國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

考慮處理器微架構之效能最佳化布局技術

Microarchitecture-Aware Floorplanning for Processor Performance Optimization

研 究 生：陳紀穎

指導教授：陳宏明 博士

黃俊達 博士

中華民國九十五年八月

考慮處理器微架構之效能最佳化布局技術

Microarchitecture-Aware Floorplanning for Processor Performance Optimization

研 究 生：陳紀穎 Student: Chi-Ying Chen

指導教授：陳宏明 博士 Advisor: Prof. Hung-Ming Chen

黃俊達 博士 Prof. Juinn-Dar Huang

國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩士論文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of Requirements
for the Degree of
Master of Science
in
Electronics Engineering
August 2006
Hsinchu, Taiwan, Republic of China

中華民國九十五年八月

考慮處理器微架構之
效能最佳化布局技術

研究生：陳紀穎　　　　　　　指導教授：陳宏明　博士

黃俊達　博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

過去的晶片布局軟體所使用的目標方程式主要著重於縮減線長與面積。由於過去晶片內部連線的延遲所需要的時間是可以忽略的，這樣的方程式被認為是足夠的。然而，隨著半導體的製程不斷進步，晶片內部連線所需要的時間如今已經不能忽略不計。這些多餘的延遲會影響到處理器的效能，但布局軟體並未考慮這些延遲。

我們提出一個方法，將微處理器架構的性能與晶片布局兩者之間的關係連結起來，以達成針對處理器效能最佳化的考慮微處理器架構的晶片布局。實驗結果顯示我們的方法的確改進性能。

Microarchitecture-Aware Floorplanning for Processor Performance Optimization

Student: Chi-Ying Chen

Advisor: Prof. Hung-Ming Chen
Prof. Juinn-Dar Huang

Department of Electronics Engineering &
Institute of Electronics,
National Chiao Tung University

ABSTRACT

In the past, floorplanner used objective functions focused on reducing wire length and area. These objective functions were considered efficient before since the latencies of interconnects were within single clock cycle or even could be neglected. However, as semiconductor technology advances, feature size continues to shrink. The communication of signals on interconnects becomes multi-cycle, therefore the latencies can not be ignored now. These latencies have impact on the performance, and most of current floorplanning frameworks do not consider these issues.

We proposed a methodology based on a heuristic for better performance in terms of microarchitecture and floorplanning achieving microarchitecture-aware floorplanning for processor performance optimization. The result from experiments shows the validity of our methodology.

誌　　謝

從七年前進入交大開始到今天，在即將得到碩士學位、結束學生生涯的此時，有許多需要感謝的人。

首先是親愛的父母。感謝您們養育我，給我一個幸福美滿的家庭，讓我接受良好的教育，也給我的經濟提供後盾，讓我衣食無缺。言語難以訴說我對您們的感謝與愛。謝謝。

感謝陳宏明老師，收留了當初只有備取的我，讓我有機會接觸到 EDA 這個面向。謝謝老師親切的指導，更要感謝老師讓我研究我有興趣的題目。同時也要感謝黃俊達老師願意撥出額外的時間與精力來指導我。感謝兩位老師的教誨。另外，我想特別在此感謝王聖智老師，謝謝您在我大學成績最差的時刻關心我，您的指導我一直銘記在心。

感謝實驗室的同學們，無論是陳老師的學生、或是黃老師的學生。兩年的相處是難忘的愉快回憶，很高興學生生涯的最後兩年時光能跟大家共度。還有許多其他的朋友：電工 92 級的同學們、台北的朋友們、復興堂的朋友們，若要一個個的感謝，時間便不夠了，容我在此簡短的向你們說聲謝謝。

最後，謝謝您，親愛的上帝，全能的父、創造天地的主。感謝您所賜予孩子的這一切。

Microarchitecture-Aware Floorplanning for Processor Performance Optimization

Chi-Ying Chen

Directed by Prof. Hung-Ming Chen
and Prof. Juinn-Dar Huang

In Partial Fulfillment of the Requirements

for the Degree of
Master of Science

Department of Electronics Engineering

National Chiao Tung University

Hsinchu, Taiwan 300, R.O.C.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cycle time is very important in integrated circuits since it determines how fast a chip can run. The cycle time of a chip is determined by its delay usually, and the delay of a chip can be calculated simply by adding all delays of functional units along the critical path in the past. This method is based on the assumption that the delays of functional units are large enough that they mask the effects of delays of interconnection. In other words, the main contributor of delay is functional units, and delays of interconnection doe not contribute any delay. From this point of view, we can safely ignore the delays of interconnection while calculating the delay of a chip.

The validity of above assumption and corresponding methodology lasted until recently. As technology advances, there is another different story about delay. To be more specific, the feature size of transistors shrinks as technology advances, therefore the delays of functional units are scaled as well. It is not the case when it comes to interconnection though. Unfortunately, delays of interconnection are not scaled when feature size shrinks. When feature size continues to shrink, problem arises: the delays of interconnection could not be ignored anymore. Therefore, the assumption for methodology we used to calculate delay before is no longer valid.

What makes things worse is that the clock frequency of a chip is getting higher and higher. Higher clock frequency means shorter clock cycle time, and shorter clock cycle time means it takes more cycle for interconnection delay. All these eventually mean that the communication between functional blocks in a chip/IP is not single cycle anymore. There is an industrial example: Pentium 4 of Intel. As a processor works in high frequency, it deploys two pipeline stages only for signal transmission[5]. Without these empty pipeline stages, the signals may not arrive their destination in time, which is a result of long interconnection delay and high clock frequency.

Above issues about latencies of interconnects happened in physical design stage were considered unimportant before since latencies of interconnects were ignored. Now we need to take these issues into account.

1.1 Motivation

Here we utilize a simplified conceptual classical 5-stage pipelined microarchitecture for our example to illustrate our motivation. This conceptual microarchitecture consists of nine blocks. These blocks are: block for instruction fetch (IF); instruction decoding (ID); execution stage(EX); memory reference, write back (MW); 4 functional blocks for execution stage: ALU for arithmetic and logical instructions, EIM for conditional move instructions, SS for swap instructions, and SR for shift instructions; and register file (RF). We show this simplified conceptual microarchitecture and the relation of its interconnections in Figure 1.1.

Let us assume that there is no interconnection delay. If each functional unit/stage takes one clock cycle to complete, we'd like to know how much time does an instruction consume. Take ALU-type instruction as example, we can see from the figure that an ALU-type instruction "goes through" a path which contains the following blocks: IF, ID, register file (RF), EX, ALU, and memory reference and write back. This path has 6 blocks in total; therefore an ALU-type instruction takes six clock

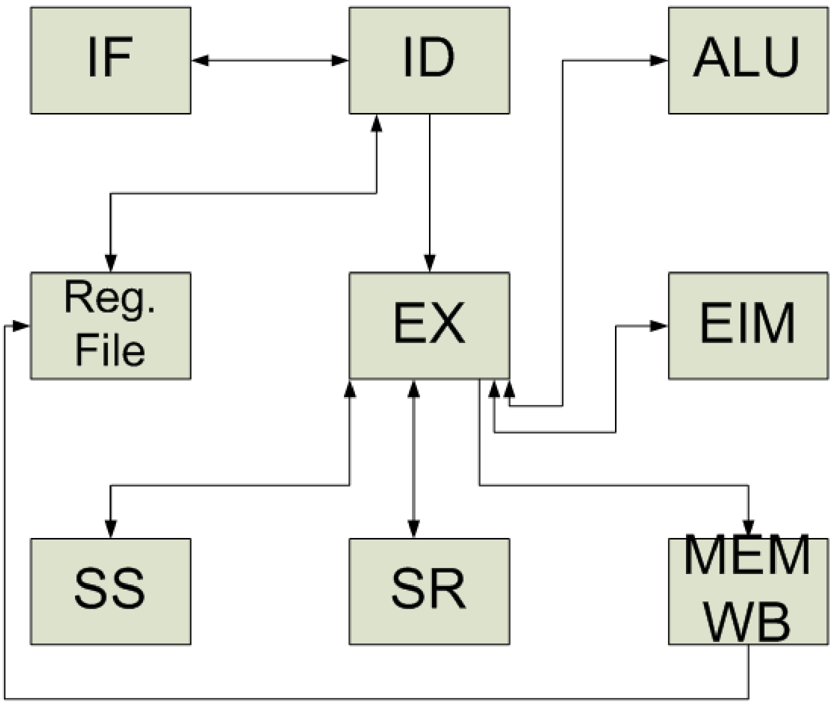

Figure 1.1: A conceptual microarchitecture.

cycles if there is no interconnection delay, since each block consumes one clock cycle. This can be applied for other three functional blocks as well.

