UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of a doctoral dissertation by

Vidyasagar Nookala

and have found that it is complete and satisfactory in all aspects,

and that any and all revisions required by the final

examining commitee have been made.

Prof. Sachin S. Sapatnekar	Prof. David J. Lilja
Name of Faculty Adviser	Name of Faculty Co-Adviser
Signature of Faculty Adviser	Signature of Faculty Co-Adviser
Date	Date

GRADUATE SCHOOL

PHYSICALLY-AWARE SYNTHESIS AND MICROARCHITECTURE DESIGN

A DISSERTATION

SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL

OF THE UNIVERSITY OF MINNESOTA

BY

VIDYASAGAR NOOKALA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Sachin S. Sapatnekar, Adviser

David J. Lilja, Co-Adviser

April 2007

ACKNOWLEDGEMENTS

I am deeply indebted to my advisers, Prof. Sachin Sapatnekar and Prof. David Lilja, who are among the most brilliant and wonderful people I have ever met. They have played an important role in shaping my personality, technically and otherwise, and I feel very fortunate to be associated with such geniuses. I learned so much from them, and it is safe to say that this thesis would not have been possible without their support. I also thank Prof. Kia Bazargan and Prof. Antonia Zhai for agreeing to serve on my dissertation committee.

I acknowledge my contemporaries at the VEDA and Reconfigurable Computing labs, namely, Shrirang Karandikar, Cristinel Ababei, Rupesh Shelar, Tianpei Zhang, Venkat Rajappan, Yong Zhan, Brent Goplen, Pongstorn Maidee, Hongliang Chang, Haifeng Qian, Jaskirat Singh, Anup Sultania, Satish Sivaswamy, Hushrav Mogal, Sanjay Kumar, Baktash Boghrati, Saeed Shamshiri, Qunzeng Liu and Xianghong Liu, for their help and friendship. Special thanks to Jaskirat Singh and Pongstorn Maidee, whose graduate careers have almost coincided with mine; We took so many courses together, I really cherish my interactions with them. Thanks to Ying Chen, who introduced me to the world of microprocessor simulations and tutored me on using the facilities at the Supercomputer Institute.

I am very lucky to find a wonderful group of friends who made my stay in the Cities very memorable and enjoyable. The time I have spent with Nikhil Wale, Deepa and Manish Kapoor, Jaskirat Singh, Abhinav Das, Salim Charaniya, Aditya Saxena, Ranjit Eswaran, and Praveen Ghanta has left me a fond set of memories that I can never forget. The list is endless, I thank them all and will miss them. Special mention to Nikhil, with whom I have been roommates during my entire stay in Minnesota; We shared so much, I am more or less an open book to him and I cannot imagine a better roommate. Thanks are also due to Syed Suhaib and Dan Sandler, my fellow interns at Intel, and Dr. Mike Kishinevsky, my supervisor at the Strategic CAD Lab.

In addition, I owe thanks to several institutions and agencies, namely, the ECE department at Minnesota, National Science Foundation, Semiconductor Research Council, Intel and Minnesota Supercomputer Institute, for providing me with the financial support and facilities required to pursue the graduate study. I also thank the ECE system support staff, namely, Chimai Nguyen, Dale Swanson and Joe Berens, for their help.

Finally, I owe a lot to my family for being with me through all the ups and downs of my life; I would have been nowhere without its support.

Dedicated to my grandmother.

“*When I want to read a good thesis, I write one!*”

-Adapted from Benjamin Disraeli

PHYSICALLY-AWARE SYNTHESIS AND MICROARCHITECTURE DESIGN

Vidyasagar Nookala, Ph.D.
Department of Electrical and Computer Engineering
University of Minnesota, Twin Cities, 2007
S. S. Sapatnekar and D. J. Lilja, Advisers

As across-chip interconnect delays can exceed a clock cycle, wire-pipelining becomes essential in high performance synchronous designs. Although wire-pipelining allows higher frequencies, it may change the circuit altogether because of the nonuniform increase in the latencies of the paths and cycles of the circuit. More importantly, it can reduce the delivered performance of a microarchitecture, since the extra flip-flops inserted may increase the operation latencies and stall cycles. Furthermore, the addition of latencies on some wires can have a large impact on the overall throughput while other wires are relatively insensitive to additional latencies. In addition, the high frequencies, coupled with high integration densities, have made operating temperature an important concern due to the nonlinearly increasing cooling costs.

Physical design, which determines the lengths of the multicycle wires and the spatial distribution of power dissipation sources, plays an important role in determining the throughput and thermal characteristics of a microarchitecture. Moreover, changes in the throughput can affect the power consumption levels through variations in the activity patterns.

In this thesis, we examine two problems related to wire-pipelining and operating temperature, one each at the circuit- and microarchitecture- levels. First, we formulate a method to automatically correct the functionality of a wire-pipelined circuit. The proposed method finds the minimal value of the input issue rate slowdown required for a circuit as it affects the throughput of the circuit. The formulation may introduce extra registers into the circuit in the process of correction, and attempts to minimize the number of extra flip-flops thus added. When experimented on the ISCAS benchmarks, the results suggest that wire-pipelining increases the overall throughput in most of the cases.

The second part of the thesis addresses interactions between microarchitecture and physical design stages. We propose a strategy for floorplanning that attempts to minimize the throughput loss that comes with wire-pipelining. We employ a statistical design of experiments strategy, which intelligently uses a limited number of simulations to rank the importance of the wires, and this information is used by the floorplanner to optimize the throughput-critical wires by keeping them short. Our results over a number of SPEC benchmarks show improvements in the overall system performance when compared with an existing technique. Additionally, we compare a couple of simulation time reduction techniques that can be used to speed up the simulation strategy.

Next, we extend the throughput-aware floorplanning methodology to incorporate thermal issues. The approach uses instantaneous dynamic power dissipated in the blocks of a microarchitecture to find a placement that is optimal on a combination of the thermal and the throughput attributes. We also model the dependence between the throughput and power, and and uses transient analysis for thermal estimation. The thermal objectives that we consider are the peak and average temperatures. The results indicate significant improvements in both the peak and the average over a previously proposed approach.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

CMOS process scaling has allowed a steady increase in the operating frequencies of integrated circuits, as predicted by Moore's Law [Moo65] by enforcing a steady decrease in device intrinsic delays; semiconductor industry trends suggest that the operating frequencies of leading edge integrated circuits approximately double every process generation [Bor00], in tune with the projections of Moore's Law [Moo65] and the International Technology Roadmap for Semiconductors, ITRS [Sem01]. For many years, the performance of a VLSI circuit was determined solely by the delays of the devices of the circuit. However, the impact of wire delays on system performance, which was negligible in earlier technology generations, has been growing steadily due to the mismatch in the scaling trends of device and interconnect delays over process generations. Although process scaling causes a considerable decrease in device delays, the same is not true in the case of interconnect delays.

The main reasons behind this trend are the increasing line resistances and coupling capacitances due to the decreasing wire cross-section and intra-layer wire spacing, respectively. The impact of the mismatch is more pronounced in the deep submicron (DSM) regime, particularly at the nanometer technology nodes, where interconnect delay became a major and dominating contributor to the shrinking clock cycle time. The increasing criticality of the interconnects has presented a variety of problems to the research community. Various approaches have been proposed to counter the interconnect delay problem, such as:

- The use of copper [EHG$^+$97] to replace aluminum for wiring reduces line resistance of the wires due to its lower resistivity.

- Repeater insertion [vG90], especially for long wires , typically linearizes the dependence of interconnect delay on its length.

- Appropriate wire sizing [CL95] can reduce wire delays.

Although such techniques helped in keeping the interconnect dominance problem at bay in the earlier technologies, the high frequencies, typically in the gigahertz range, projected and employed in the nanometer circuits make design feasibility beyond the scope of wire delay optimization. The scenario is further aggravated by the fact that die sizes increase by 7% with every process generation [Bor00], resulting in even longer wire lengths, and hence longer wire delays. In other words, while techniques such as [vG90, CL95, EHG$^+$97] work well for small or local interconnects, even the theoretically best optimizers cannot ensure that the delay of a long global wire does not exceed a clock period. For instance, even after aggressive optimization, a 2cm global interconnect, a common occurrence in nanometer designs, has a projected delay of 0.67ns in 70nm technology [Con01], placing an upper bound of about 1.5GHz on the operating frequency, much less than the multigigahertz frequencies projected for that technology. As pointed out in [SMCK04], the maximum distance a signal can travel along an optimized interconnect in a clock cycle gradually decreases as the technology further scales down, indicating an increase in the fraction of the interconnects whose delays exceed a single clock period.

In addition to the above mentioned interconnect delay problem, another issue that has become an important concern in deep submicron technologies is the operating temperature, particularly in microprocessor circuits due to the high power densities associated with high operating frequencies and integration densities; it has been observed that the power consumption of cutting-edge microprocessors doubles every four years [Bor00, GBCH01]. For high power dissipations, the cost of cooling has a nonlinear relationship with power [GBCH01], as shown in Figure 1.1. indicating expensive cooling solutions.

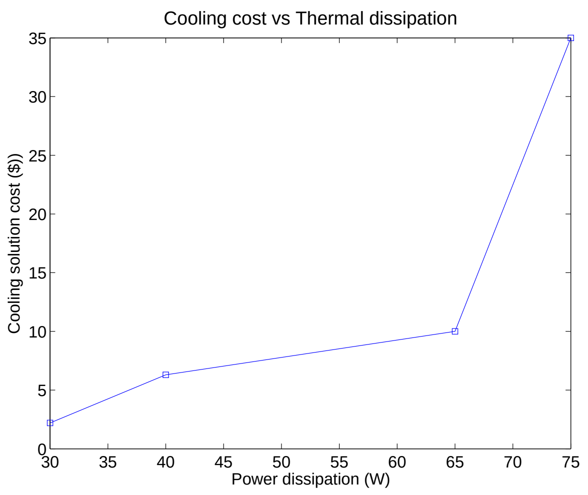


Figure 1.1: The nonlinearly increasing cooling cost. The plot is taken from [GBCH01]

1.1 Handling multicycle wire delays

The emerging dominance of the across-chip delays over device delays has forced the designers to embrace alternative design methodologies that will enable multicycle across-chip communication, so that across-chip interconnect is removed from all the timing constraints, and the chip speed is determined by the most critical intra-block/local combinational path, in order to continue employing higher operating frequencies. Some of the approaches which can be used to implement multicycle global communication are listed below:

1.1.1 Slower clock for the flip-flops latching signals from global wires

In this approach, each of the signals from the global wires whose delay is greater than the system clock period are latched by the flip-flops clocked by the new, slower clock network. However, this approach adds new complications in the form of routing the extra clock network and synchronization between the clock domains. Moreover, since the slower clock must consider the worst case across-chip wire delay, latching signals from wires whose delay is considerably smaller than the slower clock period degrades the throughput of the circuit.

1.1.2 Globally Asynchronous Locally Synchronous (GALS) design methodology

This approach, proposed in [Cha84], advocates an asynchronous style design for hiding the effects of global interconnect delays. The basic idea is to alter a circuit such that it works under the assumption of zero-delay inter-block connections. The communication between the synchronous subsystems (or blocks) of a circuit, each of which can have a different clock, is based on a full handshake protocol. For this purpose, each block of the circuit is wrapped around by an asynchronous interface. Several other works have been proposed based on this approach, such as [Sei94, BC97]. However, the overhead for the asynchronous interface may affect both the performance and the area of the design.

1.1.3 Elastic systems

An elastic design or a latency insensitive design [CKG06, CMSV01] is similar to a GALS system in that each block of a circuit is surrounded by a wrapper that communicates with the neighboring blocks. However, the difference is that an elastic system uses a synchronous framework for global communication unlike GALS, which relies on handshake protocols. In such a paradigm, each block of the circuit is required to be “stallable”, which is accomplished by encapsulating the blocks of the circuit with wrappers and connect them through internally pipelined elements called elastic controllers in [CKG06] and relay stations in [CMSV01], which comprise memory elements such as flip-flops and some control logic. All of the wrappers and relay stations/elastic controllers comply with a formally defined protocol, which forms the basis for a correct-by-construction methodology. Like GALS, the disadvantage of this approach is the area overhead accrued due to the relay stations, in addition to the shells. A further limitation is that every communication channel must be provided with a couple of additional control signals, such as “valid/void” to identify valid data and “stop” to implement the backpressure or feedback mechanism, which may add up to the total wiring requirements, increasing the already critical wire congestion. Although the authors of [CM04a], who used the software pipelining based approach of [BASB01] to stall the blocks of the circuit, alleviated some of the aforementioned problems by reducing the complexity of relay stations and eliminating the “valid/void” and “stop” signals, elastic systems (and GALS) have yet to find widespread use, partly due to the lack of adequate CAD support.

1.1.4 Wire pipelining

The delay of an interconnect is distributed over several clock cycles by inserting flip-flops along the interconnect. As an example, consider the case where a chip has a 2cm wire, with a projected delay of 0.67ns in 70nm technology, which puts an upper bound of 1.5GHz on the operating frequency. To operate the chip at a frequency of 3GHz (corresponding to a clock period of 0.33ns), the delay of the 2cm wire can be spread over two clock cycles by inserting a couple of flip-flops on the wire. This approach is analogous to classical hardware pipelining, where the logic of a circuit is spread over multiple stages in order to employ a higher clock frequency. In addition, *wire-pipelining* can be treated as an extension to the repeater insertion, where some of the repeaters inserted on a wire, while optimizing its delay, are clocked, i.e., memory elements such as flip-flops.

Though it complicates the clock network routing, wire-pipelining, besides allowing higher operating frequencies, enables designers to remain in the purview of the traditional VLSI design methodology, and therefore has become a popular approach to realize multicycle global communication in the nanometer process technologies. For instance, Intel used wire-pipelining in the Itanium processor to realize an operating frequency of up to 1.7GHz in 180nm technology [MLH$^+$00].

In this thesis, we focus on wire pipelining and propose solutions to a few issues associated with the technique that are outlined in the next section. In addition, we assume that all of the flip-flops are edge-triggered.

1.2 Issues with wire-pipelining

Although pipelining the interconnects having multicycle delays in a circuit permits higher operating frequencies, there are several issues associated with the wire-pipelining scheme:

- *Functional correctness:* The nonuniform introduction of extra flip-flops into a circuit can alter its cycle level behavior, requiring correction. Specifically, the number of latencies inserted on two different paths from a pair of blocks can be different depending upon the lengths of wires of the paths.

- *Throughput reduction:* The increase in the number of clock cycles required for each computation can result in reduced throughput.

