IPLAN floorplanning framework, it is important to maintain a high degree of both flexibility and accuracy. Based on these requirements, an execution-driven processor

simulator – MXS Simulator – is chosen.

The MXS processor performance simulator is an C-based execution-driven superscalar processor simulator, originally developed by James Bennett in the SGI IRIX operating system environment [Ben98]. As the MXS simulator is execution-driven, it can accurately model instruction timing, cache system, memory system, and other critical processor components, such as translation lookaside buffer (TLB). Figure 5.2 shows the block diagram of the MXS Simulator. The benchmark program is first processed by the MXS compiler, performing simple static instruction scheduling. The compiled benchmark is then read by the MXS simulator that simulates each instruction in the program. During the simulation, the useful statistics are written to output files.

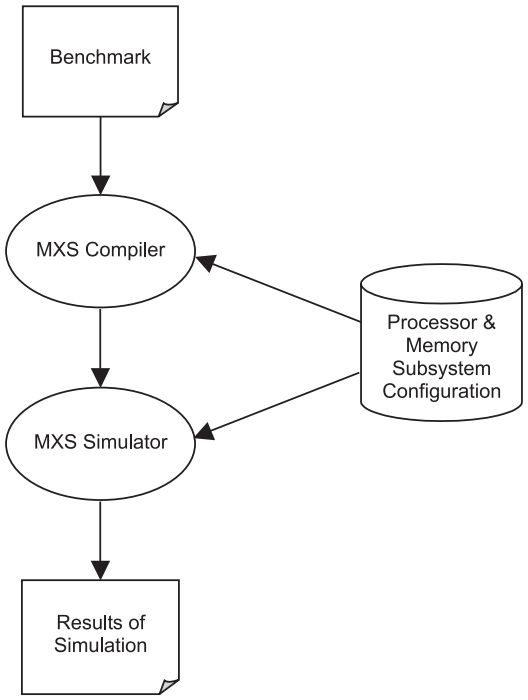


Figure 5.2: Simplified MXS Simulator Block Diagram

In order to achieve an efficient simulation performance, the MXS simulator combines a cycle-based simulation with an event driven simulation. Every cycle the simulator performs a number of regular tasks, such as instruction fetch, register renaming, instruction issue, memory access, register writeback and instruction graduation. In addition, it also checks for an event queue that handles events that completes in that cycle. Figure 5.3 shows a simplified diagram of the simulation architecture.

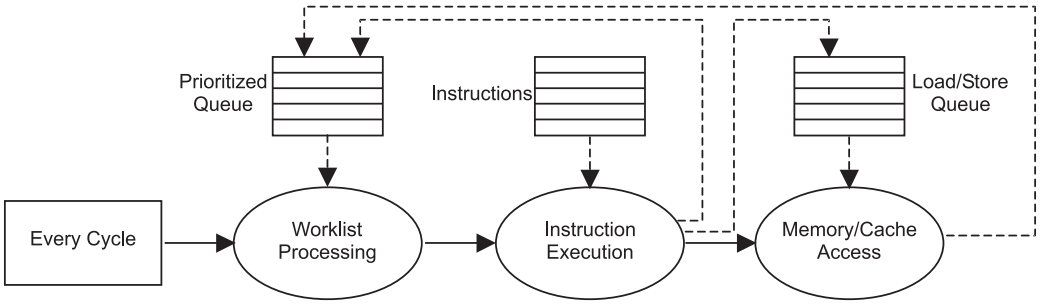


Figure 5.3: Simplified MXS Simulation Architecture

The MXS simulator was slightly modified and adapted to the IPLAN floorplanning framework. In order to work with other tools in the framework, the MXS simulator was ported from the SGI Irix operating system to the Solaris and Linux operating systems, Some useful features and functionalities were also added. For example, only a single-level data cache system was modeled in the original MXS simulator. To improve its usability, a configurable instruction cache and a two-level cache simulation (based on the inclusion principle) [Fly95] were included in the MXS simulator. In the past, all the processor parameters are hard-coded in the header files, which means that the simulator has to be re-compiled every time any processor parameter is altered. In the IPLAN framework, a Motif graphical user interface (GUI) was added to allow users to modify any processor parameters on-the-fly. Figure 5.3 shows the GUI in SUN/Solaris. Before simulation, MXS reads the user configurations defining the processor microarchitecture, such as issue width, bandwidth, latency and branch policy. The MXS simulator emulates the processor instruction fetch, instruction decode, register renaming, instruction execution, memory load/store and register writeback. At the end of simulation, MXS reports the number of cycles and other statistics in the GUI. The statistical information is also written to an output file to be read by the floorplan optimizer.

In system-level design, it is crucial to simulate the entire system, including multiple processors, memories and hardware IP's. As a result, it is often necessary to co-simulate different system-level components at the same time using multiple simulators. Since the MXS simulator is a C-based simulator, it can easily simulate in conjunction with other C/C++ based simulators, such as the SIMOS simulator [RHWG95] and the SystemC simulator [Sys]. The SIMOS simulator provides an accurate and flexible simulation environment for both the hardware and the operating system. The MXS simulator was incorporated

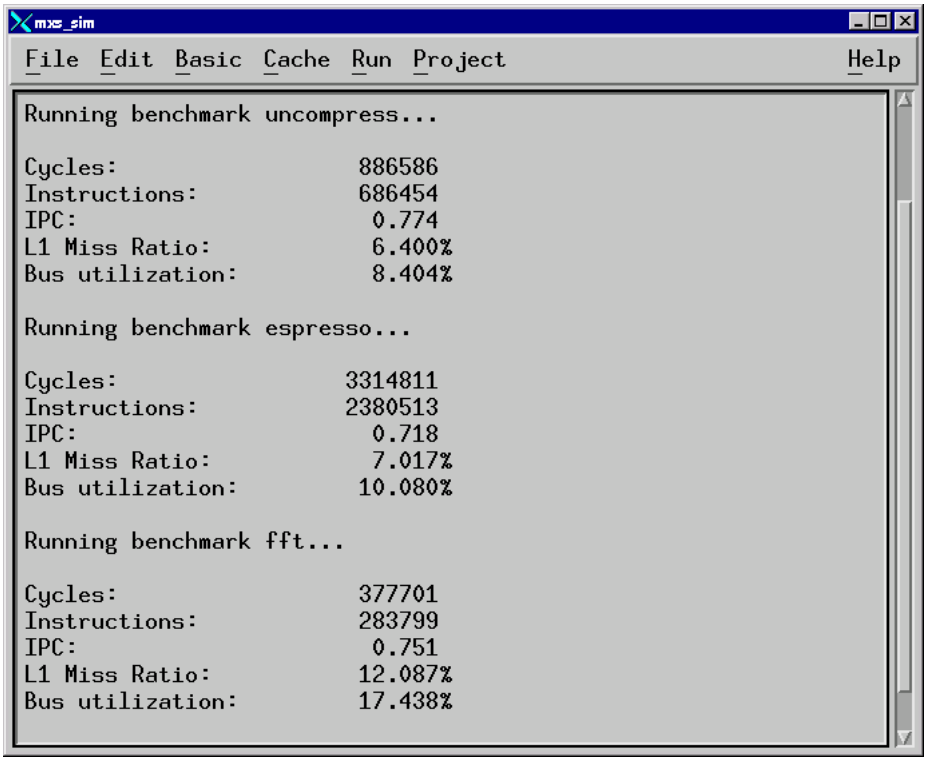


Figure 5.4: MXS Performance Simulator Graphical User Interface

into the SIMOS simulation environment, as its most detailed simulation engine. Similarly, the SystemC simulator can work with the MXS simulator in a co-simulation environment. SystemC, which consists of an open-source C++ library and a simulation kernel, is the de facto system level description language [Sys]. Other system-level C/C++ based simulators include the Cadence Cierto VCC simulator [Cad] and the CynLib and SpecC simulators [FH01].

5.2.2 Area/Delay Estimator

The area/latency estimator is built upon the research works by McFarland [McF97] and Fu [Fu99]. McFarland developed a technology scaling model for digital CMOS circuits by examining the electrical limits of MOSFETs, including subthreshold leakage, short channel effects, gate induced drain leakage, gate tunneling current, time dependent dielectric breakdown and hot carrier effects. McFarland's model is accurate to about 0.1 micron feature size. Fu extended McFarland's model by investigating the area-time relationship for floating

point units and on-chip storage.

The area/latency estimator encompasses a library of area/latency models for major functional units in a processor, including the register file, cache, translation-lookaside buffer (TLB), floating-point unit (FPU), etc. For example, the estimator can predict the cache area and latency by simply specifying the cache size, the cache configurations and the VLSI technology. Conversely, given the target application, the feature size and the available FPU die area, the estimator can also suggest the optimal hardware algorithms and latencies for different FPU operations (e.g. floating-point addition, multiplication, etc.) [Fu99].

5.2.3 Interconnect Estimator and Floorplan Optimizer

After the area of each functional block is estimated, the functional blocks are input to the front-end floorplan graphical user interface (GUI). The IPLAN floorplan GUI was written in Tcl/Tk, and the floorplanner engine (interconnect estimator and floorplan optimizer) were written in C. The interconnect estimator is responsible for estimating the wirelength within each functional block and wire congestion among the functional blocks, while the floorplan optimizer is responsible for determining an optimal floorplan based on a user-defined objective function. The mktclapp utility is used to simplify the interface between the Tcl/Tk and the C programs.

The GUI allows user to manipulate block placements and the interconnections. Users can easily create/delete blocks and interconnections, modify their sizes and positions. The placement and interconnection information is passed to the floorplanner engine to perform floorplan optimization. The interconnect estimator can model via minimization, routing obstacles, rectilinear blocks, routing among blocks and ports. Figure 5.5 shows two routing obstacles (Obstacle1 and Obstacle2) blocking the interconnections between two functional blocks (Blk0 and Blk1). The congestion estimator predicts some routing congestion (depicted in dark squares) near Obstacle1.