Next, let us take interconnects into account. Figure 1.2 shows Figure 1.1 with interconnections labeled according to their connection. From Figure 1.2, we can observe that except from the original delay of functional blocks, which is six clock cycles, we need to consider more factors, and these factors will vary with different kinds of instructions.

Let us again use ALU-type instruction as an example. Starting from IF, an ALU-type instruction will go through the following interconnections, labeled as IF-ID, ID-RF (twice), ID-EX, EX-ALU (twice), EX-MW, and MW-RF (for write back).

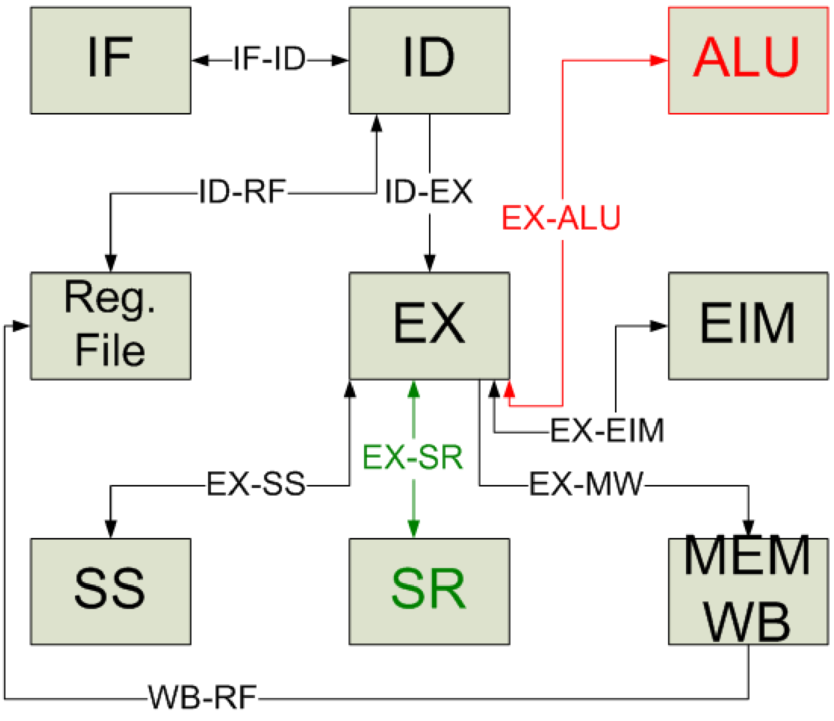

Figure 1.2: A conceptual microarchitecture with interconnections labeled.

Therefore, the time which an ALU-type instruction needs is shown in Equation 1.1:

$$Cycle_{ALU} = 6 + (IFID + IDRF * 2 + IDEX + EXALU * 2 + EXMW + MWRF) \tag{1.1}$$

Consider another kind of instruction, Shift instructions. Shift instructions use functional unit SR. Starting from IF, an SR-type instruction goes through IF-ID, ID-RF (twice), ID-EX, EX-SR (twice), EX-MW, and MW-RF (for write back). We show the time which a Shift-type instruction consumes in Equation 1.2:

$$Cycle_{SR} = 6 + (IFID + IDRF * 2 + IDEX + EXSR * 2 + EXMW + MWRF) \tag{1.2}$$

From above equations Eq. 1.1 and Eq. 1.2, we have shown that the cycle time needed is no longer the same while considering interconnection delay. Consider another example shown in Figure 1.3, which is almost the same as Figure 1.2 except that the functional units ALU and SR are swapped. This change is possible in

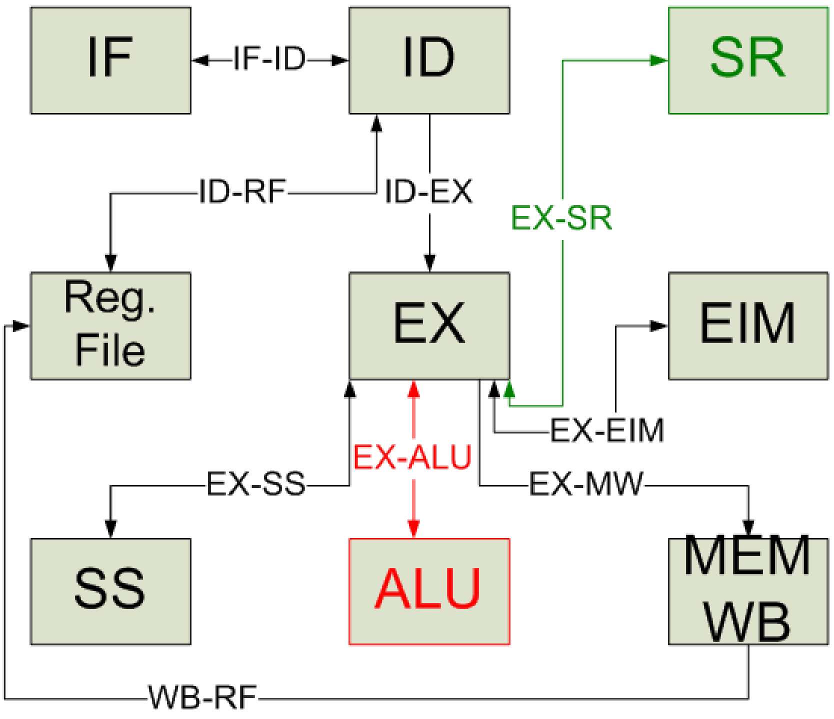

Figure 1.3: A conceptual microarchitecture with interconnections labeled. Note that ALU and SR is swapped.

floorplanning. We are interested in the effect brought by this change and it will be discussed in later chapter.

As we have seen, we wish to consider the corresponding issues in floorplanning, since we have seen that in floorplanning the relative location of blocks will change, hence the interconnection is also changed. The main considerations, in other words, objective functions of floorplanning, were wire length, area, aspect ratio, etc, which is inadequate now. Some previous [1] [2] [3] [4] works had been done in order to improve this. Later we will show our methodology to improve the conventional objective function in floorplanning.

1.2 Our Contribution

We try to integrate some performance measurement into the objective function of floorplanning. For achieving microarchitecture-aware floorplanning for processor performance optimization we propose a methodology in floorplanning to relate performance and floorplanning by using lengths of interconnects as a metric of floorplanning. Correspondingly performance modeling and objective function are shown later.

Compared with the previous work [1], which uses simulation-based methodology, our methodology only need simulation once or very limited times to obtain the instruction mix for objective function weighting. About previous works [2] and [4], both uses interconnect as metric but differ in the formulation. [2] uses a linear programming formulation to relate factors in the objective function of floorplanning which may consume a lot of time while objective function of our methodology is simple and fast. [4] uses design of experiments to reduce simulations needed for extracting characteristics of design, which we only need simulation once to get the instruction mix. Another previous work [3] uses a table-based methodology. Our methodology is directly related with the physical factors rather than mathematical techniques for modeling CPI (clock cycle per instruction)[6] to constructing a table.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. In Chapter 2, we will review some previous works done for this topic. In Chapter 3, we will first explain the performance modeling for validation, then propose our methodology. Then in Chapter 4, we will show the setup and results of experiments. Finally, Chapter 5 presents the conclusion and future works.

Chapter 2

Preliminaries

In this chapter we will introduce some background knowledge and present four previous works which had been done about floorplanning and performance in terms of microarchitecture/processor. These works are listed in the order of the year published. Finally we will state our problem formulation.