  The throughput reduction is dictated by the amount of pipelining required by the wires that form loops or cycles in the circuit. At the circuit-level, this is clearly determined as the ratio of the post- and pre- wire-pipelining latencies of the cycles of the circuit. This concept, which we call *slowdown* is dealt in detail in section 2.2.

  However, at the microarchitecture-level, the throughput reduction can be thought as the increase in the number of clock cycles to execute an instruction. For instance, inserting a flip-flop on the wire connecting *issue* and *adder* units of a microprocessor increases the latency of *add* operation by one clock cycle, which prolongs the execution of the program run on the microprocessor.

1.3 Research contributions

In this thesis, we address two problems, one each at the circuit- and microarchitecture-levels. The two problems are detailed in Sections 1.3.1 and 1.3.2, respectively.

1.3.1 Functional correction

We present an approach for correcting the functionality of wire-pipelined circuits. Given a circuit and a wire-pipelined version of the circuit, which may be functionally incorrect, we formulate a method to correct the functionality of the wire-pipelined circuit. The technique provides a minimum area solution to the problem, to minimize the number of additional flip-flops that are required to be inserted on some wires of the circuit to maintain functional equivalence. The method also ensures that the throughput slowdown described in the previous section is kept at the minimum possible level.

1.3.2 Microarchitecture-aware floorplanning

A typical microprocessor design methodology, shown in Figure 1.2, can be broadly classified into the following three steps:

- Microarchitecture design: In this step, the basic functionality issues of the design are dealt. The step determines the Instruction Set Architecture (ISA) of the architecture and the high-level implementation details such as pipelining, cache sizes, etc.

- Compiler design: This step involves translating text and applications written in a programming language such as C into assembly instructions for the ISA.

- Circuit design: The objective of this step is to transform the high-level description of the processor into a transistor-level circuit. The step includes routines such as logic design, where the high-level description is translated into a network of transistors and wires, and physical design that determines a placement of the network on the chip layout. The circuit design step is followed by layout generation, and finally a silicon implementation of the circuit.

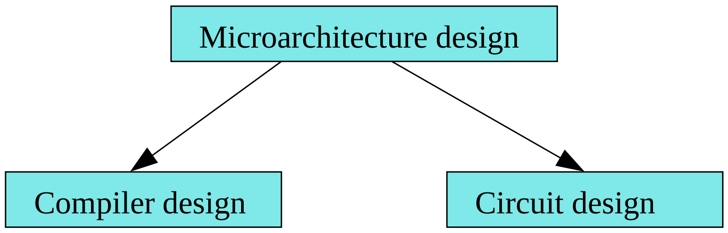


Figure 1.2: An abstract view of microprocessor design flow.

The total execution time, T_{exec}, of a program on a microprocessor can be expressed as the product of three terms [Lil00], as shown below:

$$T_{exec} = N_{inst} \cdot \text{CPI} \cdot T_{clk} \tag{1.1}$$

Where N_{inst} is the number of executed instructions, typically the instruction count of the program, CPI is the average number of instructions per cycle and T_{clk} is the clock period. The throughput of the microprocessor, measured as the average number of instructions per clock cycle (IPC), is the reciprocal of CPI.

It can be observed that the execution time can be reduced by decreasing either of the three terms. In a typical design flow, optimizing for the number of instruction N_{inst} and CPI has been solely in the hands of the microarchitecture and compiler design stages. The job of circuit design has been to minimize the clock cycle time T_{clk}, subject to the design specification passed on from the microarchitecture design step.

As noted in Section 1.2, wire pipelining can cause a reduction in the throughput of the circuit, i.e., increase in CPI, due to the increase in the number of clock cycles per computation. In addition, the amount of pipelining required by the wires of a circuit is typically determined at the physical design step, particularly, at the higher stages such as floorplanning, i.e., block-level placement. This indicates that circuit design can impact the throughput of a microarchitecture, through the routines of the physical design stage.

Under such a scenario, traditional physical design methodology, which focuses only on minimizing the clock period and topological aspects such as area and aspect ratio, can result in processors that are suboptimal in throughput. Specifically, the impact of inserting additional flip-flops can vary across the buses/wires of the microprocessor, depending upon the instruction mix executed. For better throughput, it is imperative to minimize the amount of pipelining required by performance critical wires by keeping them short.

On the similar lines, the thermal characteristics of a chip are determined by not only the power consumption but also the placement of the devices on the chip layout through various mechanisms of heat transfer that take place in the chip. Therefore, for a better thermal solution, microarchitecture optimizations, which typically involve power minimization and other similar procedures, must go hand in hand with the placement strategy.

Such a scenario asks for interaction between the microarchitecture and lower design phases, particularly physical design, in order to achieve better performance or thermal characteristics. In this thesis, we propose methodologies towards introducing microarchitecture awareness in floorplanning, an early physical design stage that has a major share in determining global wire delays. We first propose a technique for throughput-aware floorplanning and then extend it to include operating temperature in the optimization objectives. We apply the methodology on two different architectures, namely, the DLX [HP97] and the Pentium (P6) [HP96] machines.

As is typical with microarchitecture optimizations, the methodology requires cycle-accurate simulations on a set of benchmarks to evaluate the throughput and power consumption of a microarchitecture. Due to the exponential number of floorplan configurations possible, it is impractical to use simulations for each candidate floorplan that is evaluated during optimization. We employ a statistical design of experiments (DOE) based approach [Mon00] to reduce the number of cycle-accurate simulations required from exponential to a manageable limit.

1.4 Techniques for wire-pipelining

In case of the Itanium processor, wire-pipelining was performed manually: the global wires violating the clock cycle time were identified, and flip-flops were manually inserted into the RTL description by the designer. Until recently, there have been few, if any, methods for automated wire-pipelining, and several methods have been proposed in the last couple of years, some of which are explained in the next few paragraphs.

It is reasonable to assume that the delay of an optimally buffered interconnect varies linearly with its length [She95]. If we can determine the maximum length of a wire whose delay is within a clock period, called *critical sequential length* in [SMCK04], then the number of clock cycles required by a signal traveling the length of a particular wire can be estimated as the ratio of its length to the critical sequential length. The authors of [LZKC02] use this idea to pipeline an interconnect for a given clock cycle time. The approach uses the Elmore delay model [Elm48] to analytically compute the minimal number of buffers required to optimize the delay of a wire of a certain length, and from this, estimate the critical sequential length.

In addition, the work identifies the feasible regions for inserting each of the estimated number of flip-flops and buffers on a wire, without violating the clock period requirements.

Two other recent works [Coc02,HAT02] approach wire-pipelining at the global routing level. The technique of [Coc02] finds a wire-pipelining solution to optimize a given interconnect topology such as a Steiner tree [She95]. This approach extends the dynamic programming based buffer insertion algorithm of [vG90] by augmenting the buffer library with flip-flops. Given a Steiner tree, target clock period, required times at each of the destinations, candidate buffer/flip-flops insertion locations, a buffer and flip-flop library, the algorithm finds an optimal assignment of flip-flops and buffers on the buffer locations, which minimizes the number of flip-flops between the source and the latest destination of the net. Another work [SZH04] extends the method of [Coc02] for pipelining using latches, instead of edge-triggered flip-flops.

In [HAT02], wire-pipelining is handled in conjunction with global routing. The approach, based on the fast path algorithm [ZLA00], is to simultaneously route and insert buffers and flip-flops to optimize a two pin wire. The algorithm transforms the chip area into a grid graph, where the edges and vertices corresponding to the given blockages are deleted, and finds a minimal latency route from the source to the sink of the net. The solution involves the propagation of wave-front from a vertex to its neighbors, similar to maze routing [She95]. Both of the approaches use distributed Elmore wire delay models, and keep track of multiple partial solutions at every step and use techniques to prune inferior solutions to decrease the search space.

1.5 Thesis organization

The remainder of the thesis is organized as follows. Chapter 2 introduces the functional correctness problem in wire-pipelined circuits in detail and presents a solution for regaining the functionality. Chapter 3 details some background information on superscalar processors and other preliminaries associated with the content of the next few chapters. A methodology for throughput-aware floorplanning along with a comparison of various simulation time reduction techniques is provided in Chapter 4. Chapter 5 applies the methodology of Chapter 4 for the Pentium architecture. In Chapter 6, the floorplanning flow is extended to incorporate temperature issues into optimization. Finally, Chapter 7 presents conclusions of this thesis and some future directions.

Chapter 2

Functional correction of wire-pipelined circuits

In this chapter, we propose a minimum area solution to the functional correction problem introduced in Section 1.3.1. The solution is formulated as an Integer Linear Program (ILP).

The chapter is organized as follows. Section 2.1 overviews some related work on wire-pipelining, which is followed by Section 2.2, which describes the problem and an intuition behind the solution. Section 2.3 introduces the terminology used in this paper, while a mathematical formulation for the problem solution and area minimization is developed in Section 2.4. Section 2.5 presents the implementation details and experimental results, while Section 2.6 addresses a few related ideas. We finally conclude the chapter in section 2.7.

A preliminary version of the work has been published as the Masters thesis [Noo04] of the author. The new contributions include optimization of the the ILP run time and an analysis of power dissipation in the context of wire-pipelining.

2.1 Related work

There have been some attempts to address wire-pipelining at circuit-level in the recent few years, most of them use the technique of retiming [LRS83] as the underlying framework. The works of [LZ03, CYTD03] extend retiming by including the interconnect delays, in addition to the gate delays, for pipelining the wires of a circuit. Another work [TTBN00] combines retiming at floorplanning level with module selection to consider wire latencies. The objective is to find a floorplan, with module selection, that minimizes the area of the floorplan subject to a lower bound on each wire latency. The advantage of such (retiming-based) implementations is that the functionality of the circuit is not altered, due to the use of retiming as the underlying framework. On the other hand, since retiming preserves the latencies of the cycles and input-output paths of the circuit, there is a lower bound on the achievable clock cycle time.

2.2 Problem description

A typical design flow may proceed as follows. After the blocks and modules of the circuit are designed subject to a clock frequency, a block-level placement of the circuit is performed. Wire pipelining is then carried out on the global wires of the circuit, sometimes concurrently with routing [HAT02], or sometimes after routing is done [Coc02], and this may insert flip-flops on a wire if the delay of the wire exceeds a clock cycle. After the wires of a circuit are pipelined, the following two problems must be resolved:

- Increase in the latencies of the cycles of the circuit.
- Nonuniform increase in the latencies of different paths to a block from the inputs of the circuit.

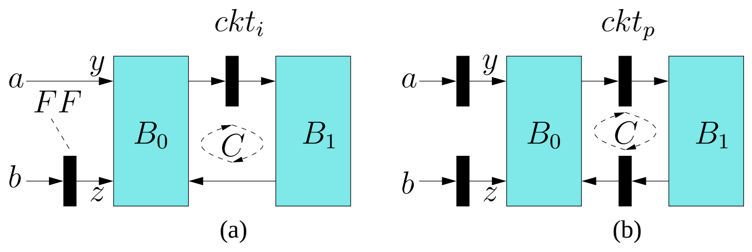


Figure 2.1: A circuit with two inputs a and b. Signals y and z are the input ports of the block B_0. (a) The circuit before pipelining its wires (ckt_i). (b) The circuit after pipelining its wires (ckt_p).

Consider Figure 2.1, which depicts a circuit comprising two combinational logic blocks B_0 and B_1, which also form the cycle C, before and after pipelining the wires of the circuit. The two scenarios are labeled ckt_i and ckt_p, as shown in Figures 2.1(a) and 2.1(b), respectively. The insertion of an extra flip-flop on the cycle C increases its latency to 2 in ckt_p from 1 in ckt_i. Hence, the output of each block of C propagates back to itself after 1 clock cycle in ckt_i, whereas it takes an extra clock cycle in ckt_p, thus altering the original functionality of the cycle. Moreover, with the insertion of an extra flip-flop between a and y, the inputs a and b reach y and z, respectively, after an equal number of clock cycles in ckt_p, which is not the case in ckt_i. Hence, ckt_i and ckt_p are not functionally equivalent.

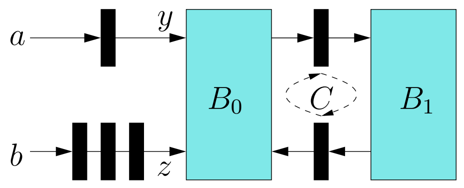


Figure 2.2: A solution to the problem shown in Figure 2.1. We refer to this circuit as ckt_f.

Wire-pipelining can therefore result in a totally different microarchitecture. This is not the desired result and therefore, must be corrected, and this thesis proposes a method for doing so. The solution lies in ensuring that every block receives its inputs at the correct clock cycle. For increased cycle latencies, we use an approach similar to the *c-slow* concept mentioned in [LRS83]. The idea is to *slowdown* the input issue rate[1] of the circuit by some factor ρ, i.e., inputs are allowed to change only every ρ^{th} clock cycle. The issue rate of the initial circuit ckt_i is assumed to be 1.

For instance, the cycle C of ckt_p will be functionally equivalent to the cycle C of ckt_i, if the inputs a and b are permitted to change only every other clock cycle in ckt_p. As a result, ckt_p computes its outputs only every 2 clock cycles, which indicates a reduction in the throughput of the circuit. In addition, the latency difference between

[1]The issue rate is defined as the number of clock cycles between successive input changes. An issue rate of 1 indicates that the inputs can change every clock cycle.

any two paths to a block from the inputs of a circuit must also be maintained in its wire-pipelined version. Going by this argument, since the latency difference between the paths $b \rightarrow z$ and $a \rightarrow y$ is 1 in ckt_i, and 0 in ckt_p, one extra flip-flop must be inserted on the path $b \rightarrow z$ in ckt_p to make it functionally equivalent to ckt_i. However, the slowdown has implications on the path latencies of a wire-pipelined circuit. For example, the latency difference of the paths $a \rightarrow y$ and $b \rightarrow z$ in ckt_i must be amplified by a factor of $\rho = 2$ in ckt_p, since it receives its inputs only every 2 clock cycles. Therefore, 2 extra flip-flops must be inserted on the path $b \rightarrow z$ in ckt_p, as shown in Figure 2.2.

Our work finds the minimal value of slowdown required for a circuit as this directly affects its throughput and also minimizes the increase in area due to the insertion of extra flip-flops in the process of correction.