As described in the previous section, Lexra's LX4380 is used as an example to illustrate how the floorplanner works. In Figure 5.6, the functional blocks and interconnections in Lexra's LX4380 are manually input to the floorplanner GUI. The LX4380 chip comprises of 12 functional blocks: RPA, DCACHE, IMU, MMU, CE, COP, IF, REGFILE, JTAG, DMATCH, IMATCH and CBI. The RPA is the LX4380's core, which consists of the main controller and its execution units. The REGFILE is the processor register file. The DCACHE and IMU are the data cache and the instruction fetch/cache unit. The MMU

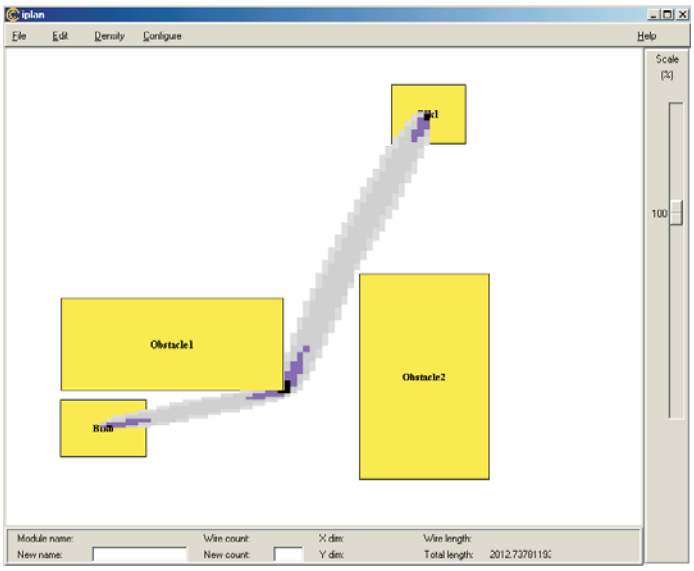


Figure 5.5: IPLAN Routing Obstacle Example

consists of the translation lookaside buffer (TLB) and the memory management unit. The COP, CE and IF are the coprocessor and the extension interfaces. The JTAG, IMATCH, DMATCH and CBI are the extended JTAG blocks for on-chip testing. The block placements are optimized at this point. In this example, the sizes of the functional blocks were taken directly from the published data, but it is also possible to use the area/delay estimator to predict their areas.

Using the area and wirelength objective function described in Section 3.3, the chip area and the total wirelength can both be minimized by the floorplan optimizer. The floorplan optimizer was based on the algorithm developed by Maggie Kang [Mag98b, Mag98a] and integrated into the IPLAN floorplanning framework. It uses simulated annealing technique to constantly change the floorplan and search for an optimal solution. In this example, there are only 12 functional blocks and the floorplan optimizer only takes a few seconds using a 300MHz SUN Ultra 10 workstation. Figure 5.7 shows the floorplan after the optimization. Internally the floorplan is represented by the sequence pair ((DCACHE, IF, REGFILE, RPA, CE, COP, IMU, DMATCH, JTAG, CBI, IMATCH, MMU), (IMU, CE, COP, RPA, DCACHE, IF, REGFILE, MMU, IMATCH, JTAG, CBI, DMATCH)) [MFNK95].

In previous floorplan, there is a tiny space in the middle of the RPA, REGFILE, JTAG, DMATCH and MMU blocks. The floorplan can further be compacted by changing the

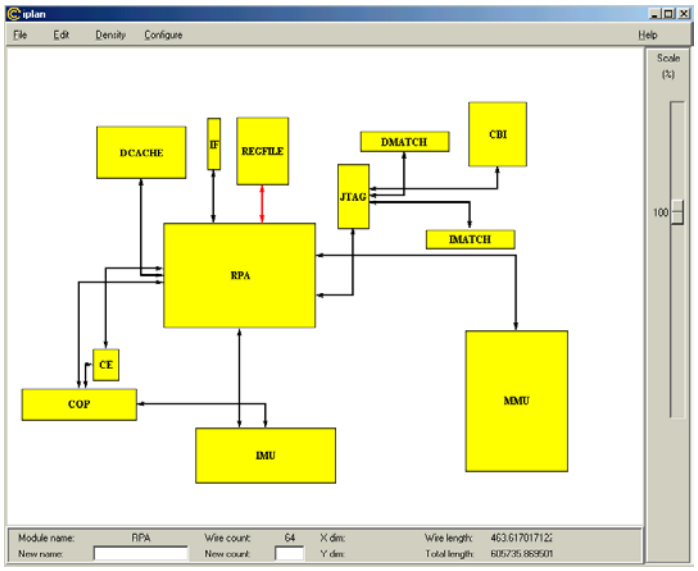


Figure 5.6: IPLAN Interconnection Example (Lexra's LX4380)

RPA block from rectangular shape to rectilinear shape (Figure 5.8). Please note that this is possible only because the RPA block is a synthesizable, soft IP block. Conceptually, the rectilinear RPA block is basically composed of three horizontal rectangular blocks.

Figure 5.9 describes the sequence pair structure in Figure 5.7. Basically, each sequence pair defines the relative positions among the blocks. In this example, there are three blocks (DCACHE, IF, REGFILE) located on the top of the RPA block, five blocks (DMATCH, CBI, JTAG, IMATCH, MMU) located on the right-hand-side of the RPA block, and three blocks (IMU, CE, COP) located on the bottom of the RPA block. During the optimization process, the sequence pair is constantly mutated in order to find an optimal floorplan with minimum area and wirelength.

The width and the height of the floorplan can easily be calculated from the sequence pair. Figure 5.10 shows the horizontal floorplan constraint graph, in which the longest path between the end points determines the floorplan width. In this example, the floorplan width is the sum of the widths of the IMU and MMU blocks. Similarly, Figure 5.11 shows the vertical floorplan constraint graph, in which the longest path between the end points determines the floorplan height. In this example, the floorplan height is the sum of the heights of the IMU, COP, RPA and REGFILE blocks.

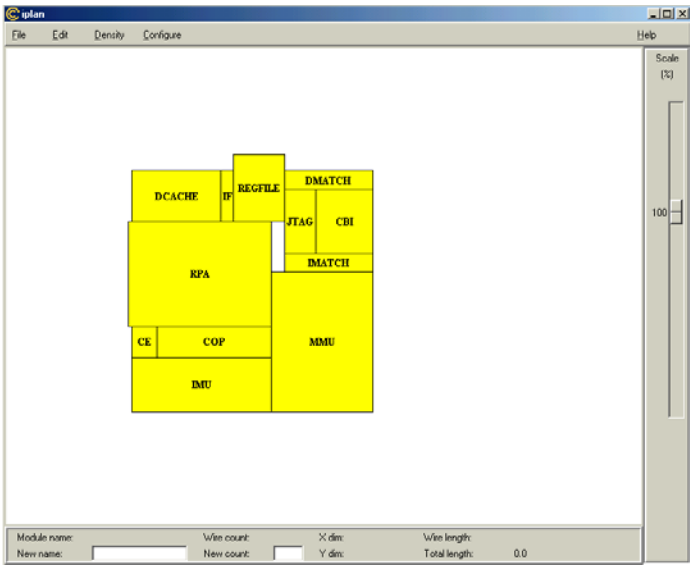


Figure 5.7: LX4380's Floorplan after Optimizaton

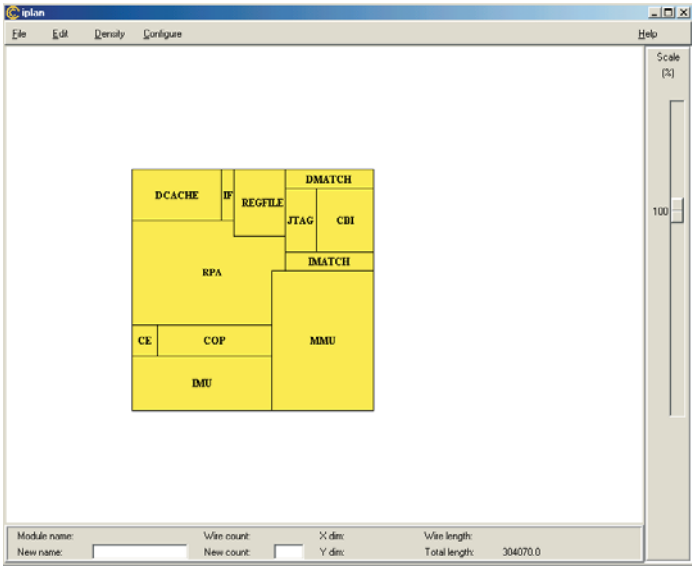


Figure 5.8: LX4380's Floorplan after Optimization and Compaction

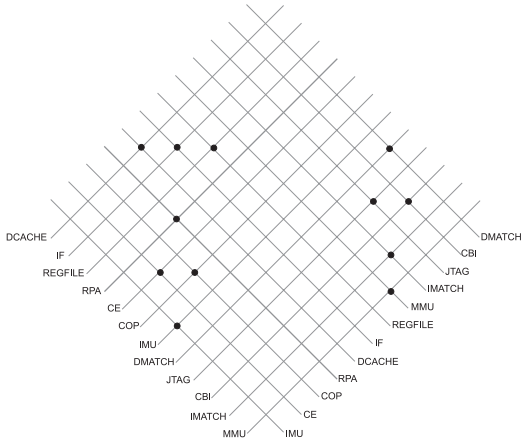


Figure 5.9: Sequence Pair Representing LX4380's Floorplan

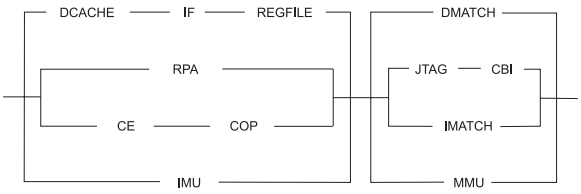


Figure 5.10: Floorplan Horizontal Constraint Graph

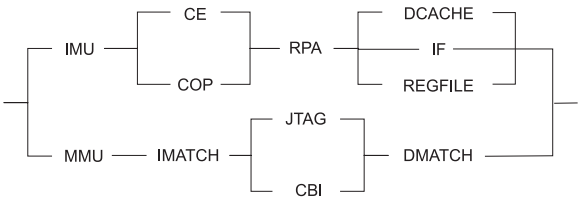


Figure 5.11: Floorplan Vertical Constraint Graph

5.3 Case Study

In this section, a design scenario is fabricated to demonstrate how the different IPLAN components work together in a unified framework. The size of the Lexra's LX4380 core is $1.8mm^2$ ($1.3mm \times 1.3mm$) in a 0.18 micron technology [Sny01]. Suppose we would like to develop an enhanced version of LX4380 with a bigger core area. What is the most effective way to utilize the extra real estate?