It is intuitive that different CPI means different performance. To clarify this, we start from the Equation 2.1 [6] frequently used to describe performance.

$$CPU\ time = Instruction\ Count * CPI * Clock\ cycle\ time \tag{2.1}$$

We can use this equation to formulate the performance of a given mircoarchitecture, since the CPU time can be used as a metric to measure performance. It is intuitive that the more CPU time used means the worse performance. Because of the execution time is proportional to the CPU time, and the performance is inverse proportional to the execution time, we can use the CPU time as a metric of performance.

Consider the three variables in Equation 2.1: Instruction Count, CPI, and Clock cycle time. In order to simplify the equation we assume that the clock cycle time is a constant first. In other words, the clock rate is a constant. This assumption reduces the number of variables to two. To reduce the number of variables further, consider the fact that for a given microarchitecture, we can treat the instruction count as a constant since the instruction set is microarchitecture-dependent, and the instruction count of a program is determined by the microarchitecture of its platform. Different microarchitecture will have different kinds of instructions; therefore the instruction count used in different microarchitecture will be different, even if they do the same thing in the final results. Therefore the variable IC can also be ignored, and the equation has only one variable now.

The variable remained at the right side of the equation is now CPI. It is directly proportional to the CPU time, which is at the left side of the equation. Also, we have learned that the CPU time is relative to the performance. Therefore, we can say that CPI is relative to the performance, in inverse proportional relationship. In other words, the lower CPI means the better performance. In fact, it is very general to use CPI as metric of performance and it is also used widely in the previous works we are going to introduce.

2.1 Microarchitecture Evaluation with Physical Planning [1]

The authors in [1] claimed the importance of considering the interaction between architectural design and physical design. For this reason, the authors proposed MEVA (Microarchitectural EVAluation), which they claimed to provide a set of flexible and customizable tools to explore an architectural design space with both accurate physical planning information and cycle-accurate architectural simulation. Figure 2.1 shows the overview of the MEVA system.

The MEVA system takes two inputs - (a) an architectural template, which is a block-level netlist that captures connectivity of the major functional blocks and (b) a library of architectural alternatives for the different blocks in the template. These

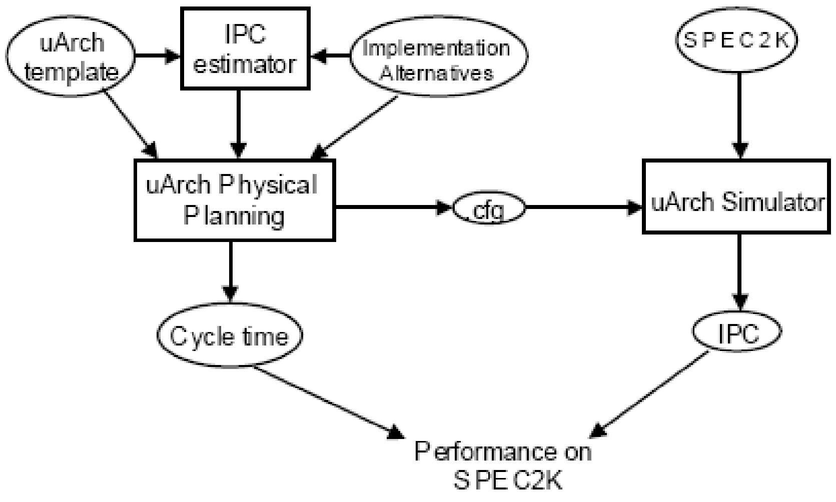

Figure 2.1: Overview of the MEVA system[1].

two inputs together provide space exploration.

The MEVA system consists of three main components: (a) physical planning engine, (b) IPC estimator and (c) cycle-accurate microarchitectural simulator. The physical planning engine will explore the design space by using different architectural alternatives in floorplanning. Its objective function is shown as below:

$$\frac{\sum_{i=1}^{n} w(i)wl(i)}{IPC(c)} \tag{2.2}$$

where *w(i)* is the weight of a net *i*, *wl(i)* is the wire length of *i*, and *IPC(c)* is the IPC of the current configuration *c*. The IPC estimator is to provide the planning engine with a quick and accurate IPC estimation for any configuration of block alternatives chosen by the planning engine at anytime in the process of floorplanning. However, in this work, the IPC is not obtained by estimation but by cycle accurate simulation in SimpleScalar 2.0 toolset[7], which apparently consumes a lot of time. Finally, the cycle-accurate microarchitectural simulator is used to verify the performance MEVA produced.

The authors claimed that the experimental results emphasize the importance of taking interconnect effects into account when exploring an architectural design space.

2.2 Profile-Guided Microarchitectural Floorplanning for Deep Submicron Processor Design [2]

This paper proposed a profile-guided microarchitectural floorplanner that considers both the impact of wire delay and the architectural behavior, namely the inter-module communication, to reduce the latency of frequent routes inside a processor and to maintain performance scalability, by using the profile information acquired at the architecture/microarchitecture level.

The flow chart of profile-guided floorplanning is shown in Figure 2.2. A machine description is provided as the input for the microarchitecture simulator SimpleScalar 2.0[7] for profiling module-to-module communication. Then a cycle-accurate simulation is performed to collect and extract the amount of interconnection traffic between modules for given benchmark programs. After the timing and area information of each module is collected, the statistical interconnection traffic and the processor target frequency range are fed into the profile-guided floorplanner to generate a floorplan for deriving the timing model (i.e. inter-module latency) of the microarchitecture as a result of the generated floorplan. With the new latencies, the architecture performance simulation is performed for simulating the IPC numbers, and later the actual performance in BIPS can be calculated together with the cycle time. This process described above is iterated, for exploring alternate architecture designs. Results from the profile-guided microarchitectural floorplanner can be used to guide the final global floorplanning such that the performance is not degraded.

The mathematical programming models for floorplanner is firstly in the form of mixed integer nonlinear programming (MINP) which minimizes the weighted wire

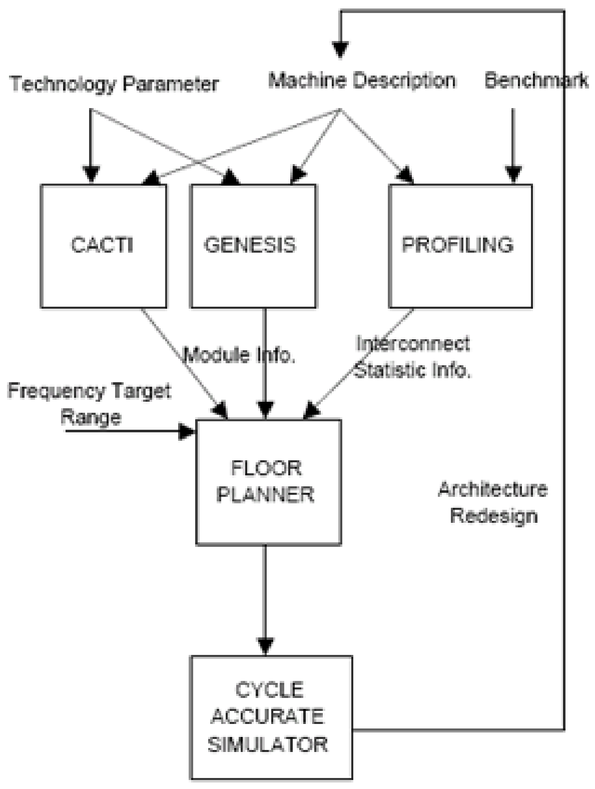

Figure 2.2: Overview of the profile-guided floorplanning[2]. Machine Description is used for inter-module profiling. The resultant interconnect statistic information is then used in floorplanner. Resultant floorplan is verified by cycle-accurate simulation.

length. It is then linearize and relax the integral constraints, since to find the optimal solution of MINP model is NP-hard.