2.3 Preliminaries

In the example in section 2.2, it was assumed that all blocks were purely combinational. In general, a circuit may have sequential as well as combinational blocks, i.e., the blocks may have internal flip-flops and/or cycles. The existence of cycles in a circuit may require that extra flip-flops be inserted within a sequential block of the circuit. For instance, consider a scenario where there are two paths from an input of a sequential block to one of its outputs. If the two paths have different latencies, and if the circuit requires a slowdown $\rho > 1$, then the solution will require that the difference of latencies be increased by a factor of ρ. Therefore, all of the wires of the block must be considered for the insertion of extra flip-flops. However, in most cases, the blocks are internally undefined blocks at an early stage of design, or IP cores, and therefore, arbitrary insertion of extra flip-flops on the wires within the blocks is not desirable. To avoid this, we use an abstract model for a sequential block that decomposes it into a set of combinational sub-blocks, interconnected by wires having flip-flops. This ensures that for any sequential block, only those interconnections that have flip-flops on them are considered for insertion of extra flip-flops. Figure 2.3 shows a sequential block and the abstract model of the block. The block is modeled as two combinational sub-blocks, S_1 and S_2, with flip-flops on the interconnections between them.

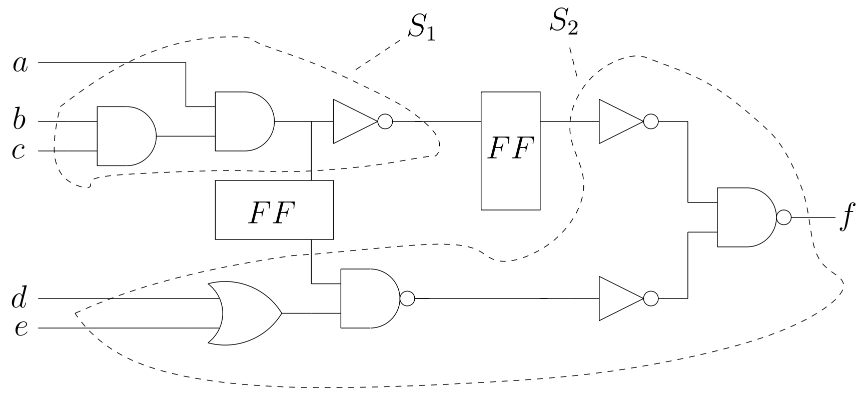


Figure 2.3: A sequential block and its abstracted model.

For a general circuit, we will consider three scenarios: the *initial circuit*, a *wire-pipelined* version of the initial circuit, and a *corrected wire-pipelined* version of the initial circuit. Flip-flops and repeaters apart, each of the three circuits consists of the same placed and routed combinational block level or sub-block level netlist. Each net of the circuits is a routed tree that connects the output of a block/sub-block (source) to the inputs of other blocks/gates (sinks) through branch points such as Steiner points [She95]. We use three edge weighted directed graphs, each of which is simply referred to as "graph" henceforth, to model the three scenarios. The graphs have the same vertex and edge sets, represented as V and E, respectively. The vertex set V of the graphs models the blocks/sub-blocks, the inputs, the outputs and the branch points of the circuit. The set E is the collection of the nets of the circuit. The graphs are described below:

- The graph $G_i = \langle V, E, w_i \rangle$ represents the *initial circuit*, which may not satisfy the frequency requirements. The weight $w_i(e)$, $\forall e \in E$ is the number of flip-flops along the wire modeled by e in G_i.

- The graph $G_p = \langle V, E, w_p \rangle$ represents the *wire-pipelined* version of the initial circuit G_i, obtained using some wire pipelining method such as [Coc02, HAT02]. Although G_p satisfies the timing constraints, it may not be functionally equivalent to G_i. The weight $w_p(e),\ \forall e \in E$ is the number of flip-flops along the wire modeled by e in G_p. We assume that the weight $w_p(e)$ is the lower bound on the required number of flip-flops by the wire modeled by e in order to satisfy the target clock period requirements.

- The graph $G_f = \langle V, E, w_f \rangle$ represents the *corrected wire-pipelined* circuit, obtained after altering G_p to make it functionally correct. Hence, G_f satisfies the timing constraints, and is also functionally equivalent to G_i. The weight $w_f(e),\ \forall e \in E$ is the number of flip-flops along the wire modeled by e in G_f.

It can be noticed that the circuit model used is similar to that of retiming formulation [LRS83]. In addition, ρ is the amount of input issue rate slowdown required so that G_f is functionally equivalent to G_i. For the example discussed in section 2.2, the graphs G_i, G_p and G_f model the circuits of Figures 2.1(a), 2.1(b) and 2.2, respectively. This thesis accepts G_i and G_p as inputs and presents a method to obtain G_f and ρ. The input issue rate of G_i is assumed to be 1, i.e., inputs of the initial circuit can be changed every clock cycle.

We extend the weight functions w_i, w_p and w_f to (simple) paths and (simple) cycles of the graphs. The weight of a path/cycle is defined as the sum of weights of all edges on the path/cycle. For the graphs G_i and G_p to have a physical meaning, the edge weights (and consequently path weights) w_i and w_p must be nonnegative, as they represent the number of registers along the wires of the circuits. Moreover, since the graphs represent synchronous systems, every cycle in the graphs must have a strictly positive weight, i.e., at least one edge of each cycle must have a weight greater than zero, in both G_i and G_p. The weights of any edge, path and cycle in G_f cannot be less than the corresponding weights in G_p, as we do not wish to *unpipeline* the wires of G_p. However, the weights

w_p can be less than the corresponding weights w_i in G_i, indicating the presence of more than necessary number of flip-flops required to meet the frequency constraints. Thus, for any edge or path in the graphs, the weight w_f in G_f can be less than[2] the corresponding weight w_i in the graph G_i. To indicate that e is an edge from u and v in the graphs, we will use the notation $u \xrightarrow{e} v$. We will also use the terms "graph" and "circuit" interchangeably.

We have seen in section 2.2 that any attempt to correct the functionality of G_p to obtain G_f may involve the insertion of extra flip-flops, thus resulting an increase in the area. The proposed method also minimizes the area increase due to the insertion of extra flip-flops, which is detailed in section 2.4. To accurately model the area, we define two nonnegative weight functions on E, as shown below:

- The weight $r_p(e), \ \forall e \in E$ represents the number of repeaters along the wire modeled by e in G_p.
- The weight $r_f(e), \ \forall e \in E$ represents the number of repeaters along the wire modeled by e in G_f.

We assume that all repeaters are identical and therefore have equal area. We make a similar assumption for the flip-flops as well, i.e., each flip-flop has equal area. If extra flip-flops are to be inserted along a wire, in going from G_p to G_f, some or all of the repeaters along the wire in G_p can be replaced with flip-flops, without violating any timing constraints. The repeaters of G_i are ignored in our model since they do not have any role in area minimization.

[2]This not true for a cycle though. For any cycle c, $w_f(c) \geq w_i(c)$, since $\rho(c) \geq 1$.

2.4 Solution technique

2.4.1 Obtaining the optimal ρ

As explained in section 2.2, the concept of slowing down the input issue rate (by a factor of ρ) can be used to correct the functionality of a cycle in the wire-pipelined circuit G_p. By restricting inputs to be allowed to change only every ρ clock cycles, we are providing "extra" clock cycles to the cycle in G_p to complete its computations. In other words, slowdown (of the input issue rate) can be thought of as a compensating factor for increased cycle latencies in G_p, at the expense of decreased throughput, since the circuit computes its outputs only every ρ clock cycles.

Let c be any cycle of the graphs, whose latencies in G_i and G_p are given by $w_i(c)$ and $w_p(c)$, respectively. Consider a block on the cycle, and suppose it has an input y, not belonging to the cycle[3]. By the time the output computed by the block propagates back to itself through the other blocks of the cycle, the number of times the signal seen at y may have changed is equal to $w_i(c)$ in G_i, and $w_p(c)$ in G_p. For functional equivalence of the two circuits, the number of input changes seen at y must be identical in both circuits, equal to $w_i(c)$. Since inputs can change every clock cycle in G_i, this can be achieved if the input y is permitted to change only every $\frac{w_p(c)}{w_i(c)}$ clock cycles in G_p. This ratio gives the slowdown $\rho(c)$ required for the cycle c in G_p. Applying this idea to the cycle C of Figure 2.1, where $w_i(C) = 1$ and $w_p(C) = 2$, we have, $\rho(C) = \frac{2}{1} = 2$. However, if $w_i(c)$ does not divide $w_p(c)$, then the weight $w_p(c)$ must be increased to the next higher multiple of $w_i(c)$, as we can only have an integral slowdown. For instance, if the values of $w_i(c)$ and $w_p(c)$ are 2 and 5, respectively, then a slowdown of $\rho(c) = 3$ is required for c in G_p and the weight $w_p(c)$ must be increased to $\rho \cdot w_i(c) = 6$.

In general, a circuit may have more than one cycle and each of these may require a different slowdown. The critical cycle is the cycle which requires the maximum value of

[3]An example of such a situation is illustrated by input y in Figure 2.1.

slowdown. The slowdown required for this cycle is the lower bound for the slowdown required for the entire circuit G_f, since the latencies of other cycles can be increased to match this slowdown. If $\hat{\rho}(G_f)$, or $\hat{\rho}$ in short denotes the minimal (also optimal) slowdown required by G_f to exhibit correct behavior, then we have

$$\hat{\rho} = \max_{c \in \mathcal{C}} \left\{ \left\lceil \frac{w_p(c)}{w_i(c)} \right\rceil \right\}$$

where $\mathcal{C}$ is the set of cycles of the graphs.

The equation shown above represents a *maximum cycle ratio problem* (MCRP) [Law66] on the graphs G_i and G_p, where the time and cost of each edge $e \in E$ is given by the weights $w_p(e)$ and $w_i(e)$, respectively. One method of obtaining $\hat{\rho}$ was proposed by Lawler in [Law66]. The idea is to iteratively apply the Bellman-Ford algorithm [CLR89] to find the longest paths in the graph $G_l = \langle V, E, w_l \rangle$, where the edge weights w_l are defined as:

$$w_l(e) \quad = \quad w_p(e) - \hat{\rho} \cdot w_i(e) \qquad \forall e \in E \tag{2.1}$$

If there is no cycle in G_l ($\mathcal{C} = \emptyset$), then $\hat{\rho}$ is 1, i.e., inputs can be issued every clock cycle in acyclic circuits. Otherwise, a binary search is performed to find the minimal value of $\hat{\rho}$ for which there is no positive cycle in G_l. The presence of a positive cycle in G_l indicates that for some cycle c in G_l, $\hat{\rho} \cdot w_i(c) < w_p(c)$, i.e., the slowdown required for c is greater than $\hat{\rho}$. The complexity of Lawler's method is $O(|V||E|\text{log}(|V|w_{max}))$, where $w_{max} = \max_{e \in E} w_i(e)$. Several other more efficient ways of solving the MCRP have been proposed in the literature [DIG99].

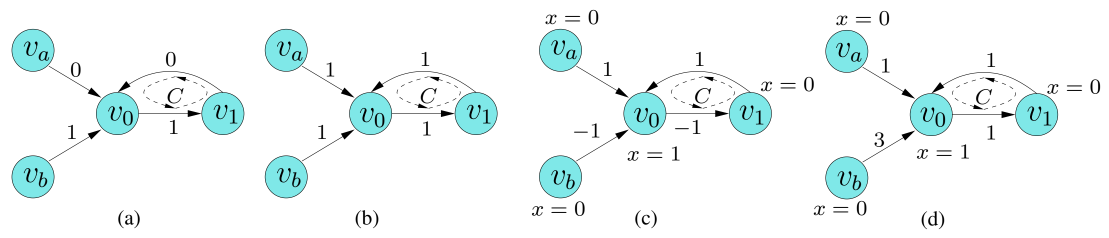


Figure 2.4: Illustration of the solution technique on the circuit shown in Figure 2.1. The numbers shown with the edges in the graphs correspond to the weights of the edges. (a) The initial circuit (G_i) depicting ckt_i. (b) The wire pipelined circuit (G_p), depicting ckt_p. (c) The corresponding graph G_l. The optimal slowdown, $\hat{\rho}$ is 2. The number shown above each vertex in G_l is the x value for that vertex. (d) A Solution (G_f). The weights w_f shown with the edges are obtained by using (2.5).

2.4.2 Obtaining a solution to G_f

A feasible solution

Let q and q' be any two distinct paths from the inputs of the circuits to any vertex $v \in V$. Since the inputs are issued only every $\hat{\rho}$ clock cycles in the circuit G_f, to compensate for the increased cycle latencies, if the difference of weights of q and q' in G_i is k, then the corresponding difference in G_f must be $\hat{\rho} \cdot k$. For example, since the difference of weights of the paths $a \to y$ and $b \to z$ in the circuit G_i shown in Figure 2.1(a) is 1, the corresponding difference must be 2 (since $\hat{\rho}$ = 2 for the circuit) in the circuit G_f shown in Figure 2.2. From this observation, we have

$$\begin{aligned} w_f(q) - w_f(q') &= \hat{\rho} \cdot (w_i(q) - w_i(q')) \\ \Rightarrow w_f(q) - \hat{\rho} \cdot w_i(q) &= w_f(q') - \hat{\rho} \cdot w_i(q') \end{aligned} \tag{2.2}$$

If $\mathcal{Q}_v$ is the set of all paths from the inputs to v in the graphs, then from (2.2), the difference of the terms w_f and $\hat{\rho} \cdot w_i$ must be equal $\forall q \in \mathcal{Q}_v$. We introduce a variable $x(v)$ $\forall v \in V$ such that

$$x(v) \quad = \quad w_f(q) - \hat{\rho} \cdot w_i(q) \qquad \forall q \in \mathcal{Q}_v \tag{2.3}$$

Let q_u be any path starting from the inputs of the circuits, ending at vertex u. For $u \xrightarrow{e} v$, we can form a path q_v ending at v by adding e to q_u. The weights of q_u and q_v can be related as follows:

$$\begin{aligned} w_p(q_v) &= w_p(q_u) + w_p(e) \\ w_i(q_v) &= w_i(q_u) + w_i(e) \\ \text{and} \quad w_f(q_v) &= w_f(q_u) + w_f(e) \end{aligned} \tag{2.4}$$

From (2.3) and (2.4), we have

$$\begin{aligned} w_f(q_v) &= x(v) + \hat{\rho} \cdot w_i(q_v) \\ \Rightarrow w_f(q_u) + w_f(e) &= x(v) + \hat{\rho} \cdot (w_i(q_u) + w_i(e)) \\ \Rightarrow w_f(e) &= x(v) - (w_f(q_u) - \hat{\rho} \cdot w_i(q_u)) + \hat{\rho} \cdot w_i(e) \\ &= (x(v) - x(u)) + \hat{\rho} \cdot w_i(e) \end{aligned} \tag{2.5}$$

In (2.5), the weights w_f are expressed in terms of x values and $\hat{\rho}$. We also have $w_f(e) \geq w_p(e)$ for all $e \in E$. From this and (2.5), the following can be deduced:

$$\begin{aligned} w_p(e) &\leq (x(v) - x(u)) + \hat{\rho} \cdot w_i(e) \\ \Rightarrow x(v) &\geq x(u) + (w_p(e) - \hat{\rho} \cdot w_i(e)) \end{aligned} \tag{2.6}$$

From (2.6), it is evident that $x(v)$ is the weight of the longest path to v in G_l, defined in the previous section, while discussing about Lawler's method of solving MCRP. When there are no positive cycles in G_l, longest paths are well defined and the Bellman-Ford algorithm outputs the x values of the vertices. Therefore, solving the MCRP by

Lawler's method also finds the x values, along with $\hat{\rho}$. The weights w_f can then be determined from the x values using (2.5). To summarize, the following steps are involved in obtaining a functionally correct wire-pipelined circuit G_f.