A simplistic approach is increase the cache size. The data cache size in LX4380 is 32KB. By increasing the data cache from 32KB to 64KB, the data cache miss rate can be further reduced. Using the IPLAN floorplanning framework, we can build a virtual "prototype" and evaluate its architecture. The first step in the IPLAN framework is to run the MXS simulator to calculate the IPC. Table 5.3 shows the baseline processor parameters in our evaluation. Please note that these processor parameters are based on our assumptions and can be very different from any particular chip design.

Table 5.1: Lexra's LX4380 Processor Core Baseline Parameters

Parameter	Baseline Value
Fetch # per cycle	1
Issue # per cycle	1
Writeback # per cycle	1
Dynamic Scheduling	yes
Cache Level	1
Cache Size	32KB
Cache Associativity	4-way set associative
Cache Policy	writeback/write-allocate
Cache Hit Latency	1 cycle
Memory Latency	20 cycles
Bus Width	64 bits
Bus Latency	4 cycles
Integer Add/Subtract	1 cycle
Integer Multiply	4 cycles
Integer Divide	30 cycles
FP Add	2 cycles
FP Multiply	3 cycles
FP Divide	12 cycles
Branch	2 cycles
Branch History Table Size	1024

Using the SPEC benchmarks, we run the MXS simulator and compare the miss rates at 32KB and 64KB cache sizes. As shown in Figure 5.12, The effects on the miss rate and IPC depend heavily on the target benchmark.

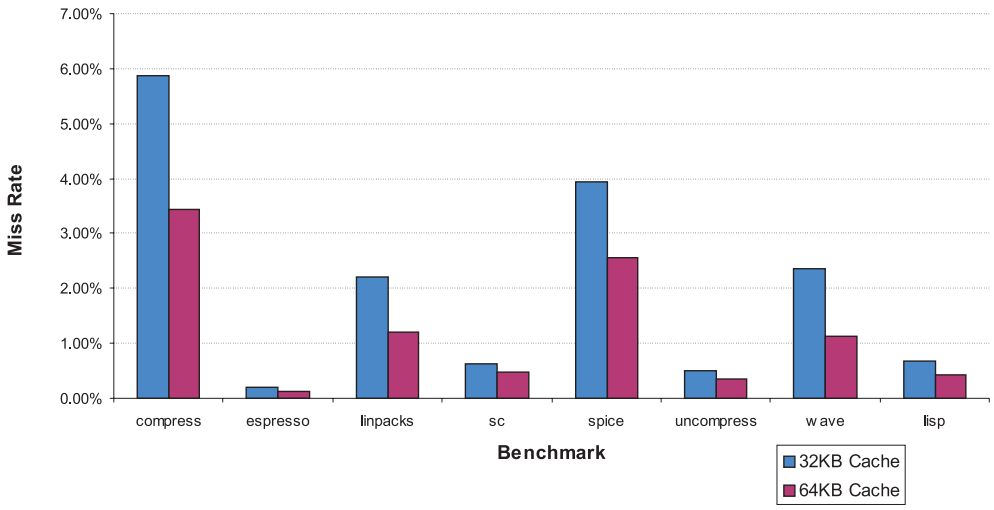


Figure 5.12: Miss Rates with 32KB and 64KB Cache Sizes

In case of the espresso, sc, uncompress and lisp benchmarks, the miss rates with 32KB data cache are already very small (less than 1%). For these types of target applications, increasing the data cache size can only yield marginal IPC improvements. On the other hand, the compress and spice benchmarks have higher miss rates and can more effectively take advantage of the additional cache size. For example, the IPC is increased from 0.753 to 0.817 for the compress benchmark (Table 5.3).

Table 5.2: MXS Simulation Results for the Compress Benchmark

	Original Core	Non-pipelined 64KB Cache
Number of Instructions	2.13M	2.13M
Number of Cycles	2.83M	2.61M
IPC (compress)	0.753	0.817
Miss Rate	5.88%	3.79%
Bus Utilization	10.2%	7.0%

The second step in the IPLAN framework is to use the area/delay estimator to calculate the new cache area and cache latency. In general, the cache area is roughly proportional to the cache size, so the 64KB cache area is approximately double the 32KB cache area. The cache latency has a logarithmic relationship with respect to the cache size. Figure 5.13 shows that the 64KB cache is about 10% slower than the 32KB cache.

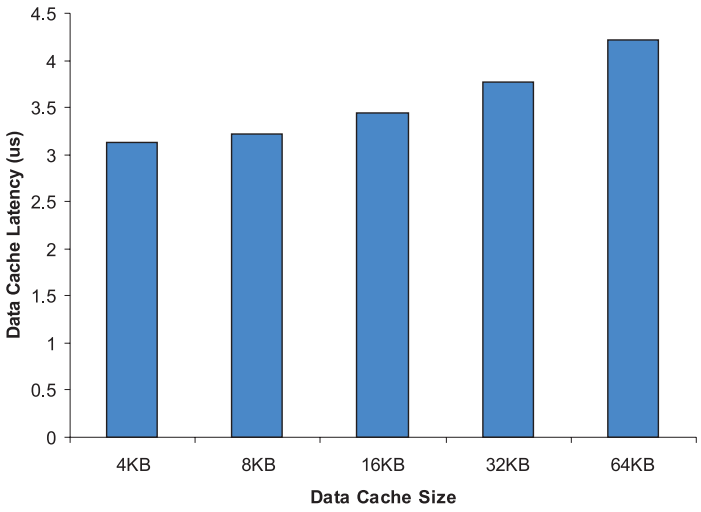


Figure 5.13: Cache Latency vs Cache Size

After performing the performance and the area/delay estimation, the last step is to run the floorplanner, which calculate the interconnect delay and determines an optimal floorplan. Figure 5.14 shows an optimized floorplan with 64KB data cache. The aspect ratios and the shapes of the RPA and the IF blocks were slightly altered manually to make the floorplan more compact.

The new processor core area is only slightly bigger than the original one with $2.1mm^2$ ($1.6mm \times 1.3mm$). With the increased cache size, the critical path in this design is the data transfer between the DCACHE and the RPA blocks. Since these two blocks are connected side by side, the interconnect delay is primarily caused by the local interconnect in the synthesized RPA block. The interconnect delay is estimated based on the wirelength distribution model, and assuming that the local interconnect width and spacing are $0.27\mu m$ and $0.18\mu m$, respectively.

Table 5.3 summarizes the performance of the new and the original architecture. Even

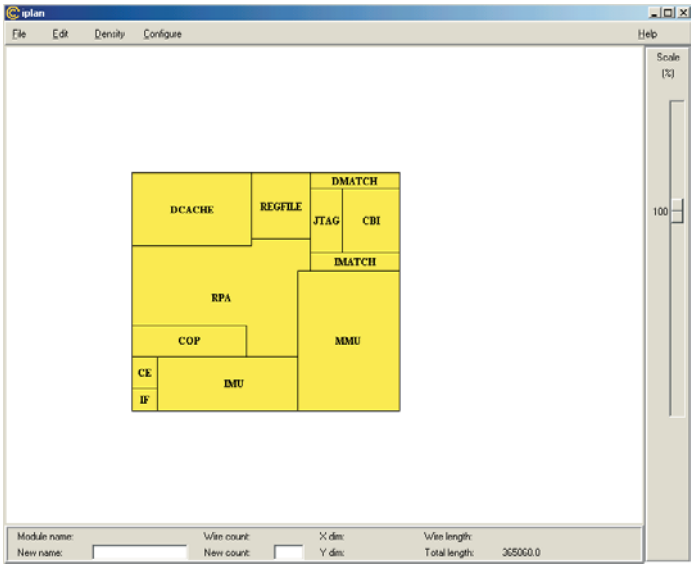


Figure 5.14: Optimized Floorplan with a 64KB Data Cache

if we run a benchmark that requires more cache size (e.g. compress), the additional cache size actually has a negative impact on the performance! Th interconnect delay in a larger cache slows down the processor clock, and the increase in IPC is insufficient to offset the decrease in the clock speed.

Table 5.3: Comparison between the Original Architecture and the Modified Architecture with a Non-pipelined 64KB Data Cache

	Original Core	Non-pipelined 64KB Cache
Die Size	$1.8mm^2$	$2.1mm^2$
Cache Latency	3.6 ns	4.1 ns
Interconnect Delay	0.1 ns	0.1 ns
Maximum Clock Frequency	267 MHz	238 MHz
IPC (compress)	0.753	0.817
Relative Performance	1.00	0.97

An obvious alternative to the previous approach is to use a two-stage pipelined cache. This technique is actually used in a number of other Lexra's chips [Sny01]. Figure 5.15 compares the block diagrams of a non-pipelined cache and a two-stage pipelined cache. As shown in the diagram, there are additional latches in the pipelined cache. Here, we assume

that the additional area for these latches is negligible so we can use the same floorplan as before. Although the two-stage pipelined cache incurs an extra clock cycle for cache access, this latency can be usually be tolerated and hidden in the processor pipeline.