Unlike [1], this work [2] does not use simulation-based methodology to obtain performance results. Rather, it uses a heuristic which try to estimate relationship between inter-module traffic and performance. The relationship is then modeled by mixed integer nonlinear programming, further simplified, and hence can be used in floorplanning. Compared with simulation-based methodology, this estimation-based methodology can reduce running time since cycle-accurate simulation is time-consuming. However, its mathematical modeling is still complex.

2.3 Floorplanning Optimization with Trajectory Piecewise-Linear Model for Pipelined Interconnects [3]

In this work [3] a table-based model called trajectory piece-wise linear (TPWL) model is developed to estimate CPI with interconnect pipelining for floorplanning optimization in order to minimize CPI. The authors claimed that TPWL model differs from cycle-accurate simulations by less than 3.0%. They also claimed that the results of their CPI-aware floorplanning is better than conventional floorplanning in CPI by up to 28.6% with area overhead of 5.69% under 100nm technology.

A bus latency vector ($\overrightarrow{B}$) is defined for a floorplan as a vector containing the latency of each interconnect in target design. For a given floorplan, if Bus 1 has a latency of 3, Bus 2 has a latency of 4, Bus 3 has a latency of 7, and so on. The $\overrightarrow{B}$ for that floorplan would be $\overrightarrow{B}$ = 3, 4, 7, ... The latencies of the interconnects are computed according to the Manhattan distance between the centers of two blocks in the floorplan by dividing the length of each interconnect by the flip-flop insertion length. For a given $\overrightarrow{B}$ cycle-accurate simulation in SimpleScalar 2.0 framework[7]

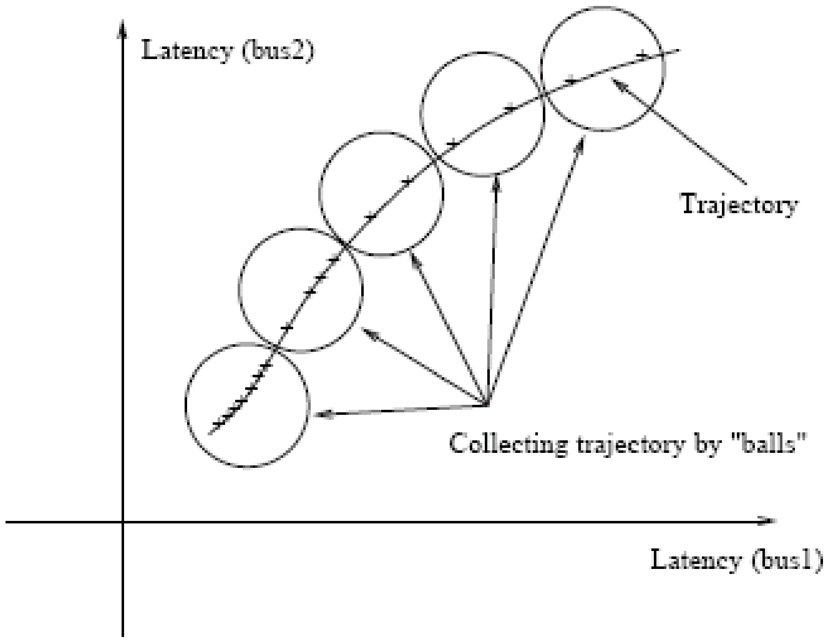

Figure 2.3: Illustration of collecting trajectory in 2-dimension space[3]. The curve in this figure is a trajectory representing moves of a floorplanning process in sampling phase. The "balls" are used to collect the trajectory by covering the trajectory in collecting phase.

to measure CPI is performed. The simulation results for different $\overrightarrow{B}$ can be used to construct the CPI look-up-table by TPWL model.

The construction of the TPWL model consists of the following phases. 1) *Sampling*; 2) *Collecting*; 3) *Simulating*. In sampling phase, each $\overrightarrow{B}$ representing a floorplan in simulating annealing process of floorplanning is recorded, which forms a "trajectory" in the n-dimensional space (n represents numbers of buses of $\overrightarrow{B}$). Then, in the collecting phase, the points of the trajectory is collected in as few "balls" as possible, as shown in Figure 2.3, and forming a trajectory points collecting (TPC) problem. Finally, in simulating phase, the TPC problem in collecting is solved by greedy algorithm. A set of points is obtained, which covers most points in the trajectory with a given radius r. The CPI for each $\overrightarrow{B}$ is obtained by cycle-accurate simulation.

For CPI represents a floorplan not identical to a known bus latency vector, the CPI is obtained by using differential technics. The difference between bus latency vector of known and new floorplan is computed; the unit variation is obtained by differentiation; then the new CPI can be obtained by adding the original CPI of known latency vector and the difference value.

A CPI-aware floorplanning integrating above techniques is shown in Figure 2.4. It starts with a traditional floorplanning with objective of area and wire length, and generates a trajectory. Then, the set of balls C is found by solving the TPC problem on the trajectory. Accurate CPI value for all balls in C are found by cycle-accurate simulation to generate the CPI table. In actual floorplanning, the objective function is the weighted sum of area, total wire length, and CPI (called ALC).

An iterative version of TPWL model which is called iTPWL model is also presented. The iTPWL model is based on the TPWL model and constructed by expanding the CPI table in each iteration. After building the CPI table for the TPWL model (called the first iteration), this CPI table is employed to estimate CPI and conduct SA with objective of ALC. A new trajectory is sampled from the SA optimization process and the CPI table is expanded by adding more entries obtained from the new trajectory2. Experiment results claim that two extra iterations can improve the accuracy of the model and produce a considerably better final floorplan.

This work is also an estimation-based methodology. The heuristic it used is to construct a look-up table for CPI look-up rather than cycle-accurate simulation. The table is constructed by mathematical model. Although it uses bus latencies which are calculated by interconnect lengths as index, the mathematical model is still a little indirect.

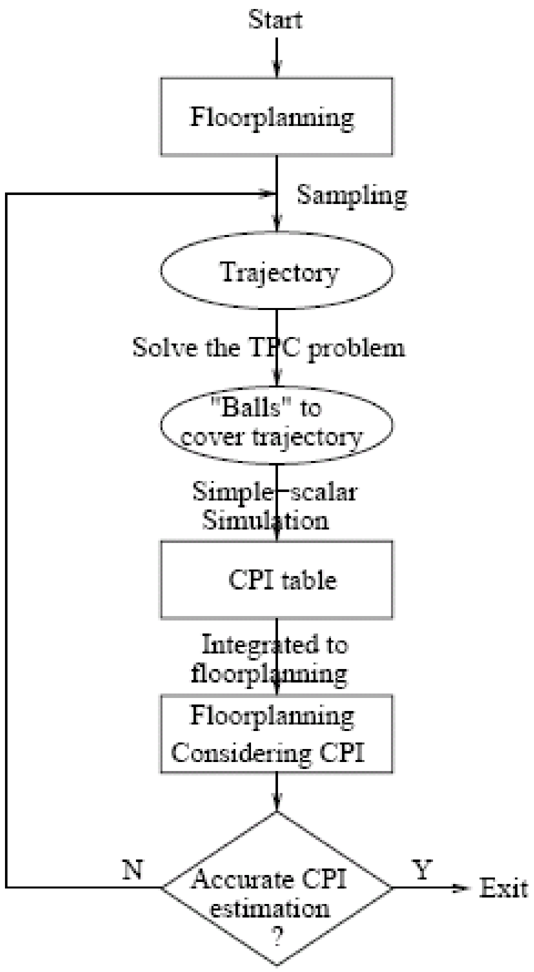
Start
Floorplanning
Sampling
Trajectory
Solve the TPC problem
"Balls" to cover trajectory
Simple-scalar Simulation
CPI table
Integrated to floorplanning
Floorplanning Considering CPI
Accurate CPI estimation ?
N
Y
Exit

Figure 2.4: Overview of CPI-aware floorplanning[3]. An initial floorplan is generated in order to obtain bus latency vectors. The bus latency vectors traversed in floorplanning process are used to construct the TPWL model. The look-up table of CPI is then constructed according to the result from TPWL model. Finally the look-up table is used to guide floorplanner.