1. Solve the MCRP to obtain $\hat{\rho}$ and the x values.

2. From the $\hat{\rho}$ and the x values computed in step 1, determine the weights w_f of G_f using (2.5).

Lemma 1 *Let* $(G_f = \langle V, E, w_f \rangle, \rho \geq \hat{\rho})$ *be a solution to* $\langle G_i, G_p \rangle$*. Then for any cycle* c *in the circuit, we have*

$$w_f(c) \quad = \quad \rho \cdot w_i(c)$$

Proof: Suppose cycle c is composed of vertices and edges $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} v_n, v_0 = v_n$. Then

$$\begin{aligned} w_f(c) &= \sum_{i=0}^{n-1} w_f(e_i) \\ &= \sum_{i=0}^{n-1} (x(v_{i+1}) - x(v_i) + \rho \cdot w_i(e_i)) \\ &= \sum_{i=0}^{n-1} \rho \cdot w_i(e_i) + \sum_{i=0}^{n-1} (x(v_{i+1}) - x(v_i)) \\ &= \rho \cdot w_i(c) \end{aligned}$$

Lemma 1 indicates that all cycle latencies are increased by a factor of ρ in G_f. This shows that G_f represents a pipelined version of G_i, retaining its functionality if the inputs are issued only every ρ clock cycles. It computes outputs every ρ clock cycles.

We demonstrate the solution technique on the circuit shown in Figure 2.1. Figures 2.4(a) and (b) show the graph models G_i and G_p, for the circuits ckt_i and ckt_p, shown in Figures 2.1(a) and (b), respectively. The blocks B_0 and B_1, and the inputs a and b are modeled as the vertices v_0, v_1, v_a, v_b, respectively. The graphs have one cycle $C = v_0 \rightarrow v_1 \rightarrow v_0$. We have seen at the beginning of this section that the optimal slowdown required for the circuit is 2, i.e., $\hat{\rho} = 2$. Figure 2.4(c) shows the graph G_l obtained by computing the edge weights using (2.1). For $\hat{\rho} = 2$, it can be observed that the weight of C in G_l is 0, which indicates that the longest paths are well defined in G_l. The x values of the vertices are shown in Figure 2.4(c). The solution obtained by using the x values from Figure 2.4(c) is shown in Figure 2.4(d). It can be seen that the graph G_f of Figure 2.4(d) is identical to the circuit ckt_f of Figure 2.2.

A minimum area solution

The solution technique presented in the previous section only finds a feasible solution, and does not consider minimization of the area increase, incurred due to the possible insertion of extra flip-flops, while obtaining a solution. In this section, we will extend the solution technique to incorporate area minimization and formulate the problem as an Integer Linear Program (ILP) and then describe a method to solve the ILP efficiently. We will consider two flexibilities for area minimization here.

Formulation as an ILP

One way of minimizing the number of extra flip-flops is to retime some or all of the extra flip-flops out of the wires of the circuit, as illustrated in Figure 2.5. In section 2.4.2, the x values are computed as the longest path weights in G_l. However, the slacks in the longest path constraints (henceforth referred to as latency constraints) (2.6) allow a range of permissible values for x. This flexibility enables the movement of flip-flops across vertices, which is exploited for area minimization.

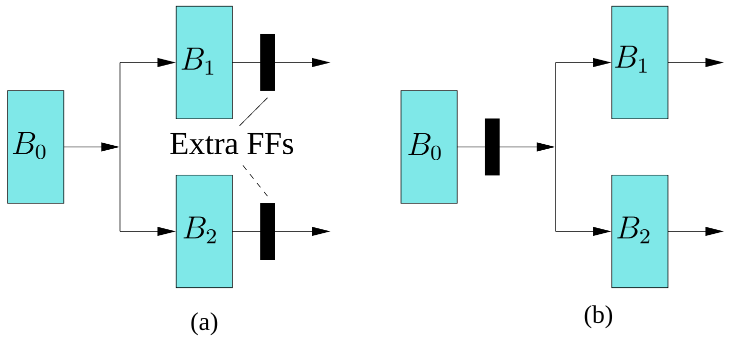


Figure 2.5: Illustration of area minimization on a portion of a circuit by retiming the extra flip-flops. (a) A solution to the problem requires one extra flip-flop each on the outgoing edges of B_1 and B_2, respectively. (b) The two flip-flops are moved over the blocks B_1 and B_2 to the outgoing edge of B_0, which reduces the flip-flop count by one.

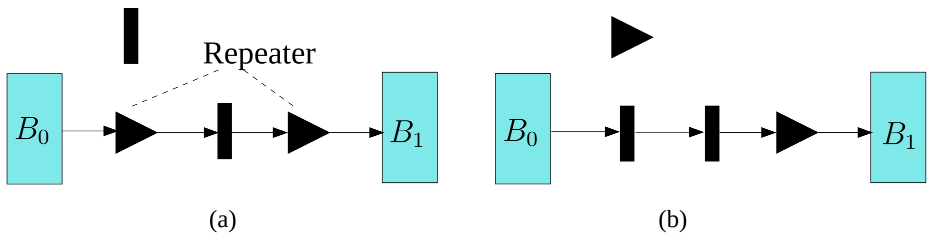


Figure 2.6: Illustration of area minimization on a wire of a circuit by replacing a repeater with the extra flip-flop. (a) A solution to the problem requires one extra flip-flop on the wire between B_0 and B_1. (b) A repeater is replaced with the extra flip-flop.

The second degree of freedom we will explore is as follows. In the event of adding extra flip-flops to the edge e, some or all of the repeaters present along e in G_p can be replaced with flip-flops. We assume that each extra flip-flop can replace one repeater from the edge, as demonstrated in Figure 2.6.

Insertion of an extra flip-flop can be thought as making one of the existing repeaters on the wire "clocked". Inserting an extra flip-flop on a wire eases the timing constraints on the wire, and therefore the wire actually requires lesser number of repeaters. Hence removing a repeater from the wire does not lead to any timing violation. The available number of slots, i.e., repeaters along the edge e in G_p is given by $r_p(e)$ and the number of extra flip-flops to be added along the edge e in G_f is given by $extra(e) = w_f(e) - w_p(e)$. If $extra(e)$ exceeds $r_p(e)$, then all of the $r_p(e)$ repeaters along e in G_p will be replaced with flip-flops. In such a scenario, the number of repeaters along e in G_f, given by $r_f(e)$, will be 0. Otherwise, $r_f(e)$ will be equal to the remaining number of repeaters on e of G_p, after some of them were replaced by extra flip-flops. Therefore, $r_f(e)$ can be expressed as the following piecewise linear (PWL) function:

$$r_f(e) \quad = \quad max\{r_p(e) - (w_f(e) - w_p(e)), 0\} \tag{2.7}$$

We define the area of the edge e in the circuit G_f, $a_f(e)$, as the area of the repeaters and flip-flops along e in G_f. If $area$ is the total area of the repeaters and flip-flops of G_f, and w_a and r_a are the areas of a single flip-flop and repeater, respectively, then for any $\rho \geq \hat{\rho}$,

$$\begin{aligned} a_f(e) &= w_f(e) \cdot w_a + r_f(e) \cdot r_a \quad \forall e \in E \\ &= (x(v) - x(u) + \rho \cdot w_i(e)) \cdot w_a + r_f(e) \cdot r_a \\ \text{and} \quad area &= \sum_{e \in E} a_f(e) \end{aligned} \tag{2.8}$$

Integer Linear Program

The minimum area solution can be formulated as an Integer Linear Program (ILP) shown below, by expressing the PWL function (2.7) as two linear constraints. The constraint set of the ILP includes the latency constraints (2.6) and the two linear repeater constraints (2.9) and (2.10) deduced from (2.7). The objective is to minimize $area$ given by (2.8).

Minimize

$$area = \sum_{e \in E} \{(x(v) - x(u) + \rho \cdot w_i(e)) \cdot w_a + r_f(e) \cdot r_a\}$$

Subject to

$$\forall e \in E \qquad s.t. \quad u \xrightarrow{e} v$$

$$r_f(e) \geq r_p(e) + w_p(e) - (x(v) - x(u) + \rho \cdot w_i(e)) \qquad (2.9)$$

$$r_f(e) \geq 0 \qquad (2.10)$$

$$x(v) \geq x(u) + w_p(e) - \rho \cdot w_i(e)$$

Solving the ILP

Solving an ILP is generally *NP–complete*, unless the problem exhibits integral polytope structure [BJS90], which means that all of the extremal points of the polytope formed by the constraint set of the ILP have purely integral components. Unfortunately, the ILP of previous section in the described form does not have an integral polytope, and is therefore hard to solve. The hardness of the problem comes from the repeater constraints (2.9) and (2.10). In this section, we will reformulate the ILP as an instance of the dual of the Minimum Cost Network Flow (MCF) problem [BJS90], which exhibits integral polytope structure, and therefore can be efficiently solved. This is accomplished by finding a closed form expression for the repeater count r_f, which can be used to eliminate repeater constraints from the ILP.

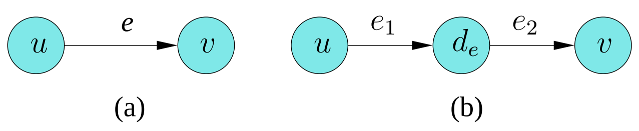


Figure 2.7: Insertion of a dummy node d_e on an edge $e \in E$.

We use the following transformation to achieve this. For each edge $e \in E$, where $u \xrightarrow{e} v$, a dummy vertex d_e is added to split the edge into two edges, e_1 and e_2, such that

$u \xrightarrow{e_1} d_e$ and $d_e \xrightarrow{e_2} v$, as shown in Figure 2.7. The edge e_1 models the case where the extra flip-flops to be inserted on e replace the repeaters of e. Inserting a flip-flop on e_1 increases the area of e by $w_a - r_a$. The edge e_2 models the case where more than $r_p(e)$ extra flip-flops are to be inserted on e. Inserting an extra flip-flop on e_2 increases the area of e by w_a. To minimize area, the ILP fills e_1 first before assigning any flip-flop to e_2. The first $r_p(e)$ extra flip-flops to be inserted on e are assigned to e_1 and the rest are inserted on e_2. Therefore, the number of flip-flops inserted on e_2, given by $w_f(e_2)$, will be strictly positive only when the number of extra flip-flops exceeds $r_p(e)$. From this, We have,

$$\begin{aligned} w_f(e) &= w_f(e_1) + w_f(e_2) \\ w_f(e_1) &\leq r_p(e) + w_p(e) \qquad (2.11) \\ r_f(e) &= r_p(e) - (w_f(e_1) - w_p(e)) \qquad (2.12) \end{aligned}$$

Equation (2.12) represents a closed form expression for the repeater count r_f in G_f, which can be used to eliminate the r_f variables from the ILP. In addition, the following latency constraints on e_1 and e_2 can be inferred from the above equations.

$$\begin{aligned} &x(d_e) \geq x(u) + (w_p(e) - \rho \cdot w_i(e)) && (w_f(e_1) \geq w_p(e)) \\ &x(v) \geq x(d_e) && (w_f(e_2) \geq 0) \\ &x(d_e) \leq x(u) + (r_p(e) + w_p(e) - \rho \cdot w_i(e)) && (\text{from (2.11)}) \end{aligned}$$

It can be observed that the first two inequalities above sum up to obtain the constraint (2.6) on e. The last constraint places an upper bound of $r_p(e)$ on the number of extra flip-flops that can be inserted on e_1 part of the edge e. With all the above equations, we obtain a new expression for $area$, as shown below:

$$
\begin{aligned}
a_f(e) &= w_f(e) \cdot w_a + (r_p(e) + w_p(e) - w_f(e_1)) \cdot r_a \\
&= (x(v) - x(u)) \cdot w_a - (x(d_e) - x(u)) \cdot r_a + \rho \cdot const. \\
&= x(v) \cdot w_a - x(u) \cdot (w_a - r_a) - x(d_e) \cdot r_a + \rho \cdot const. \\
area &= \sum_{v \in V \bigcup V_d} (k_v \cdot x(v)) + \rho \cdot const. \qquad (2.13)
\end{aligned}
$$

where V_d is the set of dummy vertices, and if $FO(v)$ and $FI(v)$ are the number of outputs and inputs of $v \in V$, respectively,

$$
k_v = \begin{cases} FI(v) \cdot w_a - FO(v) \cdot (w_a - r_a) & : \quad v \in V \\ -r_a & : \quad v \in V_d \end{cases}
$$

Equation (2.13) indicates that $area$ is a linear function of x variables and the slow-down factor ρ. The reformulated ILP for the minimum area solution to G_f is shown below.

$$
\begin{aligned}
\text{Minimize} \quad area &= \sum_{v \in V \bigcup V_d} (k_v \cdot x(v)) + \rho \cdot const \\
\forall e \in E \quad & \quad s.t. \quad u \xrightarrow{e} v \\
x(u) - x(d_e) &\leq \rho \cdot w_i(e) - w_p(e) \\
x(d_e) - x(v) &\leq 0 \\
x(d_e) - x(u) &\leq r_p(e) + w_p(e) - \rho \cdot w_i(e)
\end{aligned}
$$

For a constant ρ, the constraint set of the preceding ILP is a set of difference constraints involving x variables, and the objective is a linear function of x variables. An ILP of this structure represents an instance of the dual of the minimum cost flow problem, which can be efficiently solved by several methods such as the network simplex method [BJS90]. As before, the weights w_f can be computed using (2.5). In addition,

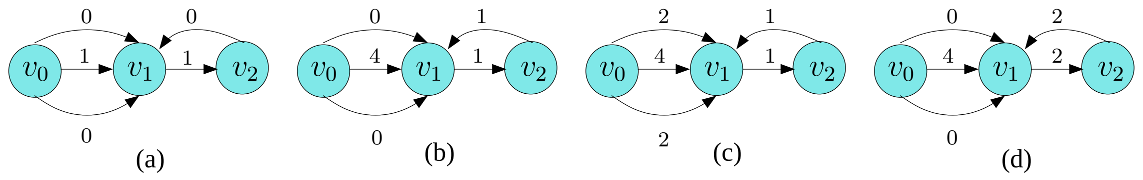


Figure 2.8: Optimal $\hat{\rho}$ may not mean minimum area. It is assumed that the circuits do not have repeaters. The number shown with each edge in the graphs denotes the flip-flop count of the edge. (a) Initial circuit. (b) Wire-pipelined circuit. (c) A minimum area solution for $\rho = \hat{\rho} = 2$: number of flip-flops = 10. (d) A minimum area solution for $\rho = 4$: number of flip-flops = 8.

there is a minimum area solution for each value of $\rho \geq \hat{\rho}$. Moreover, the minimum area solution for $\hat{\rho}$ may not be a global minimum solution, as demonstrated in Figure 2.8. However, in most cases, maximizing throughput (or minimizing ρ) is the primary objective, rather than minimizing area. In such a scenario, the ILP is solved for $\rho = \hat{\rho}$, which is determined by solving the MCRP, as detailed in section 2.4. The resultant solution represents a maximal throughput minimum area solution to G_f.