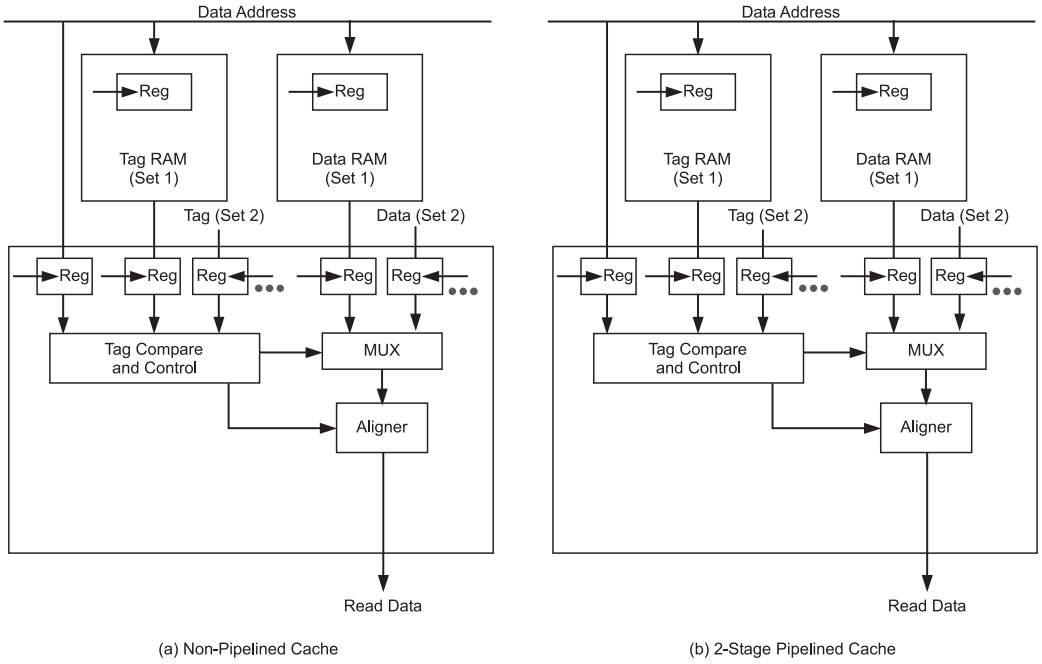


Figure 5.15: Block Diagrams of Non-pipelined Cache and 2-Stage Pipelined Cache

Using the MXS simulator, the IPC reduction caused by the extra cache cycle is found to be between 2% to 4% for the above benchmark programs. Table 5.3 details the performance of the pipelined cache. While the performance improvement is only 4%, the reduction in bus utilization is more than 30%. When there are multiple bus masters (e.g. processors or DMA controller) in the system, this reduction in bus utilization can be very important to the overall system-level performance.

The previous exercise reinforces the fact that increasing the cache size alone cannot provide very significant performance improvement. A more effective approach is to use a superscalar processor architecture. As discussed in Chapter 4, the optimum instruction issue width can be approximated by $\sqrt{\frac{\Delta IPC}{\mathcal{O}}}$ where $\mathcal{O}$ is between 10% to 20%. In general, ΔIPC can be found by running the MXS simulator. For the SPEC benchmarks described in Chapter 4, we can simply use the results in Chapter 4 and the optimum issue widths vary between 2 and 6. As most SOC applications are cost-sensitive, the issue width is limited

Table 5.4: Comparison between the Original Architecture and the Modified Architecture with a 64KB 2-Stage Pipelined Data Cache

	Original Core	Non-pipelined 64KB Cache
Die Size	$1.8mm^2$	$2.1mm^2$
Maximum Clock Frequency	267 MHz	267 MHz
IPC (compress)	0.753	0.794
Maximum Clock Frequency	267 MHz	267 MHz
Relative Performance	1.00	1.05
Bus Utilization	10.2%	6.8%

conservatively to only 2 instructions. Thus, we consider the design alternative of using a 2-way superscalar processor. Now, superscalar processors can more effectively hide the latency in the pipeline stages. Thus, it makes sense to use a larger, pipelined cache instead of a smaller, non-pipelined cache. Table 5.3 compares the performance of a 32KB data cache and a 64KB pipelined data cache in a 2-way superscalar processor. For the compress benchmark, the pipelined 64KB cache has a much lower bus utilization and is decidedly faster than the 32KB cache.

Table 5.5: Comparison between a 32KB non-pipelined data cache and a 64KB 2-stage pipelined data cache in a 2-way superscalar processor

	32KB Cache	64KB Cache
IPC (compress)	1.03	1.13
Maximum Clock Frequency	267 MHz	267 MHz
Relative Performance	1.00	1.10
Miss Rate	5.80%	3.38%
Bus Utilization	14.1%	9.8%

To avoid any performance bottleneck, duplicate functional units and a larger register file (with more read/write ports) are needed in the dual-pipeline. Figure 5.16 shows an estimated floorplan with $3.0mm^2$ area. In this case, the data cache is pipelined and the interconnections within the processor core are relatively short. As a result, the processor speed is estimated to be same as before. Table 5.3 summarizes the performance of the different design alternatives.

Table 5.6: Summary of Different Design Alternatives

	Original Core	Non-pipelined 64KB Cache	Pipelined 64KB Cache	Superscalar Core
Die Size	$1.8mm^2$	$2.1mm^2$	$2.1mm^2$	$3.0mm^2$
IPC	0.753	0.817	0.794	1.13
Clock Speed	267MHz	238MHz	267MHz	267MHz
Relative Performance	1.00	0.97	1.05	1.50

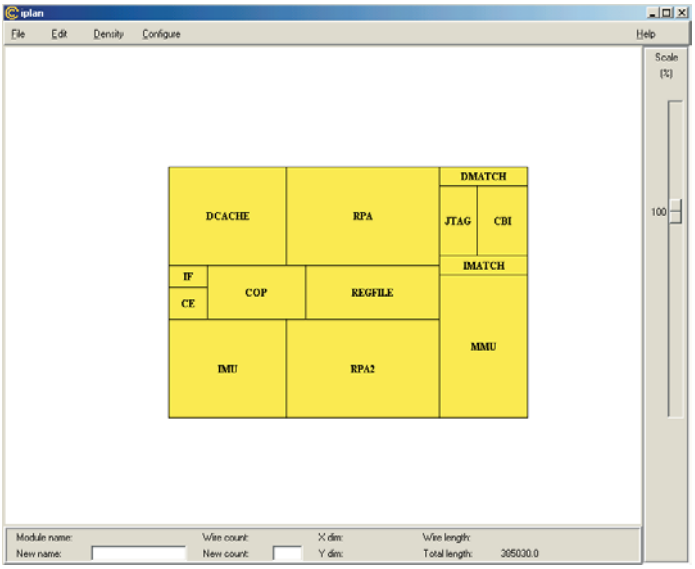


Figure 5.16: Optimized Floorplan with Dual Issue

The IPLAN floorplanning framework allows system-level designers to quickly evaluate the architecture and estimate its performance in the deep submicron era. As shown in this case study, designers can easily build many "prototypes" within a few hours and then examine the tradeoffs among these design alternatives. After the design is finalized, the floorplan can then be used to drive the physical layout. The simulation environment can also be used for system-level verification and validation.

5.4 Summary

The purpose of this chapter is to illustrate how the previous chapters come together in a unified framework. This interactive floorplanning framework, known as IPLAN, encompasses

four major components: the interconnect estimator, the MXS performance simulator, the area/delay estimator, and the floorplan optimizer.

The interconnect estimator is based on the wirelength distribution model presented in Chapter 2 and the wire congestion model presented in Chapter 3. The MXS simulator, which is an execution-based performance simulator, is used to model the microarchitecture in a superscalar processor. It can also be used in conjunction with the interconnect-driven processor performance model (Chapter 4) to determine the optimum instruction issue width in a processor. The area/delay estimator, which was developed based on the research work by Steve Fu and Grant McFarland, can be used to estimate the area and latency of a functional block. The results of these three estimators are sent to the floorplan optimizer, which determines an optimal floorplan based on the user's constraints and requirement. While most commercial floorplanning tools use a slicing floorplan structure to represent their floorplans, the IPLAN floorplanner uses the sequence-pair structure to represent non-slicing floorplans because a non-slicing floorplan are much more compact and efficient than a slicing floorplan. This is useful for area and wirelength optimization, and particularly important when the number of blocks are not too big (< 50).

In this chapter, the Lexra's LX4380 processor is used as an example to describe the different stages in the floorplanning process. It is also used to describe the different features in the IPLAN graphical user interfaces (GUIs). In Section 5.3, a design scenario is fabricated to illustrate how the different IPLAN components work together. In addition, this chapter also show how to develop quick prototypes and perform architectural tradeoffs in the IPLAN floorplanning framework.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Since the advent of the microprocessors, system-on-chip (SOC) is arguably the most important development in semiconductor industry in the past 20 years. According to Gartner Group, the market of SOC devices was about $20 billion in 2000 and is expected to grow to $60 billion in 2004 [Lew01]. By integrating hundreds of millions of transistors on a single die, complicated and power-efficient electronic systems (e.g. cellular phones) are affordable to the masses and become pervasive in every countries in the world.

While the SOC technology creates tremendous market opportunities in the semiconductor industry, it also presents many technical challenges for the existing CAD design tools. In particular, the traditional ASIC design methodology separates architectural design, logic design and physical design into three distinct phases, preventing efficient optimizations and effective design tradeoffs across the different design phases. This problem becomes increasingly more serious in SOC designs, where interconnect properties are the limiting factor in determining system performance, power consumption, signal integrity, reliability and manufacturing yield. This is one of the key reasons why many SOC's have problems surpassing 400MHz, when custom-made, high-end processors have already exceeded 2.5GHz.