2.4 Microarchitecture-Aware Floorplanning Using a Statistical Design of Experiments Approach [4]

The authors suggested in [4] that the addition of latencies on some wires can have a large impact on the overall performance while other wires are relatively insensitive to additional latencies and proposed a statistical design of experiments strategy based on a multifactorial design which uses a limited number of simulations to rank the importance of the wires, identify the throughput-critical wires to be optimized in physical design, and then applies the methodology to floorplanning.

The design flow of [4] are depicted in Figure 2.5. The first step is to quantify the impact of each system bus on the performance for each of the chosen benchmark programs and accordingly assign weights to these wires by cycle-accurate simulation. This step is denoted by Simulation Methodology in Figure 2.5. The simulation methodology here is statistical design of experiments. Statistical design of experiments is a design approach to characterize the response of a system in terms of changes in the factors. The basic idea is to conduct a set of experiments according to a given prescription in which all factors are varied systematically over a specified range of acceptable values such that the experiments provide sampling of the entire search space. However, in this case, it's impractical to use cycle-accurate simulations for all possible configurations in a processor. The authors try to reduce the number of simulations with assumption that each of the factors is restricted to have two levels, the minimum and maximum possible values, such that the greatest response for each input is provoked.

After first step the weights are obtained. The weights may differ across the benchmarks, depending upon the instruction mix executed. The concept of using these weights is similar to [2], but the precise manner in choosing these weights is

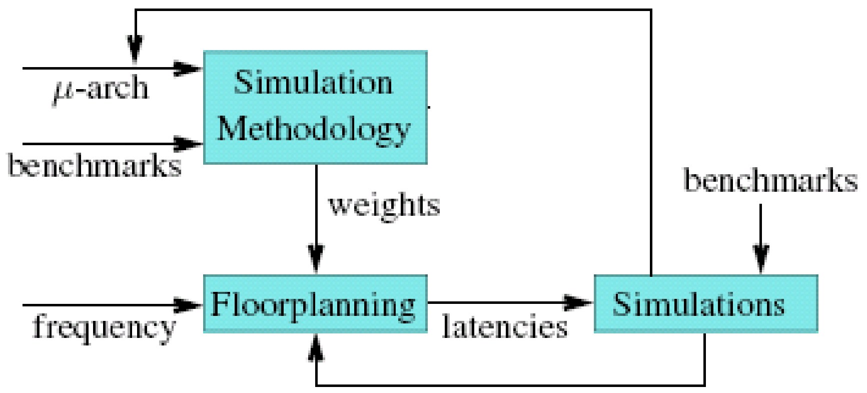

Figure 2.5: Design flow of throughput-aware floorplanning in [4]. Simulation methodology is used to obtain impact on additional latencies of system buses' interconnects in limited simulations. The degree of impact is used as weights of interconnects in floorplanner. The result of floorplanning is verified by simulations.

different.

The weights are then fed to the floorplanner, along with a target frequency. The objective function of the floorplanner is weighted sum of bus latencies and traditional objectives such as area and aspect ratio. The performance of the resultant layout is then estimated from cycle-accurate simulations. If frequency is a design variable, then the floorplanning may be repeated for several frequencies until an optimum design point or performance objective is achieved. In addition, the entire design flow of Figure 2.5 may be repeated for several microarchitectural block configurations to identify the optimal configuration. For a general case, the weights to be used in floorplanning may be obtained by combining the weights obtained from optimizing the processor performance on a set of benchmarks.

The simulator used is modified from SimpleScalar[8] to include extra latencies on buses for additional delays. The benchmarks that are used are MinneSPEC reduced input sets[9] constitute a representative workload for the SPEC benchmarks in order to reduce the complexity of simulation.

This work is also an estimation-based methodology. The authors try to establish the relationship between additional latencies of interconnects and impacts on performance. To quantity the impact they use simulation, which may consume much time even with reduced benchmark suite.

2.5 Problem Formulation

There are many methods to optimize the performance of a given microarchitecture. In this thesis we emphasize the importance of floorplanning as a factor which will impact performance and target on microarchitecture-aware floorplanning for processor performance optimization. To achieve the target we need to consider some kinds of objective functions which are different from conventional ones. Therefore the problem is formulated as follows: in order to achieve microarchitecture-aware floorplanning for processor performance optimization, we wish to find a new methodology of floorplanning which will consider performance in terms of microarchitecture. The new methodology we proposed targets on novel objective function. Given a specific microarchitecture, we wish to find an effective objective function in floorplanning. Rather than conventional objective function which targets on area and wire length, the new objective is targeted to optimize performance of the floorplan in terms of CPI for the given microarchitecture.

Chapter 3

Microarchitecture-Aware Floorplanning for Processor Performance Optimization

In this chapter we will introduce our methodology proposed for microarchitecture-aware floorplanning for processor performance optimization. Before introducing our methodology we use examples to explain the relationship between floorplanning and performance. The flow chart of our methodology and a simulator we implemented are also presented.

3.1 Relationship Between Clock Cycle and Floorplan

In Chapter 1 we have seen two equations, Equation 1.1 for clock cycles of an ALU-type instruction and Equation 1.2 for clock cycles of a shift-type instruction. We reproduce them for convenience as below:

$$Cycle_{ALU} = 6 + (IFID + IDRF * 2 + IDEX + EXALU * 2 + EXMW + MWRF)$$

$$Cycle_{SR} = 6 + (IFID + IDRF * 2 + IDEX + EXSR * 2 + EXMW + MWRF)$$

Note that these two equations differ in the variable EX-ALU and EX-SR only.

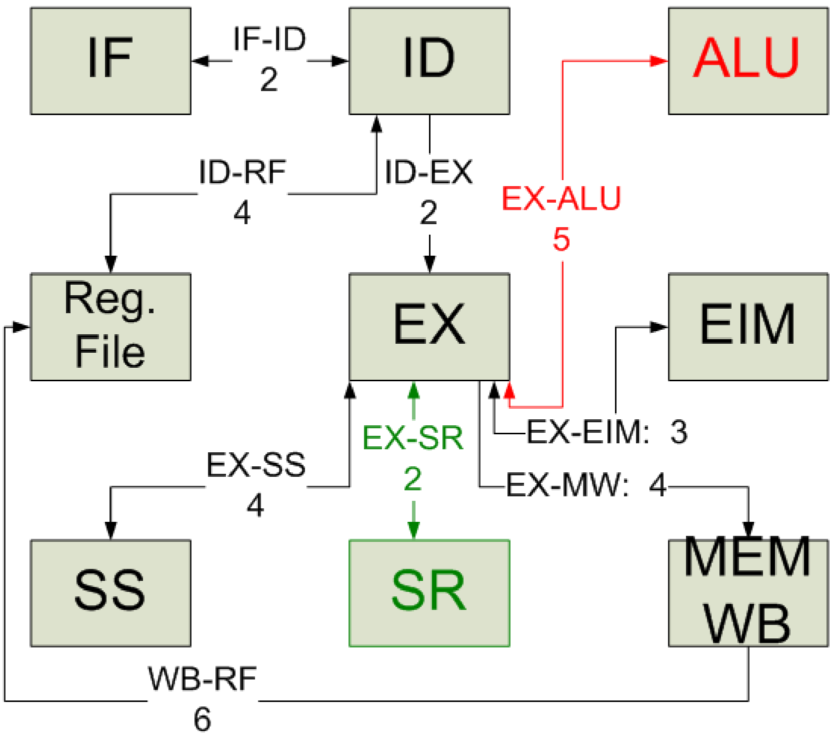

Figure 3.1: Floorplan of a conceptual microarchitecture which interconnections are labeled with latencies.