2.5 Experimental Results

2.5.1 Set up

The ideal application of the proposed technique is in the area of System on Chip design methodology, where several IP blocks are connected by long across-chip interconnects. However, the lack of appropriate SoC benchmarks makes experimentation a difficult task. The authors of the latency insensitive design methodologies of [CMSV01, CM04a] use a small MPEG circuit for their experimentations. In contrast, the wire retiming approaches proposed in [LZ03, CYTD03], the architectural retiming technique of [TTBN00] that were described in Section 2.1, a more recent work on concurrent systems [JCK06] address this issue by projecting the gate level circuits, specifically those of the ISCAS benchmark suite [BBK89], as large SoC circuits, where, in most cases, each logic gate is treated as a large IP block, and with an input to output latency of one in [CMSV01, CM04a].

We use ISCAS benchmarks for our experimentation, just as is done in [LZ03,CYTD03, TTBN00, JCK06], where each circuit is scaled into a large SoC circuit consisting of IP blocks. The results are presented in Section 2.5.2.

In addition, we also present another potential application of the proposed solution technique at the circuit- or logic- level for frequency constrained circuits. Specifically, for designs that have a strict frequency constraint, one solution is to pipeline the circuit, i.e., increase the latencies of the paths that violate the clock period constraint. This process may be different from the SoC problem discussed in the previous paragraphs, since the wires of the circuit can have small delays and the overall logic must be considered during pipelining, unlike individual wires in the SoC scenario. We use the same ISCAS benchmark circuits for this purpose and demonstrate the results in Section 2.5.3.

2.5.2 Wire-pipelined SoC circuits

An operating frequency of 3GHz is chosen for the system and the target technology chosen has a feature size of 65nm. After finding a placement using Capo [CKM], the area of the circuits was scaled to 4cm^2 to mimic the layout of a realistic chip. For smaller layouts, the wire lengths are not long enough to be pipelined. The dimensions of the circuits were scaled accordingly. Each gate in the original circuit is assumed to be a combinational functional block. The block propagation delays are randomly generated using a quantitative scale of 1–10, where 10 corresponds to the system clock period, which turns out to be 0.33ns (corresponding to a frequency of 3GHz). In addition, the output signal of each block is assumed to be latched immediately, after it leaves the block, and these flip-flops are the only memory elements in the (initial) circuit, indicating that each wire has a latency of 1, similar to the approach used in [CMSV01],

Circuit	$\|V\|$	$\|E\|$	G_p		G_f		$\hat{\rho}$	**Time**
			Rptrs	**Flops**	**Rptrs**	**Flops**		**(sec)**
s27	15	18	21	19	18	22	1	0.1
s344	110	210	265	229	198	333	2	0.1
s349	114	215	238	227	174	329	2	0.1
s1196	360	836	1864	1111	1606	1446	1	1
s1238	389	925	1519	963	1267	1277	1	1
s1423	449	913	1134	1028	752	1540	2	1
s1494	364	1104	3083	1592	2572	2231	2	1
s13207	2014	3759	4843	4094	3354	5976	2	1
s15850	3504	7215	9166	7787	6231	11382	2	2
s38417	8029	17646	29717	0947	3131	29837	2	14
s38584	9616	22515	35777	6093	6628	37446	2	24

Table 2.1: Experimental results for ISCAS benchmarks.

where each IP block has a latency of one. In this way, each wire, along with the delay of the block that it feeds data to, can be considered for pipelining independently without addressing the other parts of the circuit.

For the wire delays, the projections for a 2cm global wire made in [Con01] were used, where the delay of an optimized 2cm wire in 70nm technology is projected to be 0.67ns. The delays of the wires of the test circuits were determined by assuming a linear relationship between the delay of a wire and its length, which is reasonable for buffered interconnects. It is also assumed that a 2cm wire has 10 repeaters, and accordingly the repeater counts of the wires of the circuit were determined. Finally, the area of a flip-flop was assumed to be twice that of a repeater.

First, the optimal slowdown, $\hat{\rho}$ was obtained for each circuit by solving the MCRP, as explained in section 2.4.1. Later, the ILP, which is an instance of the dual of minimum

Circuit	**MaxFreq (GHz)**	**S**$_{G_p}$	$\hat{\rho}$	**S**$_{G_f}$
s27	2.48	1.21	1	1.21
s344	1.86	1.61	2	0.80
s349	2.13	1.42	2	0.71
s1196	1.42	2.11	1	2.11
s1238	1.35	2.20	1	2.20
s1423	1.24	2.42	2	1.21
s1494	1.49	2.01	2	1.01
s13207	1.10	2.73	2	1.36
s15850	1.57	1.92	2	0.96
s38417	1.03	2.89	2	1.45
s38584	1.19	2.51	2	1.26

Table 2.2: Performance issues with wire-pipelining.

cost network flow problem, was solved using the network simplex implementation of [Loe] to obtain a minimum area solution subject to the slowdown of $\hat{\rho}$ obtained for each circuit. The experiments were performed on a 2.4GHz Pentium 4 machine with 1GB RAM. The results obtained for different benchmarks are shown in Table 2.1. The labels **Rptrs** and **Flops** denote the number of repeaters and flip-flops, respectively, listed for both circuits G_p and G_f. It can be observed from the table that the number of repeaters, **Rptrs**, decreases in G_f, since some of the repeaters in G_p are replaced by flip-flops in G_f. In addition, for circuits such as s1238 and s1196, a slowdown of 1 indicates that none of the wires forming cycles in those circuits were long enough to be pipelined. The run times are in the order of a few seconds, as shown in the table.

Table 2.2 captures the speedup obtained by wire-pipelining. In particular, we compare the results with those obtained by utilizing conventional retiming that also considers wire delays. Our (wire) retiming implementation is based on the algorithm proposed in [LZ03]. Column 2 of Table 2.2 lists the upper bound on the operating frequency, achieved by retiming, of G_i for each benchmark. The column labeled S_{G_p} shows the frequency speedup obtained by performing wire-pipelining on G_i for a clock frequency of 3GHz. However, the frequency speedup of the wire-pipelined circuit, G_p (which may be functionally incorrect) may not entirely translate into the throughput speedup obtained for the corrected wire-pipelined circuit, G_f, since the possibility of increased cycle latencies in G_p will enforce a slowdown of $\hat{\rho}$ in the input issue rate in G_f. The column S_{G_f} shows the actual throughput speedup achieved by G_f, where $\mathrm{S}_{G_f} = \mathrm{S}_{G_p}/\hat{\rho}$.

It can be observed from Table 2.2 that for most circuits, the actual speedup achieved is greater than one, as compared to retiming, which indicates that wire-pipelining has indeed improved the performance. However, some circuits such as s344, the throughput speedup achieved is less than one, suggesting that wire-pipelining has resulted in throughput degradation for these circuits.

Table 2.3 depicts the area and power consumption issues associated with wire-pipelining. The column labeled **Area Incr** lists the percentage increase in the area of the repeaters and flip-flops in G_f. The area is calculated as the sum of the areas of the flip-flops and repeaters, which are normalized to 2 and 1, respectively. The last two columns demonstrate the dynamic power consumption[4]of repeaters and flip-flops, obtained using SIS [SSL$^+$92], in circuits G_p and G_f, respectively. It can be observed that the repeater-FF dynamic power shows the same trends as the actual speedup, $_{G_f}$ of Table 2.2, as depicted by the final column, which shows the ratios of S_{G_f} and G_f repeater-FF dynamic power for each of the benchmarks. We are unable to generate results for some of the large benchmarks (as SIS was not able to handle large input sizes), and this is

[4]Normalized to that of the initial circuit, G_i. For G_i, the maximum frequency bounds shown in Table 2.2 are assumed, while a frequency of 3GHz is used in power computations for G_p and G_f. The slowdown factor is taken care by scaling the node switching activity values by the corresponding amount of $\hat{\rho}$.

Circuit	**Area Incr**	Power		$\frac{\mathbf{S}_{G_f}}{\mathbf{Pow}}$
	(%)	G_p	G_f	
s27	5.1	1.27	1.41	0.86
s344	19.5	0.86	1.21	0.66
s349	20.2	0.73	1.02	0.70
s1196	10.1	2.61	3.24	0.65
s1238	10.9	2.67	3.37	0.66
s1423	20.1	1.30	1.85	0.65
s1494	12.2	1.34	1.79	0.56
s13207	17.5	1.45	2.04	0.67
s15850	17.2	1.01	1.42	0.68
s38417	15.6	1.63	-	-
s38584	15.4	-	-	-

Table 2.3: Repeater area and dynamic power.

indicated by the "-" entries in the table.

Although wire-pipelining causes a degradation in performance for some circuits, there could be several system-wide reasons for having a higher clock frequency. Typically, decision on the frequency is made at the system level and is handed down to the designer to implement, who tries to ensure best possible performance under this decision. The amount of slowdown required can be reduced by using better objective functions in placement, which will attempt to place blocks which form, in particular, the critical cycle closer to each other. The authors of [CM04b] try this idea at the floorplanning level by including the slowdown factor as part of the floorplanning objective.

2.5.3 Frequency constrained logic circuits

The previous section assumes that each gate of the ISCAS benchmarks as a logic block and projects the area to 4cm^2. In this section, we directly use the gate-level description of the benchmarks to pipeline the circuits for a target frequency. For this purpose, we use a SPICE generated library consisting of five logic gates, namely, an inverter, two- and three-input nand and nor gates, and an edge-triggered flip-flop. The gates along with the delays are shown in Table 2.4. After mapping the benchmarks using SIS [SSL$^+$92], we employ Capo to find a placement of the circuits.

The next step is to pipeline the circuits for the target clock period. We choose the same frequency of 3GHz that is used in the previous section. As mentioned in Section 2.5.1, the pipelining strategy must consider the whole circuit rather than on individual wires separately as done for the SoC circuits in Section 2.5.2. For circuits that do not not cycles, it is fairly straight forward to pipeline, since a simple breadth first search or a topological sort will suffice. Furthermore, the additional latencies in this scenario affect only the input (PI) to output (PO) latencies and do not impact the throughput of the circuit.

Gate	**Delay (ps)**
NOT	29.0
NAND2	33.6
NAND3	36.0
NOR2	34.5
NOR3	37.4
DFF	67.2

Table 2.4: Gate library and delays.

In contrast, for circuits that have cycles, pipelining is much more complicated, since a topological ordering of the gates cannot be defined. One approach is to remove all the *back edges* from the circuit and then pipeline the resultant acyclic circuit. The deleted back edges are then added and any timing violations can be handled by inserting flip-flops on the added edges. These back edges can be, for instance, all wires that have flip-flops, i.e., edges with a positive weight (w_i) in the initial circuit. However, adding memory elements on cycles of a circuit reduces its throughput and such a strategy can result in ρ suboptimal pipelining solutions, for it can insert additional flip-flops on cycles even when not required.

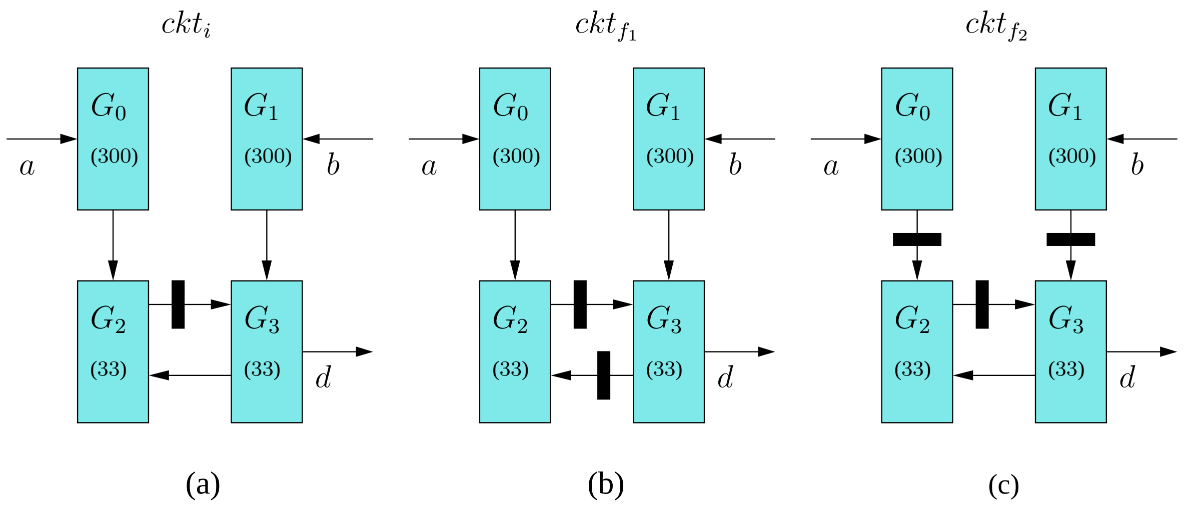


Figure 2.9: A circuit with four gates, two inputs a and b, an output d and a cycle G_2-G_3. The numbers shown in the parentheses are the delays, in picoseconds, of the blocks, and suppose the clock period constraint is 333ps. (a) The circuit before pipelining wires (ckt_i), the minimum clock period = 366ps. (b) The pipelined circuit obtained (ckt_{f_1}), when the back edge $G_2 \xrightarrow{e} G_3$ is removed and then added after pipelining the rest of ckt_i, the minimum clock period is 333ps. For this circuit, the throughput slowdown = 2. (c) A better solution (ckt_{f_2}) that has no reduction in the throughput. In addition, the minimum clock period for this circuit is 300ps.