Although interconnect-driven synthesis tools are gaining wider acceptance in recent years [Mag, Mon], interconnect-driven design methodologies and CAD tools are still missing in architectural design phase, often resulting in multiple design iterations and sub-optimal designs. Consequently, many design projects are unnecessarily canceled because these projects either cannot meet the system requirements (e.g. power and performance)

or take too long and miss the entire market windows.

The IPLAN floorplanning framework provides a new design methodology, allowing users to build rapid prototypes with floorplans and quickly evaluate their designs (e.g. in terms of cost, power and performance) in architectural design phase. This floorplanning framework, which encompasses three statistical interconnect models described in Chapters 2, 3 and 4 (wirelength distribution model, wire congestion model and interconnect-driven processor performance model), are useful to estimate the effects of interconnect in the early design stages. Our experimental results show that these interconnect models are more accurate and flexible than the previous models. Besides chip floorplanning, these models can be used in routing algorithms as well as yield estimation.

In addition to the interconnect estimator, the IPLAN floorplanning framework also consists of a performance estimator, an area/delay estimator and a floorplan optimizer. As described in Chapter 5, the results of the three estimators are provided to the floorplan optimizer, which determines an optimal floorplan based on the user's constraints and other specifications. In the IPLAN floorplanning framework, designers can easily modify design parameters (e.g. cache size, pipeline stages, issue width) and then quickly optimize the floorplans and examine the new system design costs and performances. This rapid prototyping approach minimizes design risks, avoid expensive redesigns and effectively shorten design schedules.

As feature size continues to shrink, "interconnect-driven architectures" are becoming a necessity. Our research concludes that "interconnect-driven architectural design tools" – such as the IPLAN floorplanning framework – are very useful and effective in SOC designs.

6.2 Contributions of this Dissertation

The main contribution of this work is the development of an interconnect-driven floorplanning framework (IPLAN), which allows chip designers to evaluate architectural tradeoffs in the early design stages. The other contributions of this work are summarized below:

- The development of a statistical a priori wirelength distribution model, which takes routing obstacles and the aspect ratios of the block and library cells into consideration. The wirelength distribution and the average wirelength estimation are more accurate than the previous models.

- The development of a statistical wire congestion model, which takes routing obstacles and via minimization into consideration. A fast algorithm was developed to efficiently calculate the routing density.
- The development of a floorplanning algorithm, using the sequence pair structure and based on the simulated annealing technique.
- The development of a processor performance model, which takes interconnect and complexity overheads into consideration. The development of a variety of interconnect and complexity overheads, including wire delay with repeaters, wire delay without repeaters and gate delays.
- The derivation of an upper bound on the optimum instruction issue width. This upper bound is quite tight and can therefore be used to estimate the optimum issue width.
- The graphical user interfaces for the IPLAN framework, which includes the floorplan placement user interface and the Motif interface for the MXS simulator.
- Other MXS enhancements, including porting the tool to Solaris and Linux and the support for instruction cache and second-level cache.

6.3 Future Work

The SOC design is a dynamic and changing field, consisting of many new technologies and unanswered questions. How do we utilize 10 billion transistors effectively? How do we verify such complicated SOC designs? How do we perform effective system design making trade-offs at all levels? How does the future on-chip interconnect system looks like? How do we partition design between software and hardware? How do we model and synthesize from complex behavioral models? It is virtually impossible to enumerate all the possible future directions regarding the IPLAN floorplanning framework. A short list of possible future work is listed below.

- Ideally the IPLAN floorplanning framework would allow users to build their chip automatically based on their specific design constraints and system requirements. This implies that the IPLAN framework needs to manage a database of pre-qualified intellectual properties (IP's). A logical extension of our research is to examine the

interconnect-driven IP interface issues (e.g. timing constraints, interconnect loading), ensuring that all reusable IP blocks (e.g. soft IP's, hard IP's, hard IP's) from different vendors are fully compatible [Vir].

- Another related issue is the SOC verification. This involves software, hardware and communication verifications [MS96]. Even for pre-qualified IP's, the interconnection timings and the communication protocols still need to be verified.

- In order to model a more realistic software workload, the MXS simulator needs to be extended to support different real-time operating systems (e.g. eCOS, VxWorks, LynxOS). On the hardware side, the MXS simulator can be integrated to the SystemC or System Verilog simulation environment to better model custom IP blocks.

- The current IPLAN floorplanning framework uses the MXS simulator, which models a dynamically-scheduled superscalar processor. In order to model multiple processors, we need integrate a multiple-processor simulator (e.g. ABSS simulator [Dwi98]) into the IPLAN floorplanning framework in the future.

- As power consumption becomes critical in many mobile applications, the IPLAN floorplanning framework can be improved by adding high-level power modeling, estimation and optimization. For instance, instruction-level power estimation [TMWL96] or similar techniques can be used in the performance simulator to estimate software power consumption.

Appendix A

Structural Wirelength Distribution Function

A.1 Introduction

The structural wirelength distribution function $s(l)$ is defined as the number of all possible connections between two gates in a structure. An elegant way to compute such an enumeration is proposed by Stroobandt [Str96], which is based on a representation of the wirelength distribution by its moment generating polynomial $G(x)$.

$$G(x) = \sum_{l=0}^{\infty} s(l)x^l \tag{A.1}$$

The moment generating polynomial contains all the information regarding the wirelength distribution, and it provides a simple way to compute the average wirelength as well as all higher order moments. Each moment may be calculated by differentiating $G(x)$ as many times as the order of the moment and evaluate the results for $x = 1$. For instance, the average value (first moment) of the structural wirelength distribution function is equal to $G'(1)$. An important property is that the moment generating polynomial of a convolution of two wirelength distributions is the product of the generating polynomials of each distribution. This property is very useful because the structural wirelength distribution function of a two-dimensional structure can then be calculated from one-dimensional structural wirelength distribution functions.

A.2 Building Generating Polynomials of One-Dimensional Structures

In this section, we calculate the moment generating polynomials of some simple one-dimensional structures. These one-dimensional moment generating polynomials are used to derive the structural wirelength distributions of the two-dimensional structures in the next section. The first case is to consider two chains of n nodes arranged in a one-dimensional Manhattan grid. The two chains are assumed to be different but they are connected directly at every single node (Figure A.1). The distance between P_i and P_j is $abs(i-j)$. By inspection, the structural length distribution function $s1(l)$ is as follows.

$$s_1(l) = \begin{cases} n & \text{if } l = 0 \\ 2(n-l) & \text{if } 1 \leq l \leq n \\ 0 & \text{otherwise} \end{cases} \tag{A.2}$$

The moment generating polynomial $G_1(x,n)$ is shown in Equation A.3.

$$G_1(x,n) = n + \sum_{l=1}^{n}(2(n-l)x^l) = \frac{2x^{n+1} - nx^2 - 2x + n}{(x-1)^2} \tag{A.3}$$

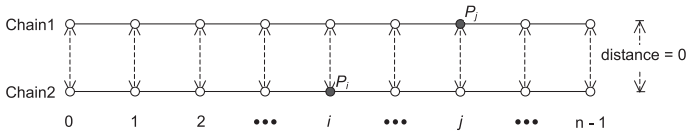


Figure A.1: Two chains of nodes connected at every node in a 1-D Manhattan grid

The second case is to consider two chains of nodes that are connected at one end (Figure A.2). One of the chains has m nodes whereas the other one has n nodes. In this diagram, the distance between P_i and P_j is equal to $i+j$. The moment generating function $G_2(x,m,n)$ is equal to the product of two zero-dimensional moment generating functions (Equation A.2)

$$G_2(x,m,n) = \sum_{l=0}^{m-1}\sum_{l=0}^{n-1} x^{l+k+1} = \frac{x(x^m-1)(x^n-1)}{(x-1)^2} \tag{A.4}$$

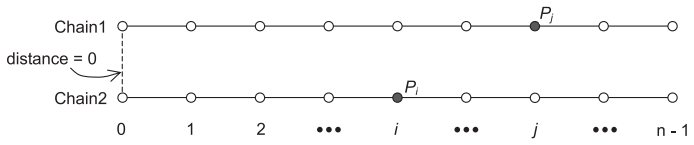


Figure A.2: Two chains of nodes connected at one end node in a 1-D Manhattan grid

The last case is to consider two chains of nodes that are connected at both ends (Figure A.3). In this diagram, each of the chains has n nodes and the distance between P_i and P_j is equal to $min(i+j, 2n-(i+j))$. The moment generating function $G_3(x,n)$ in this case may be derived by inspection (Equation A.5).

$$G_3(x,n) = nx^n + 2\sum_{l=1}^{n-1} l \cdot x^i = \frac{x(x^{n+1} - (n+1)x^n + 1)}{(x-1)^2} \tag{A.5}$$

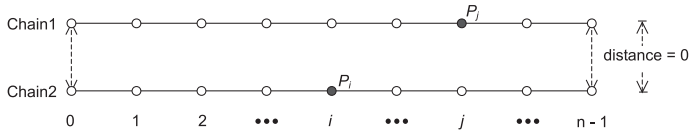


Figure A.3: Two chains of nodes connected at two end nodes in a 1-D Manhattan grid

A.3 Generating Polynomials for More Complicated Architectures

In this section, we use the one-dimensional moment generating polynomials calculated in the last section to derive the moment generating functions and the structural wirelength distribution functions of two-dimensional structures.