Here, we assign the variables with real values, which represent latencies of interconnects. See the Figure 3.1 for the values of latencies. Using latencies in Figure 3.1, we can find that an ALU-type instruction takes

$$6 + (2 + 4 * 2 + 2 + 5 * 2 + 4 + 6) = 38\ cycles$$

And a Shift-type instruction takes

$$6 + (2 + 4 * 2 + 2 + 2 * 2 + 4 + 6) = 32\ cycles$$

Now, let us consider another example: the position of ALU and SR is swapped, as shown in Figure 3.2. After this change, the corresponding latencies will also change. For example, the variable EX-ALU is changed from 5 to 2, and EX-SR is change from 2 to 5. Therefore, an ALU-type instruction now takes

$$6 + (2 + 4 * 2 + 2 + 2 * 2 + 4 + 6) = 32\ cycles$$

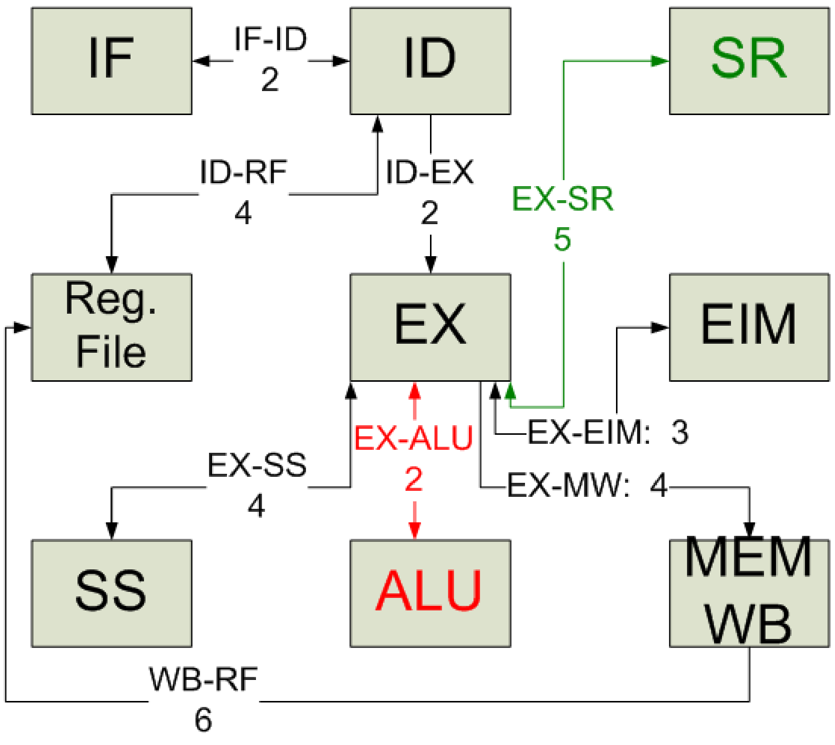

Figure 3.2: Floorplan of a conceptual microarchitecture which interconnections are labeled with latencies. Note that the functional units ALU and SR are swapped compared with Figure 3.1.

And a Shift-type instruction now takes

$$6 + (2 + 4 * 2 + 2 + 5 * 2 + 4 + 6) = 38 \ cycles$$

From the results above, we can find that as the positions of functional units change, their corresponding latencies of interconnects will also change, thereby the time instructions consume will change as well. Take the ALU-type instruction for example, it will consume 38 clock cycles for a single instruction before change in our example, and 32 clock cycles after change. This is a good change in terms of performance for ALU-type instruction, since it takes less time for an instruction to complete. However, this change comes with the price that Shift-type instruction now needs more time to complete for a single instruction, from 32 clock cycles to 38 clock cycles. Of course, this trade-off seems to be reasonable since ALU is most frequently used functional unit in general.

In fact, above discussion is exactly about what is done in floorplanning: the position of blocks are changed to meet a given objective function. Also, as we have seen, the quality of floorplan will have impact performance in terms of clock cycle time an instruction takes.

3.2 Relationship Between Performance and Floorplanning

From previous section we have learned that the floorplanning will impact the clock cycle count. Different types of instructions take different clock cycles to complete, and different floorplan will also influence the clock cycle count of different types of instructions. Therefore the change in floorplanning will influence the total cycle count a program needed, and CPI is changed as well.

On the other hand, we have learned that CPI is relative to the performance. Together with the fact that floorplanning impacts CPI, we can conclude that floorplanning will influence CPI. In other words, the different floorplan means different performance in terms of CPI.

3.3 Proposed Methodology

We have established the relationship between performance and floorplanning as above. We know that the process of floorplanning is usually iteration-based. For example, for a simulated annealing floorplanner, the floorplanning process will iterate with change in temperature. The temperature will go uphill or downhill until a termination condition is meet. The move in temperature means change in floorplan. The cost of changed floorplan is evaluated by the objective function, and whether the move is accepted or rejected is decided by the cost. Therefore the objective function plays an important role in floorplanning since it decides whether a floorplan is accepted or not. However, the objective functions generally used before are like area, total wire length, aspect ratio. The question now is if these objective functions are capable to guide the floorplan to achieve better performance and if these objective functions are able to guide the floorplan to make the change like what we have discussed in Section 3.1.

To resolve these issues, we propose a methodology in hope that it will guide the floorplanning process from the view of performance. We believe it is not enough to consider total wire length and area for performance. The main concept of our methodology is that, for different kinds of instructions, they pass through different functional blocks in the processor. The collection of different functional blocks an instruction pass through is like a path. For example, an ALU-type instruction will pass through a path consisted of functional blocks IF, ID, RF, EX, ALU, MEM/WB; and a Shift-type instruction will go through IF, ID, RF, EX, SR, MEM/WB. We try to guide the floorplanning process in the view of these paths instead of traditional objective functions such as area, total wire length, etc. We try to model a new objective function, which is consisted of the ”ingredients” in these paths. The ingredients are in fact the latencies of interconnects along the paths. We model the delay of each path as function of interconnects like what we showed in Equation 1.1, Equation 1.2 and calculated in Section 3.1. These functions are then used as the objective function to guide the floorplanning process.

To model the latencies of interconnects, the Manhattan distance of two functional blocks form a specific interconnect is used. After the Manhattan distance is obtained, corresponding latency can be calculated. To compute the latency, we use the value of unit interconnect delay from [10], which is 55pS/mm. Using the value of time needed for an unit length of interconnect, the time an interconnect needs can be calculated accordingly by multiply the length of interconnect and time needed for an unit length. Then the latency can be obtained from dividing the time needed for a interconnect by the clock cycle time.

Since there are many kinds of instructions, correspondingly there are many paths exist in a processor. However, there is no need to optimize the floorplan for each kind of instructions. In fact it is also impossible to optimize all kinds of instructions. We need some trade-off between these instructions.

Here we use instruction mix for weighting. Instruction mix is a measure of the dynamic frequency of instructions across on or many programs by definition in [6]. For example, if a program has 14% of Load/Store instructions, 57% of ALU-type instructions, 15% of Shift-type instructions, and 12% of branch instructions, the numbers in percentage is the instruction mix of this program. We can use a simulator to run some simulations on benchmarks for profiling the instruction mix first. The details of simulator will be shown in the next section. After the instruction mix is obtained, it can then be used as the weighting of different paths.

By using this heuristic, the relationship between a floorplan and its performance