For instance, consider the circuit ckt_i with four gates (depicted as large rectangles) shown in Figure 2.9(a) that has a cycle involving the gates G_2 and G_3 with a latency of one. For simplicity, we ignore wire and flip-flop delays from the calculations. It can be seen that the minimum clock period for this circuit is 366 units. Suppose there is a maximum constraint of, say, 333 units (corresponding to 3GHz if the units are picoseconds) on the clock period. The approach detailed in the previous paragraph can be executed as follows. First, the back edge $G_2 \xrightarrow{e} G_3$ is removed and the remainder of the circuit is pipelined using a topological traversal. In the process, a flip-flop needs to be inserted on the wire $G_3 \xrightarrow{e'} G_2$. When e is added later, the flip-flop must be retained as the path $G_0 \rightarrow G_1 \rightarrow G_2$ has a delay of 366 units, and this exceeds the target clock period. The resultant circuit with a ρ of two, is shown as ckt_{f_1} in Figure 2.9(b). This certainly is not an ideal solution, since it results in an overall loss of performance, and ckt_{f_2} of Figure 2.9(c) is a better solution that transfers the insertion of additional flip-flops to the non-cyclic wires of the circuit and ensures that there is no slowdown, i.e., $\rho = 1$.

Therefore a better strategy may to be to minimize the amount of pipelining required on cycles. To this purpose, we use an approach that removes non-cyclic wires from the delay calculations. The sequence of the steps is shown below:

- The cyclic wires, i.e., all of the wires that are part of the cycles of the circuit are identified and this can be done using an all pairs path algorithm such as Floyd-Warshall [CLR89] or multiple iterations of depth first search.

- All of the remaining, i.e., noncyclic wires, along with the cyclic wires that have flip-flops are removed from the circuit. These cyclic wires can be treated as back edges removed to break the cycles.

- The circuit is pipelined for the target clock period using topological ordering.

- The removed edges are then added, and are pipelined (flip-flops are inserted) if they result in any timing violations.

Circuit	$\|V\|$	$\|E\|$	**MaxFreq (GHz)**	$\mathbf{S}_{G_p}$	$\hat{\rho}$	$\mathbf{S}_{G_f}$
s13207	2708	4648	0.86	3.47	4	0.87
s15850	4566	8289	0.75	3.96	5	0.79
s38584	13669	25908	0.79	3.78	4	0.95
b14	9970	19100	1.26	2.37	5	0.48
b15	10010	18852	0.45	6.60	17	0.39
b20	19979	38484	0.89	3.36	8	0.42

Table 2.5: Performance comparison of pipelining the circuits with retiming.

Most of the ISCAS benchmarks that are used in Section 2.5.2 have very low gate counts and do not offer much scope for pipelining. We apply the approach on a few large benchmarks, namely, s13207, s15850 and s38584. In addition to these, we use a few circuits from the ITC benchmark suite [CRS00], b14, b15 and b20, for experimentation.

Table 2.5 presents the results of the proposed correction technique and a comparison with the performance achieved with retiming. The labels of the columns have the same meaning as those of Tables 2.1 and 2.2. For instance, the column labeled **MaxFreq** denotes the maximum frequencies obtained for the original, unpipelined, circuits with retiming.

The last column, labeled $\mathbf{S}_{G_f}$, indicates that pipelining the circuits does not result in overall performance improvement. One reason for this is that, although the pipelining strategy explained earlier in the section eliminates the noncyclic wires from the delay computations, it is still possible that the latencies of some cycles are increase even when not required since the cycles may not be independent and can contain many common wires. The problem is more pronounced in the ITC benchmarks, where there are very few noncyclic wires. There is need for a better, more optimal, pipelining strategy to handle this issue.

2.6 Related concepts

2.6.1 G_f and ρ-slowing

The notion of $slowdown$ used in our thesis can be related to the idea of ρ-*slowing* defined in [LRS83, LS83][5] to establish techniques to transform a synchronous circuit into a functionally-equivalent systolic circuit, i.e., a circuit where each of its functional units have unit delay, there is at least one flip-flop along each of its interconnections. If a circuit, which consists of unit-delay functional units, has a maximum combinational path delay of ρ units, then a corresponding systolic circuit can be constructed as follows. First, each flip-flop of the circuit is replaced by a sequence of ρ flip-flops to produce a ρ-*slow* functional equivalent circuit, i.e., it computes valid outputs only every ρ clock cycles. Next, retiming is performed on the ρ-*slow* circuit to reduce the clock period. This process can be extended to a general circuit with unequal functional unit delays. In such a case, the objective may not necessarily be to obtain a systolic circuit, but to decrease the clock period by ρ-*slowing* and retiming, at the expense of decreased throughput.

The circuit G_f can be related to $G_{\hat{\rho}\cdot i} = \langle V, E, \hat{\rho} \cdot w_i \rangle$, a $\hat{\rho}$-*slow* version of the initial circuit G_i obtained by replacing each flip-flop in G_i with a sequence of $\hat{\rho}$ flip-flops, where $\hat{\rho}$ is the optimal slowdown as calculated in section 2.4. A. Like $G_{\hat{\rho}\cdot i}$ (or any of its retimed configurations), any feasible solution to G_f is a $\hat{\rho}$-*slow* version of G_i. In either circuits, G_f and $G_{\hat{\rho}\cdot i}$, the cycle latencies are scaled by a factor of $\hat{\rho}$. While the same is valid for input-output path latencies as well in $G_{\hat{\rho}\cdot i}$, this may not be true for input-output path latencies in G_f, since the path latencies in G_f are dictated by the lower bounds (weights w_p in G_p) on the number of flip-flops required to pipeline the wires of the paths. For instance, a purely combinational input-output path, say q, in G_i will also be purely combinational in $G_{\hat{\rho}\cdot i}$, and if there is a strictly positive lower bound on the number of registers required to pipeline one of the wires, say e, on q, i.e., $w_p(e) > 0$,

[5]In [LRS83, LS83], this concept is actually called *c-slowing*.

then the latency of q must be at least that much in any feasible solution to G_f. Even for nonzero weighted paths in G_i, the scaled weight in $G_{\hat{\rho}\cdot i}$ may be less than the lower bound, i.e., $\hat{\rho} \cdot w_i(q) < w_p(q)$. Such a situation can arise since the scaling factor $\hat{\rho}$ is solely determined by the cycles of the circuit, and increasing the latency of the path q in G_i by a factor of $\hat{\rho}$ may not be sufficient to obtain a wire-pipelining solution for the wires of q. On the other hand, for some paths, the lower bound may be less than its weight in $G_{\hat{\rho}\cdot i}$, and in such cases, the redundant flip-flops can be removed from the circuit, thus reducing the area of the circuit.

Consider any output vertex v and let q_v be any input-output path ending at v. A close look at the expression $x(v) = w_f(q_v) - \hat{\rho} \cdot w_i(q_v)$, defined in section 2.4, indicates that $x(v)$ represents the amount by which the latency of q_v (or any input-output path ending at v) in $G_{\hat{\rho}\cdot i}$ is altered in the corresponding G_f, where the weights w_f are computed using (2.5). A positive $x(v)$ implies that the weight of q_v in G_f exceeds that of q_v in $G_{\hat{\rho}\cdot i}$. Likewise, if $x(v) < 0$, the latency of q_v in G_f is less than the corresponding latency in $G_{\hat{\rho}\cdot i}$ by $|x(v)|$. In G_f, the edge and path latencies are constrained by the inequalities (2.6), which represent lower bounds on the register count on the wires of the circuit G_f. From these observations, the solution approach discussed in section 2.4 can be thought of a way of altering the input-output latencies, while retaining the cycle latencies, of $G_{\hat{\rho}\cdot i}$ such that the weight of each wire exceeds the given lower bound. Such line of thinking is analogous to finding a feasible solution to G_f by applying retiming on $G_{\hat{\rho}\cdot i}$ with the following framework.

- *Variables:* The retiming variable for each $v \in V$ is $x(v)$.

- *Constraints:* The retiming constraint graph consists of the lower bound constraints (2.6), which model both the nonnegativity and clock period constraints of the traditional retiming formulation. The inequalities (2.6) can be treated as a special case of nonnegativity constraints, where the lower bounds on some or all of the edge weights in the retimed circuit are strictly positive. In addition, they also represent the minimum number of flip-flops required on the wires of the retimed circuit to meet the target clock period requirements.

- *Input-output path latencies:* In conventional retiming, a host vertex is introduced into the circuit such that it has incoming edges from the outputs and outgoing edges from the inputs of the circuit. Therefore, each input-output path forms a cycle along with the host vertex, thus ensuring that the latency is retained in the retimed circuit, since retiming preserves cycle latencies. In order to permit changes in input-output path latencies, the host vertex cycles must be broken, and this can be done by removing, from the circuit, the incoming edges to the host vertex from the outputs of the circuit. In such a scenario, the change in the latency of an input-output path ending at $v \in V$ is equal to $x(v) - x(host)$ and, if $x(host) = 0$, this evaluates to $x(v)$.

In theory, this approach appears to be a better strategy for pipelining the logic of a circuit, than the techniques presented in Sections 2.5.2 and 2.5.3. Furthermore, the solution through *ρ-slowing* is correct by construction and does not require any correction techniques addressed in this chapter. However, the primary issue with retiming based implementations is the assumption of a simple model for the delays, such as the Elmore delay model. While such approaches may work well for an optimization problem, where relative accuracy will suffice, they cannot be used for finding a solution that strictly adheres to a given frequency constraint. A viable strategy may be to insert flip-flops wherever timing violations occur, which is where our proposed solution technique arrives into the picture, to correct the functionality. However, as noted in Section 2.5.3, the challenge is to find a throughput-optimal pipelining solution.

Another motivation behind *ρ-slowing* in [LRS83], besides reducing the clock period, was to simultaneously process ρ input streams by properly multiplexing and de-multiplexing the I/O ports of the *ρ-slow* circuit. Since the circuits G_f and $G_{\hat{\rho} \cdot i}$ are functionally equivalent, this advantage can also be extended to G_f, i.e., $\hat{\rho}$ input streams can be processed to hide the slowdown, which leads to an overall throughput speedup equal to the frequency speedup obtained by wire-pipelining the circuit G_i.

2.6.2 Software implementation of *slowdown*

For single input stream data, there is an alternative way of implementing slowdown if the circuit G_i represents a microprocessor architecture, instead of an interconnection of circuit blocks as assumed earlier. In microprocessors, the input presented is in the form of a sequence of assembly instructions, and in the simplest case, the microprocessor accepts and executes a single instruction every clock cycle. In this context, a slowdown of $\hat{\rho}$ can be thought as a pause of $\hat{\rho}$ clock cycles between successive instruction executions, and this can be realized by inserting $\hat{\rho}$ NOPs, i.e., instructions which do not implement any function, after every instruction in the assembly code.

2.7 Conclusion

This chapter has presented an approach to solve the problems created by wire-pipelining. The proposed method also finds the optimal value of input issue rate slowdown required for a circuit, since it directly affects the throughput of the circuit. The problem is formulated as an instance of the dual of minimum cost flow problem, to incorporate the minimization of area increase, incurred due to the insertion of extra flip-flops in the process of obtaining a solution. Though wire-pipelining improves overall throughput of most circuits, it may degrade the throughput for some circuits. However, this is still a useful solution since clock frequencies are typically decided by system-wide considerations, and the task of the designer is to obtain the best achievable performance under such system-level constraints. In addition, the throughput can be improved by choosing better objective functions. Finally, there is need for further research on minimizing the overhead in the repeater area and power consumption, due to wire-pipelining, an important concern which must not be ignored.

Chapter 3

Microarchitecture-aware floorplanning preliminaries

This chapter presents some background information about superscalar processors and techniques that are used for the floorplanning methodologies dealt in the subsequent chapters.

3.1 Superscalar microprocessors

The hardware of microprocessors typically consists of the fetch and decode logic, execution core, memory, and writeback/retire logic. In pipelined processors, the logic is distributed into multiple stages, where each stage implements a particular functionality, to ensure high performance by keeping the clock period short. A superscalar processor [HP96] implements a pipelined architecture that can execute multiple operations per cycle. Instructions are pre-fetched and stored, and are executed when the corresponding resources or functional units become available.

The main attributes of a superscalar processor are branch prediction and instruction scheduling. Branch prediction allows the processor to fetch and execute instructions that follow a branch operation, before the result of the branch is known. If the prediction turns out to be correct, then the execution continues, otherwise all of the instructions fetched and executed in the incorrect path are squashed from the processor pipeline. Instruction scheduling, on the other hand, relates to the techniques employed in dispatching multiple instructions per clock cycle to the execution units. Such schemes can be broadly classified into two categories: dynamic and static. Most architectures such as the Pentium processor [HP96] employ dynamic scheduling, where instructions can be issued and executed “out of order”. Specifically, an instruction can be dispatched to the execution unit when all control and data dependencies, such as a Read After Write (RAW) hazard, associated with the instruction are resolved, and not necessarily in program order. This capability can significantly increase instruction level parallelism (ILP) through efficient resource utilization, which reduces the CPI of the processor, thereby decreasing the execution time T_{exec} of (1.1). Although the instructions can execute and complete out of order, architecture state must be updated in program order, i.e., the instructions must be retired in the order they are fetched from memory.

In contrast, static scheduling handles instruction issue in program order. The advantage is that this technique requires less hardware than dynamic scheduling. However, such a scheme can only be effective when combined with techniques such as Very Long Instruction Word [HP96], where the optimization is handled mostly at the compiler level. The Itanium processor [MLH$^+$00], which uses the VLIW format, is an example of an architecture that employs static instruction scheduling. Although the instructions are scheduled in program order, operations can complete execution out of order. The retire logic must be able to handle such a scenario.

3.2 Superscalar architecture simulation

Software modeling of microarchitectures provides an effective way of validating architecture changes and performance/CPI estimation. Processor simulation has been a major area of research and several techniques have been proposed and implemented in the past decade. Examples include those developed in the public domain, such as SimpleScalar [BA97] and industry simulators such as Asim [EAB$^+$02] by Intel and Turandot [EAB$^+$02] by IBM. The primary components of these simulators are the timing and functional models. The timing model implements an event driven engine that simulates each clock cycle of programs executed on the architecture modeled by the simulator. In contrast, the functional model maintains the microarchitecture state, such as the contents of the register file, and actually executes the simulated instructions on the host machine where the simulator is installed.