A.3.1 Rectangular Block

The first two-dimensional example is to consider a $m \times n$ rectangular block (Figure A.4). The Manhattan distance between any two points in the rectangular block is the sum of their horizontal distance and their vertical distance. Apparently, the structural wirelength distribution is the convolution of two one-dimensional distributions ($G_1(x,n)$ and $G_1(x,m)$). However, please note that this number of interconnections has to be divided by two because

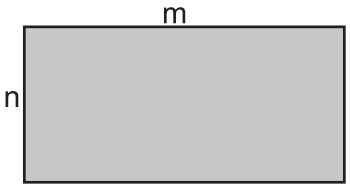


Figure A.4: $m \times n$ Rectangular Block

the two end points (P_i and P_j) are symmetrical so twice the number of interconnections have been counted. The moment generating function of a rectangular block $G_4(x, m, n)$ is derived in (A.6).

$$\begin{aligned} G_4(x,m,n) &= \frac{G_1(x,m)\cdot G_1(x,n)}{2} \\ &= \frac{(2x^{m+1} - mx^2 - 2x + m)(2x^{n+1} - nx^2 - 2x + n)}{2(x-1)^4} \\ &= \frac{2x^{m+n+2} - nx^{m+3} - 2x^{m+2} + nx^{m+1} - mx^{n+3} - 2x^{n+2} + mx^{n+1} + \ldots}{(x-1)^4} \end{aligned} \tag{A.6}$$

To obtain a more manageable closed-form expression, we define $f_l(n, i)$ as in Equation A.7.

$$f_l(n,i) = {}_{n-l-1}C_{i-1} = \frac{(n-l-1)!}{(i-1)!(n-l-i)!} \tag{A.7}$$

Using the definition in (A.7), $G_4(x, m, n)$ can be simplified as in (A.8). By comparing the coefficients in the moment generating function, the structural wirelength distribution function of a rectangular block can be found as listed in Table A.1.

$$\begin{aligned} G_4(x,m,n) = 2\sum_{l=0}^{m+n-2} f_l(m+n+2,4)x^l - n\sum_{l=0}^{m-1} f_l(m+3,4)x^l - \\ 2\sum_{l=0}^{m-2} f_l(m+2,4)x^l + n\sum_{l=0}^{m-3} f_l(m+1,4)x^l - m\sum_{l=0}^{n-1} f_l(n+3,4)x^l - \\ 2\sum_{l=0}^{n-2} f_l(n+2,4)x^l + m\sum_{l=0}^{n-3} f_l(n+1,4)x^l \end{aligned} \tag{A.8}$$

Table A.1: Structural Length Distribution Function for Rectangular Block

Range	$s_4(l)$
$1 \leq l \leq n$	$\frac{l^3}{3} - (m+n)l^2 + \frac{6mn-1}{3}l$
$n \leq l \leq m$	$-n^2 l + mn^2 + \frac{n(n^2-1)}{3}$
$m \leq l \leq m+n-2$	$\frac{((m+n-l)^2-1)(m+n-l)}{3}$

A.3.2 L-Shaped Block

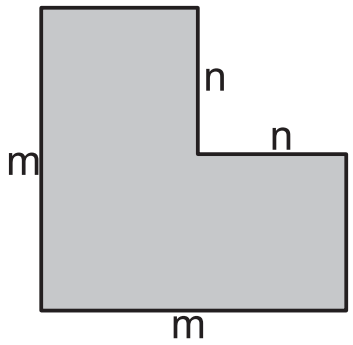


Figure A.5: L-Shaped Block

The second two-dimensional example is to consider an L-shaped block (Figure A.5). The moment generating function can be calculated by first considering is a $m \times m$ square block and then removing a $n \times n$ square block from its corner. By removing the $n \times n$ square block, we not only need to remove the interconnections within the $n \times n$ block but also remove the interconnections from the $n \times n$ block to the L-shaped block. The L-shaped block may be sub-divided into three rectangular blocks (an $n \times m-n$ block, an $m-n \times m-n$ block and an $m-n \times n$ block). Thus, we need to remove the interconnections from the $n \times n$ block to these three blocks (Equation A.9).

$$G_5(x,m,n) = \frac{G_1(x,m)^2}{2} - \frac{G_1(x,n)^2}{2} - 2G_1(x,n) \cdot G_2(x,n,m-n) - G_2(x,m-n,n)^2 \tag{A.9}$$

As before, Equation A.9 can further be simplified as in Equation A.10. By comparing

the coefficients in the moment generating polynomial, the structural distribution function of an L-shaped block can be found as listed in Table A.2.

$$
\begin{aligned}
G_5(x,m,n) = & \sum_{l=0}^{2m-2} f_l(2m+2,4)x^l - 2\sum_{l=0}^{m+n-2} f_l(m+n+2,4)x^l + \\
& 2\sum_{l=0}^{2m-n-2} f_l(2m-n+2,4)x^l + \sum_{l=0}^{2n-2} f_l(2n+2,4)x^l - 2(m-n)\sum_{l=0}^{m-1} f_l(m+3,4)x^l + \\
& 2(m-n)\sum_{l=0}^{m-3} f_l(m+1,4)x^l - \sum_{l=0}^{2m-2n-2} f_l(2m-2n+2,4)x^l - 2\sum_{l=0}^{n-2} f_l(n+2,4)x^l - \\
& 2n\sum_{l=0}^{m-n-1} f_l(m-n+3,4)x^l - 2\sum_{l=0}^{m-n-2} f_l(m-n+2,4)x^l + \\
& 2n\sum_{l=0}^{m-n-3} f_l(m-n+1,4)x^l
\end{aligned} \tag{A.10}
$$

Table A.2: Structural Length Distribution Function for L-Shaped Block

Range	$s_5(l)$
$1 \le l \le m-n$	$\frac{1}{2}l^3 - 2ml^2 + \frac{4m^2-4n^2-1}{2}l$
$m-n \le l \le n$	$\frac{1}{6}l^3 - (m-n)l^2 + \frac{6m^2-12mn+6n^2-1}{6}l + \frac{m^3-m+n+3m^2n-9mn^2+5n^3}{3}$
$n \le l \le 2m-2n$	$-\frac{1}{6}l^3 + (2n-m)l^2 + \frac{6m^2-12mn+1}{6}l + \frac{m^3-m+3m^2n-9mn^2+6n^3}{3}$
$2m-2n \le l \le m$	$-\frac{1}{3}l^3 + nl^2 + \frac{6mn-3m^2-6n^2+1}{3}l + \frac{5m^3-2m+n-9m^2n+3mn^2+2n^3}{3}$
$m \le l \le 2n$	$-\frac{1}{3}l^3 + (2m-n)l^2 + \frac{18mn-15m^2-6n^2+1}{3}l + \frac{11m^3-2m+n-15m^2n+3mn^2+2n^3}{3}$
$2n \le l \le 2m-n$	$-\frac{1}{6}l^3 + 2(m-n)l^2 + \frac{36mn-30m^2+1}{6}l + \frac{11m^3-2m+2n-15m^2n+3mn^2-2n^3}{3}$
$2m-n \le l \le m+n$	$\frac{1}{6}l^3 - nl^2 + \frac{12mn-6m^2+6n^2-1}{6}l + \frac{3m^3+n-3m^2n-3mn^2-n^3}{3}$
$m+n \le l \le 2m$	$\frac{((m+n-l)^2-1)(m+n-l)}{6}$

A.3.3 O-Shaped and C-Shaped Blocks

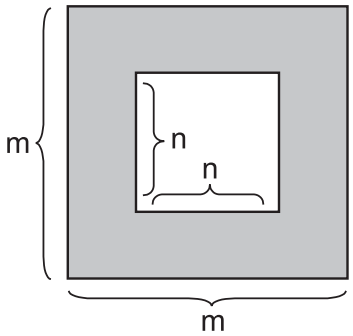


Figure A.6: O-Shaped Block

The next two-dimensional example is to consider the moment generating function of an O-shaped block (Figure A.6). In this diagram, the O-shaped block may be sub-divided into eight sub-blocks. Four of them are $\frac{m-n}{2} \times \frac{m-n}{2}$ square blocks and the remaining four are $\frac{m-n}{2} \times n$ rectangular blocks. Removing the middle $n \times n$ block from the $m \times m$ block eliminates not only the interconnections within the $n \times n$ block but also the interconnections from the $n \times n$ block to the O-shaped block. Unlike the previous two cases, an interesting side effect is that the all the wirelengths between the opposite $\frac{m-n}{2} \times n$ rectangular blocks are extended. This is because these interconnections cannot pass through the $n \times n$ obstacle directly. Equations A.11 shows the structural wirelength distribution function of an O-shaped block.

$$G_6(x,m,n) = \frac{G_1(x,m)^2}{2} - \frac{G_1(x,n)^2}{2} - 4G_1(x,n) \cdot G_2(x, \frac{m-n}{2}, \frac{m+n}{2}) - 4G_2(x,n,\frac{m-n}{2})^2 + 2G_2(x,\frac{m-n}{2},\frac{m-n}{2}) \cdot (G_3(x,n) - G_1(x,n)) \cdot x^{n+1} \quad \text{(A.11)}$$

The last example is to consider the moment generating function of a C-shaped block (Figure A.7). The C-shaped block may be sub-divided into three rectangular sub-blocks. Two of them are $\frac{m-n}{2} \times n$ blocks and the remaining one is an $m-n \times n$ block. Removing the $n \times n$ block from the $m \times m$ block eliminates not only the interconnections within the $n \times n$ block but also the interconnections from the $n \times n$ block to the C-shaped block. As in the O-shaped block, removing the $n \times n$ block also increases the wirelength between the two $\frac{m-n}{2} \times n$ blocks. The moment generating function of a C-shaped block $G_7(x,m,n)$ is

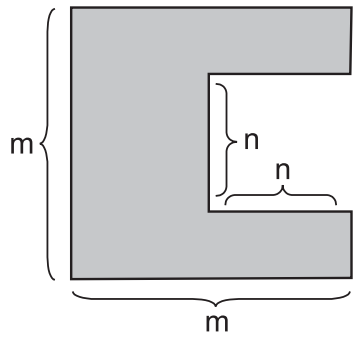


Figure A.7: C-Shaped Block

shown in Equation A.12.

$$G_7(x,m,n) = \frac{G_1(x,m)^2}{2} - \frac{G_1(x,n)^2}{2} - G_1(x,n)\cdot G_2(x,n,m-n) - 2G_2(x,n,m-n)\cdot G_2(x,n,\frac{m-n}{2}) - 2G_1(x,n)\cdot G_2(x,n,\frac{m-n}{2}) + G_2(x,n,n)\cdot G_2(x,\frac{m-n}{2},\frac{m-n}{2})\cdot x^{n+1} \quad \text{(A.12)}$$

A.3.4 Comparison of L-Shaped Block, C-Shaped Block and O-Shaped Block

Figure A.8 compares the structural wirelength distribution $s(l)$ of an L-shaped block, a C-shaped block and an O-shaped block for $M = 256$ and $N = 128$. Given the same number of gates, the L-shaped block has the shortest wirelength distribution whereas the O-shaped block has the longest wirelength distribution. Since the L-shaped has a more compact structure, it is usually better to place obstacles at the corner of a functional block.