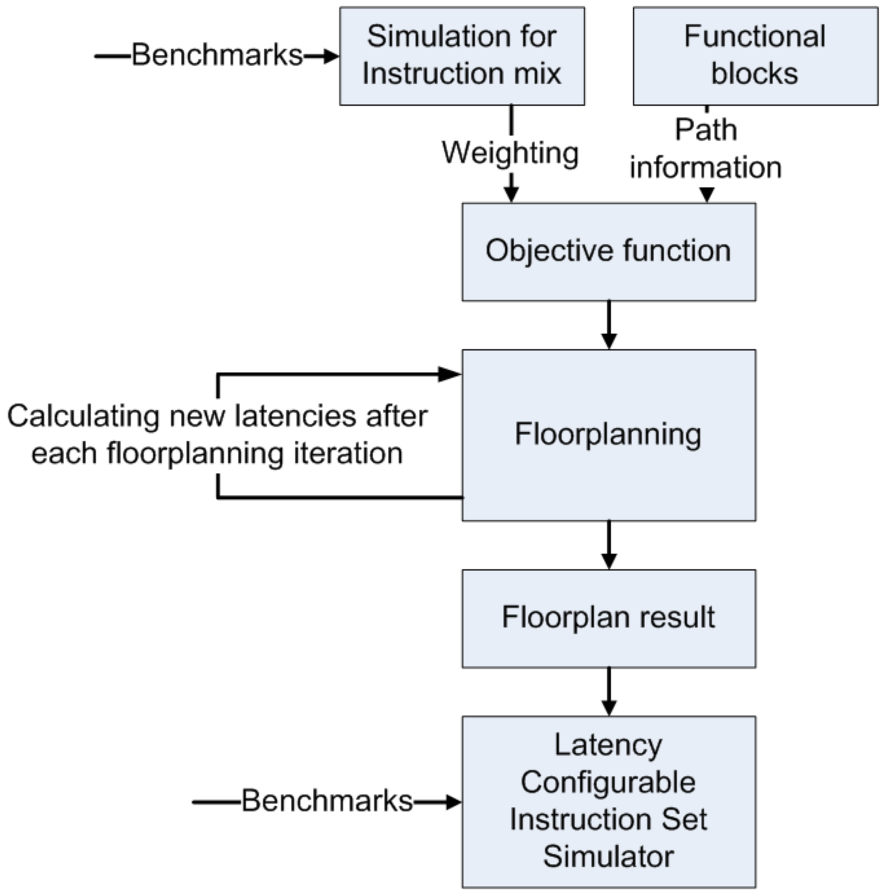
Benchmarks
Simulation for Instruction mix
Functional blocks
Weighting
Path information
Objective function
Calculating new latencies after each floorplanning iteration
Floorplanning
Floorplan result
Benchmarks
Latency Configurable Instruction Set Simulator

Figure 3.3: The flow chart of proposed methodology. The objective function is obtained from characterizing the given microarchitecture. Weights of factors in objective function is obtained from instruction mix. After floorplanning, result of performance is verified by Latency Configurable Instruction Set Simulator.

is established. Floorplan influences clock cycle count of instructions, and clock cycle count of instructions influence performance, therefore floorplan influence performance. Different paths in the processor are weighted differently by their importance, therefore achieving a good trade-off. We believe this heuristic will relate physical information with micro-architectural issues well.

The flow chart of proposed methodology is shown in Figure 3.3.

3.4 Latency Configurable Instruction Set Simulator

We must examine the performance of floorplan from our methodology in order to prove the validity of our methodology. This can be done by cycle accurate simulation. However, the existing simulators do not consider the latencies of interconnects. Further more, the target design we used is a reduced subset of MIPS64 instruction set, thereby no existing simulator is available. Therefore we must implement an instruction set simulator of our reduced subset of MIPS64 instruction set. Besides functional simulation, the simulator also needs the capacity of considering latencies of interconnects. These latencies should be adjustable. We call this simulator Latency Configurable Instruction Set Simulator.

The simulator is written in C++, with the usage of SystemC library. The reason for using SystemC is that it provides convenient data types such as fixed-width signed/unsigned integers, bit vectors, and corresponding functions such as bit range selection, bit-wise logical operations, etc. This simplifies the implementation of the instruction set simulator. The simulator accepts assembly code in hexadecimal format. The instructions are simulated without parallelism. For each instruction the needed register values are first obtained. The result of instruction and target pc (program counter) are then calculated. Result will be written back to memory if needed. After completion of an instruction the simulator will read the next instruction according to the target pc calculated in simulation of previous instruction, and the next instruction is simulated until the HALT instruction is fetched. During simulation the instruction count executed and clock cycles needed including additional latencies introduced by interconnects are tracked and recorded. Data dependencies, corresponding forwarding and additional latencies introduced by pipeline stalls are also taken into account while calculating the cycle count. After simulation the total instruction count and total cycle count of program executed are displayed, and the value of CPI is also calculated from dividing total cycle count of program executed by total instruction count. The instruction mix can also be analyzed and displayed.

Chapter 4

Experimental Results

We present the details of experiment in this chapter. The setup of experiment including target microarchitecture, floorplanner, new objective functions and experimental benchmarks is introduced first. The results are then presented.

4.1 Experimental Setup

4.1.1 Target Microarchitecture

The target microarchitecture we used for implementing our methodology is a MIPS64[11]-like processor. This processor implements a subset of MIPS64 instruction set, which is shown in Table 4.1.

The processor has a 5-stage pipeline as shown in Appendix A.1 in [12]. It also has forwarding capacity. The forwarding architecture is as shown in Appendix A.4 in [12]. It has 64-bit data width general purpose registers and data memory. The instruction width is 32-bit. About the branch operations, the MIPS64-like processor calculates branch condition and target address in ID (instruction decode) stage, the second stage in the processor.

Table 4.1: Instruction set of target design. Note that these are a reduced subset of MIPS64 instruction set.

Type	Instruction
Data Transfer	LW, LWU, SW, LH, LHU, SH
	LD, SD, LB, LBU, SB
Arithmetic / logic/ bit manipulation	DADD, DADDI, DADDU, DADDIU
	DSUB, DSUBU
	AND, ANDI, OR, ORI, XOR, XORI
	LUI
	DSLL, DSRL, DSRA, DSRAV, DSLLV, DSRLV,
	DROTR, DROTRV;
	DSBH, DSHD, DEXT, DINS, SEB, SEH
	SLT, SLTI, SLTU, SLTIU
Control	BEQ, BNE
	MOVN, MOVZ
	J, JR
	JAL, JALR

4.1.2 Floorplanner and New Objective Functions

For implementing our methodology, we must use a floorplanner with our proposed objective function. Here, we use a floorplanner from [13]. It uses the method of simulated annealing. The representation of floorplanning it uses is normalized Polish expressions which enables carrying out the neighborhood search effectively and hence speeds up the search procedure significantly. A simultaneous minimization of area and total interconnection length in the final solution is also utilized so that the interconnection information as well as the area and shape information can be considered simultaneously. Its original objective function consists of area and wire length. There is an adjustable weighting factor for wire length.

In this thesis we consider two new objective functions for verifying our methodology. The first one objective function consists of latency factors which are ingredients of the path instructions pass through, as described in Section 3.1, instead of total wire length. In other words, the latencies of interconnects which instructions may pass through are used as factors of this objective function. These factors are equally weighted. This objective function is targeted at optimizing the latencies of critical path without considering the frequency of path usage. We wish to show it is superior objective function than original ones, since the latencies of interconnects are considered instead of wire length.

The second objective function has the same factors of the previous objective function. However, unlike previous objective function where all factors are equally weighted, these factors are weighted by instruction mix, as we have discussed in Section 3.3. We hope this difference will further improve the result compared with previous objective function.

We also run the floorplanner with its original objective function. Here we utilize two different configurations based on the original objective function. One of these has objective function which area and wire length are equally weighted. The other one is targeted on wire length optimization. The ratio of weighting between area and wire length is 1:30. We wish to show that shorter wire length does not mean better performance necessarily. Also, we can compare the results for area and wire length between original and new objective functions to find out the overhead of proposed methodology on area and wire length.

4.1.3 Experimental Benchmarks

In order to verify the performance, we need some benchmarks. Here we use five benchmarks, written in MIPS64 assembly. They are: DCT (discrete cosine transformation), iDCT (inverse discrete cosine transformation), FIR (finite impulse response), Bubble sort, and Hailstone. Hailstone accepts a number and do the following: If number is odd, multiply by 3 and add 1; if number is even, divide it by 2; this iteration is repeated until number is 1.

Table 4.2: Results of CPI. Note that the result from reducing wire length shows it actually slightly degrades performance. Comparing results from original and new objective function shows that our methodology indeed improves performance.

	DCD	iDCT	FIR	Bubble Sort	Hailstone	Average CPI
Original (1:1)	8.89	6.25	7.69	7.08	6.09	7.20
Original (1:30)	9.01	6.25	7.75	7.28	6.18	7.29
All factors are equally weighted	7.07	5.25	6.25	6.09	5.28	5.99
Weighted by instruction mix	6.26	4.75	5.63	5.39	4.83	5.37

These five benchmarks are first profiled to obtain their average instruction mix. After the floorplanning is completed, the latencies are fed into our latency configurable instruction set simulator to obtain CPI of each individual benchmark. The average CPI can be calculated accordingly.