We use SimpleScalar, which simulates the DLX superscalar architecture [HP97], for most of the work presented in this thesis, specifically for the simulation strategies presented in Chapters 4 and 6. The simulator is widely used for microarchitecture research, partly due to availability of the source code, since it is developed in the public domain. Furthermore, it models all the important features of superscalar architectures such as out-of-order execution. Another feature that is useful to evaluate different microarchitectures is that the architecture configuration, such as cache size and fetch width, is parameterized. In addition, for the simulation methodology of Chapter 5, we utilize Asim, an industry simulator that models the Pentium architecture (P6) [Int98].

3.3 Benchmarks

An important aspect of performance estimation is the set of benchmark programs that need to be used for the simulations. The programs must depict real life work load scenarios seen for the simulated architecture, in order to attain an effective performance characterization. Several work loads have been developed in the last few years for different applications. The most widely used are the programs of the SPEC 2000 benchmark suite [Hen00], which consists of work loads based on programs executed on general purpose microprocessors, such as gcc and gzip. The benchmarks of the MediaBench suite [LPMS97] mostly depict multimedia applications, while those of TPC-C [Tra97] involve programs at transaction level. In this thesis, we use SPEC benchmarks for the purpose of validating the proposed floorplanning methodologies.

3.4 Architecture power estimation

3.4.1 Sources of power dissipation

There are three major components in the power dissipated by CMOS circuits:

- **Dynamic power:** Also known as active power, this component corresponds to the power consumed during the charging and discharging of circuit capacitances. The magnitude of the dynamic power is proportional to the switching frequencies of the devices.

- **Leakage power:** This component is due to the presence of subthreshold currents and, more recently, gate oxide tunneling currents. Leakage power are inversely proportional to transistor threshold voltages, and has been gaining importance due to the scaling of the threshold voltages and increasing chip temperatures, because of the exponential dependence of subthreshold current on operating temperature.

- **Short circuit power:** This component is due to the presence of a short circuit between the supply voltage and the ground during the time of input switching. In general, short circuit power tends to have a small magnitude if the input switching times are controlled, and we do not consider this component for the work addressed in this thesis, specifically the thermally-aware floorplanning approach of Chapter 6.

3.4.2 Architectural power estimation

Microarchitecture optimizations typically focus on power estimation at the block-level. The event-driven/cycle-accurate model of microarchitecture simulations mentioned in Section 3.2 provides an effective framework for determining the block switching activities required to estimate, particularly, the dynamic component of the power.

The leakage power is generally independent of the activity levels. Such a scheme requires a block-level power characterization that can be constructed using circuit-level simulations.

A number of frameworks have been proposed for architectural power estimation in the public domain. Most of them incorporate power models into existing cycle-accurate simulators, as described in the previous paragraph. For instance, Wattch [BTM00], which we use in this thesis for simulations, and SimplePower [YVKI00] extend SimpleScalar to add the capability of power estimation.

3.5 Experimental design

Experimental design involves of determining a set of experiments, which is a subset of a generally large solution space, that characterizes the response or output of a system in terms of changes in the factors (inputs) of the system. The typical objective is to build a prediction model for the output, through for instance, regression, where the variables are the inputs of the system. In such a process, the inputs are varied over a set of finite, usually small number of, values in order to observe the effect the changes have on the response. The set of values and experiments can be chosen in a number of ways, and it is important to identify the best choices to achieve an accurate characterization.

3.5.1 One-factor-at-a-time design

The one-factor-at-a-time is a simple approach to experimental design, where the inputs are varied one at a time over a specified range of acceptable values, instead of all simultaneously. The advantage of this approach is that the number of experiments is linear in the number of factors. The major disadvantage, however, is that it is not easy to estimate interactions between the factors [Czi99]. Furthermore, the impact of the factors can be more effectively estimated when they are varied simultaneously.

3.5.2 Statistical design of experiments

Design of Experiments is a systematic approach that provides an appropriate sampling of search space for response characterization. Unlike the one-factor-at-a-time technique of the previous section, this approach consists of simultaneously changing the factors, and the subsequent analysis of the resulting experimental data will identify the critical factors, the presence of interactions between the factors, etc. The influence of the individual factors is expressed as *main effects*, while *interaction effects* describe the influence of interactions. For a system affected by N factors, there are N main effects, $\binom{N}{2}$ two-factor interaction effects, and so on. In all, there are $2^N - 1$ effects that must be estimated.

Multifactorial designs and resolution

The size of the design, i.e., the number of experiments in the sampling, depends on and typically increases with the number of effects that need to be estimated. The simplest design, commonly referred to as *full factorial design*, permits estimation of all of the main and interaction effects. However, such a design involves experimenting over all possible number of factor combinations and the size is exponential in the number of factors.

On the other hand, *fractional factorial designs*, which require relatively less number of experiments, assume that some of the interaction effects are negligible and all other effects can be estimated. Specifically, each of the effects are grouped or *aliased* with some other effects, and it is only possible to estimate the sum of the effects of each group. In such a scenario, if all but one of the effects of a group are found to be negligible, then the sum can be solely attributed to that one nonnegligible effect. The grouping pattern can be determined from the structure of the design.

Fractional factorial designs are categorized using the concept of *design resolution*. For a design with resolution R, all of the main effects are grouped with interaction effects involving R-1 and higher number of factors. In general, any effect that is associated with i factors is aliased with effects of R-i and higher factor interactions, and the smallest possible resolution is III. A resolution III design, which has a minimum size that is linear in the number of factors like the one-factor-at-a-time design of Section 3.5.1, works under the assumption that all of the interactions are negligible, and this can be used to build a linear model for the response that includes only the main effects.

Components of effects

Each of the effects contain a linear component and components of higher degree such as quadratic. The number of estimable components for a factor is determined by the number of distinct values, also called factor levels, utilized for the factor in the design. In a k-level design, where each factor is varied across k different values, components of up to a degree of k-1 can be estimated. For instance, a three-level design can be used to compute both the linear and quadratic components of each of the main and interaction effects. However, the size of a design rapidly increases as the number of factor levels increases.

In this thesis, we utilize a two-level resolution III design for the floorplanning of the DLX architecture in Chapters 4 and 6. However, for the Pentium architecture, owing to the associated nonlinearities that will be explained later in Chapter 5, we use a two-step approach, where the first step is a screening two-level resolution IV design that is used to separate factors that have insignificant impact on the response. In the second step, a three-level resolution V design is applied for the remaining, significant, factors to build a quadratic response surface model.

3.5.3 Significance testing

Significance testing is an important part of experimental design to identify whether an effect can substantially affect the response. A typical application is hypothesis testing, where a *null hypothesis* is set up to test an *alternative hypothesis*, which has more effects/terms in the model. If the null hypothesis matches the results of the alternative hypothesis, then the additional terms (of the alternative hypothesis) are statistically insignificant and can be easily discarded.

The F-test, that we employ in this thesis, is the widely used scheme for significance testing. The test involves matching the residual sum of squares of an effect (interaction or main) to the sum of squares of the error. If both are comparable, then the effect can be termed unimportant and can be safely removed from the model. For the purpose of comparison, the ratio of the two sum of squares (for the effect and error/noise) is used to index the F-distribution [Mon00]. The result is a p-$level$ which corresponds to the likelihood of the effect being significant. The higher the p-$level$, the lower is the likelihood, and a p-$level$ of 0.05 is typically used as the threshold of statistical significance, i.e., if the value is greater than 0.05, the effect can be ignored from the model.

3.6 Floorplanning

Floorplanning involves finding an optimal placement of the blocks of a circuit on the layout of the circuit hat minimizes a cost function that typically includes topological attributes such as the layout area, total wire length and aspect ratio. The problem is an instance of combinatorial optimization that belongs to the class of NP [Ger99], several heuristics have been proposed in the past. Most widely used algorithms employ a Simulated Annealing (SA) framework [Ger99]. This technique involves iterating over several candidate floorplans before arriving at a near-optimal solution. At each step, the algorithm makes a move and constructs a new solution by slightly changing current solution. If the new solution has a lower cost than the current solution, it is accepted. If it has a higher cost, it is accepted with a probability that depends on the difference in the costs and on a parameter called the annealing temperature that is gradually decreased during the process. If the new solution is accepted, it replaces the current solution. The probability of acceptance (of new solutions that have higher cost than the current solutions) is high initially and is gradually decreased. This process ensures that the method does not get stuck in a “local minimum”.

Chapter 4

Throughput-aware microarchitecture floorplanning

4.1 Introduction

As noted in section 1.3.2, employing wire-pipelining to support high frequencies can result in a reduction in the throughput of the circuit (increase in the CPI of (1.1)) due to the increase in the number of clock cycles per computation. This penalty depends on the locations at which these extra latencies are added: increasing the latencies on some buses can impact the throughput more than on others.

In particular, the number of flip-flops that must be inserted on a bus is proportional to the length of the bus, which in turn depends on the locations of the connecting functional units (end points) of the bus in the layout. These lengths are determined during the *physical design* step of the microprocessor circuit design cycle, which transforms a functional net-list into a circuit layout, through procedures that include floorplanning, placement, and routing.

For improved performance, physical design must attempt to keep the CPI-critical buses as short as possible to minimize the amount of pipelining required by those buses. Such a microarchitecture/CPI-aware strategy [Sch02] is particularly useful at floorplanning or block-level placement, which being an early stage of physical design, has a major role in determining the system/global bus delays. For a particular combination of bus latencies, the CPI can be computed using cycle-accurate simulations on simulators such as SimpleScalar [BA97], on widely-used benchmark programs such as SPEC [Hen00].

The clear bottleneck in such a design flow is the microarchitecture simulation time. Firstly, cycle-accurate simulations are inherently slow, and this, coupled with the large search space considered during physical design optimizations, makes it virtually impossible to use simulations for each layout that is to be evaluated. Specifically, if each of n wires on a layout can have k possible latencies, then the cycle-accurate simulator may have to perform up to k^n simulations to fully explore the search space.

The exponentially large search space prompts us to consider a design of experiments (DOE) [Mon00] strategy, a well-established approach that is particularly efficient at extracting the basic characteristics of a large design space through a small number of samples, as described in Section 3.5.2. Specifically, we propose a strategy [NCLS05], based on a multifactorial resolution III design, to accurately identify the CPI-critical wires to be optimized in physical design, and then applies the methodology to floorplanning. The advantage of this approach is that the total number of simulations required to sample the space is proportional to n, compared to the $O(k^n)$ possible combinations of bus latencies. The CPI-critical wires are explicitly identified and regression models are constructed to estimate CPI, and these are used in the cost function of floorplanning.

Even with n simulations, the simulation time of each run is still an issue. The SPEC benchmark suite [Hen00], along with the `reference` input sets has become the *de facto* standard for microarchitecture research. However, reference input sets comprise huge instruction counts and therefore have long run times, typically in the range of a few days to run to completion. To maintain the run times with in practical limits, it is essential to employ alternative techniques that speed up the simulations, such as reducing the size of the input sets and statistical sampling. This reduction in the simulation times, however, comes at the cost of loss of accuracy associated with simulating only a fraction of the `reference` input sets. Such inaccuracies can potentially lead to incorrect conclusions and performance bottlenecks, and, therefore, can undermine a microarchitecture optimization process such as CPI-aware floorplanning.

Due to the inaccuracies, it is necessary to understand the nature of the simulation speedup techniques, and, importantly, how these techniques affect the results of the optimization, i.e., whether different approaches lead to different conclusions and optimizations. We compare [NCLS06] two simulation techniques, namely, reduced input sets and sampling, for the proposed CPI-aware floorplanning, and study their impact on the overall performance speedup obtained.

The remainder of the chapter is organized as follows. Section 4.2 describes some related work. Section 4.3 presents the design flow of the proposed CPI-aware floorplanning methodology, along with the baseline architecture and block configuration used in this work, while section 4.4 outlines the simulation speedup techniques compared for CPI-aware floorplanning. Section 4.5 demonstrates the experimentation process and the results. We conclude the chapter in section 4.6.

4.2 Related work

CPI-aware floorplanning: There have been some recent attempts [LSLH04,EMW+04, JYK+05] towards microarchitecture-aware design at the floorplanning level. In [LSLH04], a CPI look-up table (LUT), indexed by the set of bus latencies, is constructed using cycle-accurate simulations. For a given layout (and the corresponding bus latencies), the CPI is evaluated from the LUT using some distance metrics. In contrast, the approach in [EMW+04] assigns weights to each of the system buses that are proportional to the amount of traffic seen on the buses, operating under the notion that the more often a bus is accessed, the more critical it is. The objective of the floorplanner then is to minimize a weighted sum of bus latencies, where the weights depend on the amount of traffic. The work of [JYK+05] uses a one-factor-at-a-time (refer to Section 3.5.1) approach to build CPI sensitivity models for a few selected critical paths, and these models guide the floorplanner to maximize the system throughput (or minimize CPI).

While these approaches indicate welcome progress in the quest for microarchitecture-aware design, the accuracy of the strategies used to optimize the CPI-critical wires shows room for improvement. For instance, the LUT has to be reconstructed if a different frequency is chosen. On the other hand, bus access frequencies may not exactly capture the quantitative impact of the bus latencies on the CPI. Specifically, the effect of extra latencies on the execution path of a particular operation is primarily determined by the dependencies the following instructions have on the data generated by that operation.

In a way, all of these approaches focus on assigning a weight of importance for each wire of the circuit, and propose strategies to reduce the number of simulations required for tapping the large solution space. While [EMW$^+$04] uses a latency-independent and traffic-oriented approach, [LSLH04, JYK$^+$05] vary the latencies in the simulations. In this respect, our DOE based methodology, although has the same objective, provides a structured approach for conducting simulations. Such an approach, where the inputs, i.e., bus latencies, are varied simultaneously, can capture the solution space in a much better way than the one-factor-at-a-time approaches [Czi99], such as the one proposed in [JYK$^+$05].

Furthermore, the DOE method also provides a framework for estimating the interactions between the inputs. While [LSLH04] consists of simultaneously changing the bus latencies, it does not model any interactions between buses. As will be seen later in Sections 4.3.2 and 4.5, there are instances where the interactions can be significant. In addition, it may be a better idea to focus on specific buses than paths as done in [JYK$^+$05], since some paths can have common buses, which complicates the latency modeling. For instance, the instruction commit path, which handles updates to the register file, and the decode path, which dispatches the decoded instructions along with any available data through the register file to the reorder buffer, can have a common (bidirectional) bus, between the reorder buffer and the register file. The approach of [JYK$^+$05] focuses on buses that have no buses in common, and does not address the extension to cases where paths can have common buses. Nevertheless, the idea of the DOE can be extended for the paths addressed in [JYK$^+$05], for a more effective-modeling than the one-at-a-time approach considered.