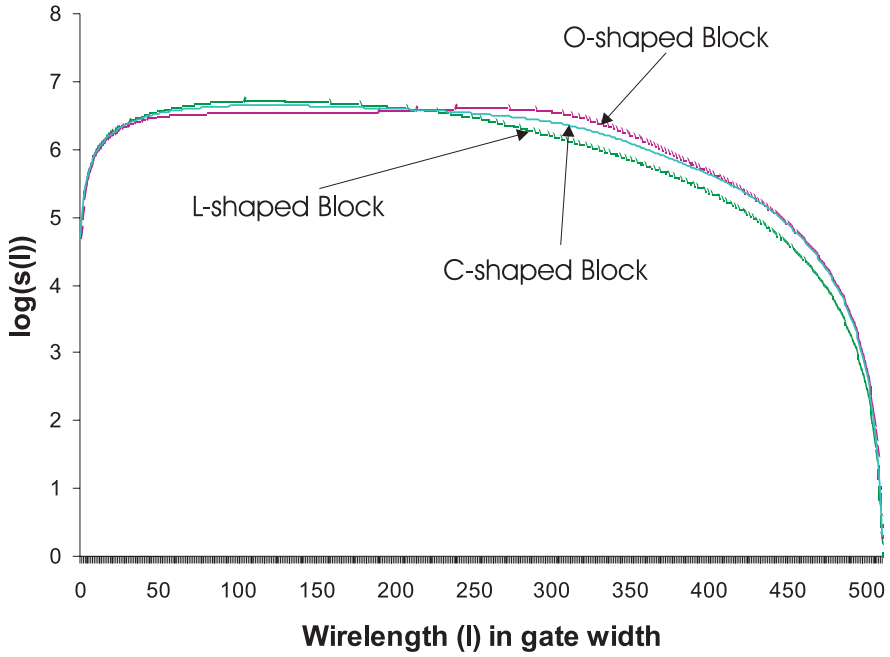


Figure A.8: Structural Wirelength Distributions of an L-shaped block, a C-shaped block and an O-shaped block (M = 256, N = 128).

Bibliography

[ARM] ARM Limited. ARM Architecture. In *http://www.arm.com/*.

[Ass01] Semiconductor Industry Association. The National Technology Roadmap for Semiconductors. San Jose, CA, 1997 and 2001.

[Bak90] H. Buman Bakoglu. *Circuit, Interconnections, and Packaging for VLSI*. Addison Wesley, 1990.

[BCR90] B. S. Baker, E. G. Coffman, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1990.

[Ben98] James E. Bennett. *Latency Tolerant Architectures*. PhD thesis, Stanford University, 1998.

[Ber97] Bertrand Meyer. *Object-oriented Software Construction*. Prentice-Hall, 1997.

[Bha94] Neal A. Bhadkamkar. *A Nonlinear Silicon Cochlea*. PhD thesis, Stanford University, 1994.

[Bla69] James R. Black. Electromigration – A brief survey on some recent results. *IEEE Transactions on Electron Devices*, ED-16:338, 1969.

[Bri01] Pierre Bricaud. Standards for SOC IP? In *Design Verification Exhibition*, November 2001.

[Cad] Cadence Design System. Virtual Component Co-Design (VCC) Overview. In *http://www.cadence.com/products/vcc.html*.

[CdG01] Phillip Christie and Jose Pineda de Gyvez. Pre-layout prediction of interconnect manufacturability. In *IEEE/ACM International Workshop on System Level Interconnect Prediction*, pages 167–173, 2001.

[CP96] J. M. Chang and M. Pedram. Energy minimization using multiple supply voltages. In *International Symposium on Low Power Electronics and Design*, pages 157–162, 1996.

[CSB92] Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low-Power CMOS Digital Design. *IEEE Journal of Solid State Circuits*, pages 473–484, 1992.

[DDM96] Jeffrey A. Davis, Vivek De, and James Meindl. Optimal Low Power Interconnect Networks. *Symposium on VLSI Technology Digest of Technical Papers*, pages 78–79, 1996.

[DDM98] Jeffrey A. Davis, Vivek K. De, and James D. Meindl. A Stochastic Wire-Length Distribution for Gigascale Integration (GSI) - Part 1: Derivation and Validation. *IEEE Transaction on Electron Devices*, 45:580–589, March 1998.

[DF90] Pradeep K. Dubey and Michael J. Flynn. Optimal Pipelining. *Journal of Parallel and Distributed Computing*, 8:10–19, 1990.

[Die00] Keith Diefendorff. Best New Technology: POWER4. *Microprocessor Report*, February 2000.

[Don79] Wilm E. Donath. Placement and Average Interconnection Lengths of Computer Logic. *IEEE Transaction on Circuits and Systems*, CAS-26(4):272–277, April 1979.

[Don81] Wilm E. Donath. Wire length distribution for placements of computer logic. *IBM Journal of Research and Development*, 25:152–155, 1981.

[Dwi98] Dwight Sunada and David Glasco and Michael Flynn. ABSS v2.0: a SPARC Simulator. Technical Report CSL-TR-98-755, Stanford University, 1998.

[ES00] Terry C. Edwards and Michael B. Steer. *Foundations of Interconnect and Microstrip Design*. Wiley, 2000.

[FCJV97] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic. The Multicluster Architecture: Reducing Cycle Time Through Partitioning. In *Annual International Symposium on Microarchitecture*, pages 149–159, 1997.

[Feu82] Michael Feuer. Connectivity of random logic. *IEEE Transactions on Computer*, C-31:29–33, 1982.

[FH01] Michael J. Flynn and Patrick Hung. CAD Tools for System-Level Modeling and Implementation. *Wiley Software Focus*, pages 134–139, 2001.

[FHP00] Michael J. Flynn, Patrick Hung, and Armita Peymandoust. Using Simple Tools to Evaluate Complex Architectural Tradeoffs. *IEEE Micro*, 20(4):67–75, July/August 2000.

[FHR99] Michael J. Flynn, Patrick Hung, and Kevin W. Rudd. Deep-Submicron Microprocessor Design Issues. *IEEE Micro*, 19(4):11–22, July/August 1999.

[Fis83] Joseph A. Fisher. Very long instruction word architectures and the ELI-512. In *Annual Symposium on Computer Architecture*, pages 140–50, 1983.

[FJC95] Keith I. Farkas, Norman P. Jouppi, and Paul Chow. Register File Design Considerations in Dynamically Scheduled Processors. *Western Research Laboratory Research Report 95/10*, pages 1–28, November 1995.

[Fly95] Michael J. Flynn. *Computer Architecture Pipelined and Parallel Processor Design*. Jones and Bartlett Publishers, 1995.

[Fu99] Steve Fu. *Cost Performance Optimization of Microprocessors*. PhD thesis, Stanford University, 1999.

[GA89] Carol V. Gura and Jacob A. Abraham. Average Interconnection Length and Interconnection Distribution Based on Rent's Rule. In *Design Automation Conference*, pages 574–577, 1989.

[Gla02] Peter N. Glaskowsky. Stretch Goals for Intel Servers. *Microprocessor Report*, March 2002.

[GM92] Cal T. Gabriel and Jim P. McVittie. How plasma etching damages thin gate oxides. *Solid State Technology*, 35:81–87, June 1992.

[Gom83] Hasaan Gomaa. The Impact of Rapid Prototyping on Specifying User Requirements. *ACM SIGSOFT Software Engineering Notes*, 2(8):17–28, 1983.

[HC01] Charlie Hauck and Charlie Cheng. VLIS Implementation of a Portable 266MHz 32-Bit RISC Core. *Microprocessor Report*, October 2001.

[Hen99] Henry Chang and Larry Cooke and Merrill Hunt and Grant Martin and Andrew McNelly and Lee Todd. *Surviving the SOC Revolution.* Kluwer Academic Publishers, 1999.

[HF97] Patrick Hung and Michael J. Flynn. Stochastic Congestion Model for VLSI Systems. Technical Report CSL-TR-97-737, Stanford University, October 1997.

[HF99] Patrick Hung and Michael J. Flynn. Deep Submicron VLSI Floorplanning Algorithm. In *1999 Electronic Devices and Systems Conference*, pages 84–87, 1999.

[HF00] Patrick Hung and Michael J. Flynn. Designing Future Microprocessors. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2000.

[HS90] Dwight Hill and Don Shugard. Global Routing Considerations in a Cell Synthesis System. In *Proc. of 27th Design Automation Conference*, pages 312–316, 1990.

[HSF01] Patrick Hung, Luc Sémeria, and Michael Flynn. Effects of Aspect Ratio and Routing Obstacles on Interconnect Wirelength. In *IEEE ASIC/SOC Conference*, 2001.