4.2 Results

We show the CPI results of each individual benchmarks and average CPI of four objective functions in Table 4.2.

The upper two rows in Table 4.2 are the results by original objective function of the floorplanner from [13]. The difference between these two rows is the weighting of factors area and wire length: the ratio of weighting in the first row is 1:1 while 1:30 in the second row. The objective function of second row has more weight on the factor wire length which means it intends to reduce wire length. In the third and fourth row, we represent the result of new objective functions. In the third row, we show the results of objective function which consists of latencies of interconnects with equal weighting. Finally, in the last row, the factors are weighted by the average instruction mix.

We show a chart of comparison between average CPI from different objective functions in Table 4.3. The values in Table 4.3 show the improvement of performance. We know that the relationship of two configurations, X and Y, can be defined as described in [6]:

$$\frac{Performance_X}{Performance_Y} = \frac{Exexution\ time_Y}{Exexution\ time_X} = n \tag{4.1}$$

Also, we can use CPI to denote execution time, as we have discussed in Section 3.1. Therefore, we can rewrite the Equation 4.1 as follows:

$$\frac{Performance_X}{Performance_Y} = \frac{Exexution\ time_Y}{Exexution\ time_X} = \frac{CPI_Y}{CPI_X} = n \tag{4.2}$$

The results in Table 4.3 are calculated according to Equation 4.2. X in Equation 4.2 comes from the row it is in while Y comes from corresponding column. The calculated n is then minus one in order to represent the net improvement and the final value is presented in percentage.

Table 4.3: Comparison between average CPI from different objective functions. The improvement is up to 35.75% comparing original and new objective functions.

	Original (1:1)	Original (30.0)	All factors
Original (1:30)	-1.23%		
All factors are equally weighted	20.20%	21.70%	
Weighted by instruction mix	34.08%	35.75%	11.55%

From the CPI results we can find that the performance of wire length reduction oriented objective function actually degrades, instead of the intuition that reducing wire length may improve performance. For the new objective functions, the results show that it indeed improvs the performance by up to 35.75% when compared to the original objective functions. Compare the results of equally weighted factors and weighted by instruction mix, the latter shows improvement of 11.55%, which shows validation of our methodology.

Despite improvement in performance, there may be overheads. The overheads are mainly in wire length and area. To calculate the overheads we should first find out the actual values of them. Table 4.4 shows the results of wire length and area from four objective functions.

Table 4.4: Results of wire length and total area. The results of two new objective functions are clearly larger than results from original objective function.

	Wire Length	Total Area	Average CPI
Original (1:1)	51375.45	673682.56	7.20
Original (1:30)	49783.50	706621.25	7.29
All factors are equally weighted	55156.06	708309.44	5.99
Weighted by instruction mix	53932.86	741857.50	5.37

From the Table 4.4 we can find that the area and wire length of results from the two new objective functions are larger than those from original objective function, which means that there are indeed overheads. Here we present two tables to show the overheads of wire length and area comparing new objective functions and original objective functions.

Table 4.5: Comparison of wire length and total area between two new objective function and original objective function with equal weight of area and wire length (1:1).

	Wire length	Area
All factors are equally weighted	7.36%	5.14%
Weighted by instruction mix	4.98%	10.12%

Table 4.6: Comparison of wire length and total area between two new objective function and original objective function for wire length reduction (1:30).

	Wire length	Area
All factors are equally weighted	10.79%	0.24%
Weighted by instruction mix	8.33%	4.99%

4.3 Discussion

From the Table 4.5 and 4.6 we can know that the ranges of overheads are from 4.98% to 10.79% for wire length, and 0.24% to 10.12% for area. Specifically, for the case of instruction mix weighted objective function, the worst case of overhead in wire length is 8.33% and 10.12% in area. This may seem to be a large overhead. However, the overheads come with performance improvement over 30%. It is designer's choice to determine the trade-off between performance and wire length/area.

Also note that the CPI results which are significantly larger than ideal case: one. The cause is lack of ability for wire pipelining. We know that pipelining can reduce the CPI to reach ideal case. The ideal case is the longest delay in all pipeline stages. If each stage takes one clock cycle, in ideal case the pipelined CPI will be one since the longest delay in all pipeline stages is one. Without wire pipelining the additional latencies introduced by interconnects in each stage of pipeline will be counted in while calculating the delay of each stage. Therefore the value of CPI is also increased.

Chapter 5

Conclusion and Future Works

This thesis proposed a methodology based on a heuristic to relate performance in terms of microarchitecture and floorplanning, therefore achieving microarchitecture-aware floorplanning for processor performance optimization.

In the past, floorplanner used objective functions focused on reducing wire length and area. These objective functions were considered efficient before since the latencies of interconnects were within single clock cycle or even could be neglected. However, as feature size continues to shrink, the communication of signals on interconnects becomes multi-cycle. The latencies can not be ignored now. These latencies have impact on the performance. However, floorplanner does not consider performance aspect. Hence we propose the methodology as described in this thesis in order to consider performance in floorplanning. We have proven the validity of our methodology since it indeed enhances performance as shown in the experimental results. The results also emphasize the importance of considering performance in microarchitectural aspects.

About the future works, since the experiment is based on a MIPS64-like processor with reduced subset of instruction set, we wish to implement the proposed methodology on a fully-functional MIPS64 processor. This also makes it easier to compare the results of our methodology with previous works. Our methodology also lacks ability to consider wire pipelining which may improve performance further. Therefore future works may also contain the ability of wire pipelining/flip-flop insertion in terms of performance.

Bibliography

[1] Jason Cong, Ashok Jagannathan, Glenn Reinman, and Michail Romesis. "Microarchitecture Evaluation with Physical Planning". In *Proceedings IEEE/ACM Design Automation Conference*, pages 32–35, June 2003.

[2] Mongkol Ekpanyapong, Jacob R. Minz, Thaisiri Watewaiy, HsienHsin S. Lee, and Sung Kyu Lim. "Profile-Guided Microarchitectural Floorplanning for Deep Submicron Processor Design". In *Proceedings IEEE/ACM Design Automation Conference*, pages 634–639, June 2004.

[3] Changbo Long, Lucanus J. Simonson, Weiping Liao, and Lei He. "Floorplanning Optimization with Trajectory Piecewise-Linear Model for Pipelined Interconnects". In *Proceedings IEEE/ACM Design Automation Conference*, pages 640–645, June 2004.

[4] Vidyasagar Nookala, Ying Chen, David J. Lilja, and Sachin S. Sapatnekar. "Microarchitecture-Aware Floorplanning Using a Statistical Design of Experiments Approach". In *Proceedings IEEE/ACM Design Automation Conference*, pages 579–584, June 2005.

[5] P. N. Glaskowsky. "Pentium 4 (Partially) Previewed". *Microprocessor Report*, 14(8):11-13, August 2000.

[6] David A. Patterson and John L. Hennessy. *"Computer Organization and Design: The Hardware/Software Interface"*. Morgan Kaufmann Publishers, Inc., second edition, 1998.

[7] D. C. Burger and T. M. Austin. "The SimpleScalar tool set version 2.0". Technical Report CS-TR-97-1342, The University of Wisconsin, Madison, June 1997.

[8] T. M. Austin. "Simplescalar tool suite". http://www.simplescalar.com.

[9] A. J. KleinOsowski and David J. Lilja. "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research". *IEEE Computer Architecture Letters*, vol. 1, June 2002.

[10] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. "The Future of Wires". In *Proceedings of the IEEE*, 2001.

[11] http://www.mips.com/content/Products/Architecture/MIPS64/.

[12] John L. Hennessy and David A. Patterson. *"Computer Architecture - A Quantitative Approach"*. Morgan Kaufmann Publishers, Inc., third edition, 2003.

[13] D. F. Wong and C. L. Liu. "A New Algortihm for Floorplan Design". In *Proceedings IEEE/ACM Design Automation Conference*, pages 101–107, 1986.