[Hsu83] C.P. Hsu. Minimum-via topological routing. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, CAD-2(4):235–246, 1983.

[Isr00] Israel Koren and Zahava Koren. Incorporating Yield Enhancement into the Floorplanning Process. *IEEE Transactions on Computers*, 49:532–541, 2000.

[Joh91] Mike Johnson. *Superscalar Microprocessor Design.* Prentice Hall, 1991.

[JW89] N. P. Jouppi and D. W. Wall. Available instruction-level parallelism for superscalar and superpipelined machines. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 290–302, 1989.

[Kaj83] Y. Kajitani. Order of Channels for Safe Routing and Optimal Compaction of Routing Area. *IEEE Transaction on Computer-Aided Design*, CAD-2(4):293–300, 1983.

[Kel96] Jim Keller. The 21264: A superscalar alpha processor with out-of-order execution. In *Annual Microprocessor Forum*, October 1996.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[Kre01a] Kevin Krewell. Alpha Quietly Reaches 1GHz. *Microprocessor Report*, August 2001.

[Kre01b] Kevin Krewell. Athlon XP Eschews GHz. *Microprocessor Report*, November 2001.

[Leb83] André Leblond. CAF: A Computer-Assisted Floorplanning Tool. In *Proc. of 20th Design Automation Conference*, pages 747–753, 1983.

[Lew01] Bryan Lewis. Semiconductor Intellectual Property: The Key Ingredient for SLI/SOC. *Gartner Group Research Brief*, March 2001.

[LFK+93] P. Geoffrey Lowney, Stefan Freudenberger, Thomas Karzes, W.D. Lichtenstein, Robert P. Nix, John S. O'Donnell, and John C. Ruttenberg. The Multiflow Trace Scheduling Compiler. *Journal Of Supercomputing*, 7(1-2):51–142, May 1993.

[LR71] Bernard S. Landman and Roy L. Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *IEEE Transaction on Computers*, C-20(12):1469–1479, December 1971.

[LSW94] K. F. Liao, M. Sarrafzadeh, and C. K. Wong. Single-Layer Global Routing. *IEEE Transaction on Computer-Aided Design/ICS*, 13(1):38–47, 1994.

[Mag] Magma Design Automation, Inc. BlastFusion and BlastChip Systems Overview. In *http://www.magma-da.com/*.

[Mag98a] Maggie Kang and Wayne W. M. Dai. Arbitrary Rectilinear Block Packing Based on Sequence Pair. In *Proc. of International Conference on Computer Aided Design*, pages 259–266, 1998.

[Mag98b] Maggie Kang and Wayne W. M. Dai. Topology Constrained Rectilinear Block Packing for Layout Reuse. In *Proc. of International Symposium of Physical Design*, pages 179–186, 1998.

[McF97] Grant W. McFarland. *CMOS Technology Scaling and its impact on Cache Delay.* PhD thesis, Stanford University, 1997.

[MFNK95] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-Packing-Based Module Placement. In *Proc. of International Conference on Computer Aided Design*, pages 472–479, 1995.

[Mic02] *Microprocessor Report*, Various issues, 1994-2002.

[MNK95] H. Matsuda, S. Nakatake, and Y. Kajitani. Optimum Slicing Structure Floorplanning with Routing Area Included. *IEICE Technical Report VLD94-109*, 94(531):9–14, 1995.

[Mon] Monterey Design System. System-Driven Physical Design. In *http://www.montereydesign.com/.*

[MPR02] MPR's Analysts' Choice Award Winners. *Microprocessor Report*, April 2002.

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. 21:1087–1092, 1953.

[MS96] Giovanni De Micheli and Mariagiovanna Sami. *Hardware/Software Co-Design.* Kluwer Academic Publisher, 1996.

[MY87] Akira Masaki and Minoru Yamada. Equations for Estimating Wire Length in Various Types of 2-D and 3-D System Packaging Structures. *IEEE Transaction on Components, Hybrids, and Manufacturing Technology*, CHMT-10(2):190–198, June 1987.

[Naf99] Samuel Naffziger. Design Methodologies for Interconnect in GHz+ ICs. In *ISSCC Tutorial*, 1999.

[Nai87] Ravi Nair. A Simple Yet Effective Technique for Global Wiring. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, CAD-6:165–172, March 1987.

[NFMK96] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module Placement on BSG-Structure and IC Layout Applications. In *Proc. of International Conference on Computer Aided Design*, pages 484–491, 1996.

[NMN87] N. J. Naclerio, S. Masuda, and K. Nakajima. Via minimization for gridless layouts. In *Proc. of 24th Design Automation Conference*, pages 159–165, 1987.

[Ott82] Ralph H.J.M. Otten. Automatic floorplan design. In *Proc. of 19th Design Automation Conference*, pages 261–267, 1982.

[PBS98] Phiroze N. Parakh, Richard B. Brown, and Karem A. Sakallah. Congestion Driven Quadratic Placement. In *Proc. of 35th Design Automation Conference*, pages 275–278, 1998.

[Phi97] Philip G. Emma. Understanding some simple processor performance limits. *IBM Journal of Research and Development*, 41(3):215–232, May 1997.

[PJS96] Subbarao Palacharla, Norman P. Jouppi, and James E. Smith. Quantifying the Complexity of Superscalar Processors. Technical Report CS-TR-96-1328, University of Wisconsin-Madison, November 1996.

[Pol] Dale Pollek. Deep submicron design is more than just "noise". In *EEdesign. June 9, 2002.*

[PP89] M. Pedram and B. Preas. Interconnection Length Estimation for Optimized Standard Cell Layouts. In *International Conference on Computer Design*, pages 390–393, 1989.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[RHWG95] Mendel Rosenblum, Stephen Herrod, Emmett Witchel, and Anoop Gupta. Complete Computer System Simulation: the SIMOS Approach. *IEEE Parallel & Distributed Technology: Systems & Applications*, 3:34–43, 1995.

[RKN89] Chong S. Rim, Toshinobu Kashiwabara, and Kazuo Nakajima. Exact Algorithms for Multilayer Topological Via Minimization. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 8(4):1165–1184, 1989.

[Ron01] Ron Ho and Kenneth W. Mai and Mark A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[Roy70] Winston W. Royce. Managing Development of Large Scale Software Systems: Concepts and Techniques. In *IEEE WESCON*, volume 14, pages A/1–1–A/1–9, August 1970.

[RS95] Salil Raje and Majid Sarrafzadeh. Variable Voltage Scheduling. In *International Symposium on Low Power Design*, pages 9–14, 1995.

[Rud99] Kevin Rudd. *VLIW Processors: Efficiently Exploiting Instruction-level Parallel Processors.* PhD thesis, Stanford University, 1999.

[SBV95] Gurindar S. Sohi, Scott E. Breach, and T.N. Vijaykumar. Multiscalar processors. In *22nd International Symposium on Computer Architechture*, pages 414–425, 1995.

[SFK97] Dezso Sima, Terence Fountain, and Peter Kacsuk. *Advanced Computer Architectures: A Design Space Approach.* Addison-Wesley, 1997.

[She95] Naveed Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, 1995.

[SNK98] Keishi Sakanushi, Shigetoshi Nakatake, and Yoji Kajitani. The Multi-BSG: Stochastic Approach to an Optimum Packing of Convex-Rectilinear Blocks. In *Proc. of International Conference on Computer Aided Design*, pages 267–274, 1998.

[Sny01] Cary D. Snyder. Synthesizable Core Makeover. *Microprocessor Report*, July 2001.

[SO72] I.E. Sutherland and D. Oestreicher. How big should a printed circuit board be? *IEEE Transactions on Computers*, C-22:537–542, 1972.

[SP86] Sarma Sastry and Alice C. Parker. Stochastic Models for Wireability Analysis of Gate Arrays. *IEEE Transaction on Computer-Aided Design*, CAD-5(1):52–65, January 1986.

[ST83] T. Sakurai and K. Tamaru. Simple Formulas for Two- and Three- Dimensional Capacitances. *IEEE Transaction on Electron Devices*, ED-30(2):183–185, February 1983.

[Sto83] L. Stockmeyer. Optimal Orientations of Cells in Slicing Floorplan Designs. *Information and Control*, 59:91–101, 1983.

[Str96] Dirk Stroobandt. Improving Donath's technique for estimating the average interconnection length in computer logic. Technical Report DG 96-01, ELIS, June 1996.

[Str99] Dirk Stroobandt. Tutorial on System-Level Interconnect Prediction. In *Workshop on System-Level Interconnect Prediction*, 1999.

[Sys] SystemC Consortium. In *http://www.systemc.org/*.

[Tai] Taiwan Semiconductor Manufacturing Company. In *http://www.tsmc.com/*.

[TEE+96] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreaded Processor. In *23rd International Symposium on Computer Architechture*, pages 191–202, May 1996.

[Tex] Texas Instruments. Open Multimedia Applications Platform (OMAP) Overview. In *http://www.ti.com/sc/omap/*.

[TMWL96] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction Level Power Analysis and Optimization of Software. *Journal of VLSI Signal Processing*, 13(2/3), 1996.

[UM97] R. Uhlig and T. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, 29:128–170, June 1997.

[Uni] United Microelectronics Corporation. In *http://www.umc.com/*.

[Vir] Virtual Socket Interface Alliance. Virtual socket interface architecture document. In *http://www.vsi.org/library/vsi-or.pdf*.

[WFW+94] R. P. Wilson, R. S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, M. W. Hall, M. S. Lam, and J. L. Hennessy. SUIF: An infrastructure for research on parallelizing and optimizing compilers. In *SIGPLAN*, volume 29, pages 31–37, December 1994.

[XC98] J. Xu and C. Kuan Cheng. Rectilinear Block Placement Using Permutation-pair. In *Proc. of International Symposium of Physical Design*, pages 173–178, 1998.