Another recent work [CJRR03] explores the frequency-CPI tradeoff in floorplanning. A set of implementations varying in area and latency is specified for some or all of the blocks of the processor. The objective of the floorplanner is to find a configuration of blocks with a placement to reduce the product of clock period and the CPI. The lengths of the global wires, combined with given arrival times at the terminals, determine the clock period.

Comparing simulation techniques: With respect to reducing simulation times of the `reference` input sets of the SPEC benchmarks, which can speed up our DOE based approach, a recent work [YKS$^+$05] evaluates the accuracies of a number of simulation techniques, including reduced input sets and sampling. The comparison is based on three different characterizations, one each at the hardware (processor bottleneck), software (execution profile), and architecture levels. In addition, the work attempts to quantify the effect of the inaccuracies on the execution times of the benchmarks, for a couple of microarchitecture enhancements [YL02, Jou90]. The results of the comparison indicate that, in general, sampling techniques are more reliable than reduced input sets in tracking the actual performance speedups obtained, due to the enhancements, on the `reference` sets.

However, while these results hold for the enhancements considered, it is possible that the impact of the inaccuracies can vary across different optimizations. Specifically, for the hardware enhancements handled in [YKS$^+$05], the decision making is directly based on the results obtained from the simulations, and therefore a high reliability is required. CPI-aware floorplanning, on the other hand, is a discrete optimization problem where the variables are bus latencies. The purpose of the simulations is to describe the CPI of a program as a function of the bus latencies, and the floorplanner uses this description to determine a block-level placement that represents an CPI-optimal bus latency configuration.

For such optimization problems, a reasonably accurate characterization that does not significantly alter the relative ordering of the performance-criticality of the parameters is sufficient; "absolute" accuracy may not be necessary. We focus on this issue in this thesis and the objective is to determine if there is any correlation between per-

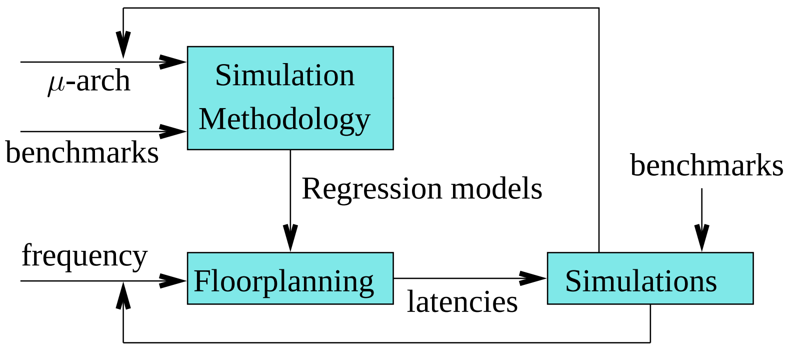


Figure 4.1: CPI-aware floorplanning: design flow.

ceived inaccuracy of the reduced input sets and the corresponding optimization results for the CPI-aware floorplanning problem. Although our study specifically concentrates on floorplanning, the results are likely to be applicable for any microarchitecture optimization in the physical design context, or, in fact, any related discrete (microarchitecture) optimization problem.

4.3 CPI-aware floorplanning flow

The amount of pipelining required by each bus of a microprocessor is proportional to its length, which is typically true for buffered interconnects [She95], and therefore, for every block-level placement, where the blocks represent the functional units of the processor, there is a corresponding bus-latency configuration. For each of these configurations, the CPI for a given program can be determined using a cycle-accurate simulation. The objective of floorplanning is to obtain a bus-latency configuration that minimizes the CPI for each benchmark program.

To incorporate wire-pipelining issues into floorplanning, we develop a design flow for microarchitecture-awareness, as depicted in Figure 4.1. The first step is to quantify the impact of each system bus on the system performance through a CPI regression model for each of the chosen benchmark programs. The regression models (and coefficients) may differ across the benchmarks, depending upon the instruction mix executed. The concept of using these models is similar to the access-frequencies based weights of [EMW$^+$04], but the precise manner in which we obtain the weights (regression coefficients) is different. A comparison between the two approaches is shown in Section 4.5.3.

The CPI regression models are then fed to the floorplanner, along with a target frequency. The objective of the floorplanner is to determine the positions of the blocks, therefore the set of bus latencies, such that the CPI, in addition to traditional objectives such as area and aspect ratio, is minimized. 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 4.1 may be repeated for several microarchitectural block configurations to identify the optimal configuration [CJRR03]. For a general case, the CPI model to be used in floorplanning may be obtained by combining the regression models obtained from optimizing the processor performance on a set of benchmarks.

The succeeding sections illustrate this approach, and we tie the description to the processor microarchitecture employed in this work. The DLX microarchitecture, which is essentially a five-stage pipeline defined in the SimpleScalar simulator [BA97], and the corresponding functional blocks are shown in Table 4.1 and Figure 4.2, respectively.

The instruction fetch and decode blocks are shown as fet and dec, respectively, while $il1$ and $dl1$ are the level-1 instruction and data caches, respectively. The instruction and data translation look-aside buffers (TLB) are indicated as $itlb$ and $dtlb$, respectively, while $l2$ represents the unified level-2 cache. The block ruu is the register update unit, which contains the reservation stations and instruction issue logic, while the block lsq represents the load store queue. The system register file is represented by reg, whereas $bpred$ consists of the branch predictor and the target buffer (BTB), which predict the direction and target address for a branch instruction, respectively. The blocks $iadd1$, $iadd2$, $iadd3$, $imult$, $fadd$ and $fmult$ are the functional units that exe-

Parameter	**Value**
Fetch width	8 instrs/cycle
Issue width	8 instrs/cycle
Commit width	8 instrs/cycle
RUU entries	128
LSQ entries	64
IFQ entries	16
Branch pred	comb, 4K table 2-lev 2K table, 11-bit 2K BHT
BTB	512 sets, 4-way
IL1	64K, 64B, 2-way LRU, latency: 1
DL1	32K, 32B, 2-way LRU, latency: 1
L2	2M, 128B, 4-way latency: 12
ITLB, DTLB	128 entries Miss latency: 200

Table 4.1: Configuration of the microarchitecture used in this work.

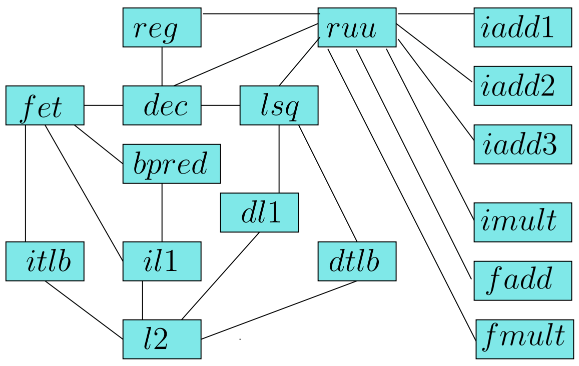


Figure 4.2: The functional blocks and buses of the microarchitecture of Table 4.1. The blocks are shown as rectangles, while the lines between the rectangles (blocks) represent the buses connecting them.

cute arithmetic and logic instructions. The figure also shows the 22 system buses that can impact the throughput/performance (IPC or the number of instructions executed per cycle, which is the reciprocal of CPI) of the processor, when pipelined.

4.3.1 Wire pipelining models

The first step of the floorplanning flow is introducing wire pipelining models into the chosen simulator, which in this work is based on sim–outorder, a detailed simulator provided in the SimpleScalar package. The simulator is modified to include extra latencies on these buses as additional delays. To achieve this, we use 19 factors to model the 22 buses, as shown in Table 4.2, where 17 of the 19 factors directly model the buses with the same name. The modeling of extra latencies is described below, for each stage of the DLX processor pipeline.

- **Fetch:** The typical path followed by an instruction, on an $il1$ hit, from the initiation of its fetch to its insertion into the fetch queue is fet–$il1$–$bpred$–fet. The addition of latencies on any of the buses of this path is equivalent to inserting dummy pipeline stages on the fetch path. This path is modeled by the factor $extra\_fet$,

Bus	**Factor**	**ID**
dec_reg	dec_reg	f_1
ruu_reg	ruu_reg	f_2
dec_ruu		
dec_lsq	max_lsq_ruu	f_3
ruu_iadd1	ruu_add1	f_4
ruu_iadd2	ruu_iadd2	f_5
ruu_iadd3	ruu_iadd3	f_6
ruu_imult	ruu_imult	f_7
ruu_fadd	ruu_fadd	f_8
$il1\_l2$	$il1\_l2$	f_9
$dl1\_l2$	$dl1\_l2$	f_{10}
fet_il1		
$il1\_bpred$	$extra\_fet$	f_{11}
fet_bpred		
fet_dec	fet_dec	f_{12}
fet_itlb	fet_itlb	f_{13}
$itlb\_l2$	$itlb\_l2$	f_{14}
ruu_lsq	ruu_lsq	f_{15}
ruu_fmult	ruu_fmult	f_{16}
lsq_dl1	lsq_dl1	f_{17}
$dtlb\_l2$	$dtlb\_l2$	f_{18}
lsq_dtlb	lsq_dtlb	f_{19}

Table 4.2: The set of buses of the microarchitecture of Figure 4.2, and the factors, with the corresponding IDs, that model the impact of pipelining the buses in the simulator. There are 22 buses, which are grouped into 19 factors, where most of the factors have a one-to-one relation with the buses, except $extra\_fet$ and max_lsq_ruu.

whose latency is the sum of the latencies of buses along the path. When an $il1$ miss occurs, the $itlb/il1$ miss penalty is added, which, in turn, might have been increased due to extra latencies on the buses $fet\_itlb/itlb\_l2/il1\_l2$. Each of these is represented as a separate factor with the same name, as shown in Table 4.2.

- **Decode:** This stage performs register renaming and dispatches instructions to ruu and lsq (for memory operations). In addition, if the input data corresponding to the instruction is available in reg, it is forwarded to the ruu. For functional correctness, therefore, the latencies of the buses dec_ruu, dec_lsq, and the path dec–reg–ruu must be equal. The number of extra decode stages can then be determined as the maximum of the latencies of these buses summed with that of fet_dec, as shown in (4.1).

$$ex\_dec = fet\_dec + \max \left\{ \begin{array}{c} dec\_lsq \\ dec\_ruu \\ dec\_reg + ruu\_reg \end{array} \right\} \tag{4.1}$$

  However, the path ruu–reg also appears in the instruction commit stage, when data from an ruu entry is written to the register file. Due to this, unlike the fetch stage, the extra decode stages cannot be modeled by a single parameter ex_dec, although dec_ruu and dec_lsq can be combined into a single factor, which we name max_lsq_ruu, indicated in Table 4.2, and defined as shown below:

$$max\_lsq\_ruu = \max\{dec\_lsq, dec\_ruu\} \tag{4.2}$$

  The number of extra decode stages is then internally computed in the simulator using (4.1) and (4.2).

- **Issue:** This stage issues ready-to-execute instructions to the corresponding execution unit upon availability, and schedules writeback events. The changes required in this stage are adjustments to the functional unit and the data cache ($dl1$ and $l2$) access latencies. In addition, we have incorporated functional unit scheduling in the simulator. For instance, the number of latencies inserted on the three buses ruu_iadd1, ruu_iadd2 and ruu_iadd3 can be different, and while issuing an integer add instruction, of all the available units, the one with the least latency is chosen.

- **Writeback:** This stage is accounted by altering the branch misprediction latency, which is modeled by the factors related to the extra fetch and decode latencies.
- **Commit:** The instruction commit latency is adjusted, and this is modeled by the factor/bus ruu_reg.

Each of the factors is made completely configurable by modifying the SimpleScalar configuration file.

4.3.2 Simulation methodology

The next step of the proposed flow is to use the wire-pipelining-aware simulator constructed as described in the previous section, to quantify the performance impact of the factor/bus latencies in the form of CPI regression models. To reduce the number of simulations required for this purpose, we use a strategy based on the theory of statistical *design of experiments*.

As noted in Section 3.5.2, the simplest design, commonly referred to as *full factorial design*, permits estimation of all of the main and interaction effects. However, such a design involves experimenting over all combinations of the possible values subscribed by the factors. As mentioned earlier in Section 4.1, the number of possible bus latency configurations in floorplanning is an exponential function of the number of factors. Even though the number of factors N is relatively small ($N = 19$) for this microarchitecture, given the high simulation times, it is impractical to use cycle-accurate simulations for each of the allowable configurations to determine the response, which in this case is the CPI of a program, that needs to be minimized (to maximize the throughput or IPC, its reciprocal).

We address the problem of reducing the number of simulations with a few assumptions. Each of the factors is restricted to have two levels: the minimum and the maximum possible values for the factor, thereby permitting us to employ a two-level factorial design. The idea is that, by stimulating the system with inputs at their extreme values, we provoke the greatest response for each input. The assumption is that the system response is a monotone function of changes in the inputs (factor levels). While this assumption cannot be guaranteed in these types of systems, it works quite well in practice[1]. Besides, higher level designs, which permit the estimation of quadratic and higher degree components of the effects as described in Section 3.5.2, exhibit a complex effect structure and require more simulations, which make them unreasonable for studies like ours. As is shown in [YLH03], the two-level approach can be effectively used to design simulation strategies for microarchitectural optimizations.

Since the factor levels represent bus latencies, the extreme (high and low) values can be obtained by assuming worst-case and best-case scenarios for the corresponding wire lengths. The high/low value for a bus latency may be determined by placing the connecting blocks as far/close as possible. A valid assignment may, for example, be 0 for the low value, and the latency corresponding to a corner-to-corner connection across the chip for the high value.

Interactions

In general, it is not easy to identify potential significant interactions before hand in a complex system such as a microprocessor. However, in most cases, the interactions in a microarchitecture tend to be negligible. For instance, it is unlikely that, say, the level-

[1]Although this is not a proof, it seems intuitively acceptable to believe that increasing the latency of a bus will decrease the system throughput.

2 cache ($l2$) interacts with the instruction decoder, given the varied functionalities of the two units. We have identified a few potential significant interactions, which resulted from the nature of wire-pipelining models integrated into the simulator, as shown below:

We have identified a few potential significant interactions, which resulted from the nature of wire-pipelining models integrated into the simulator.

- As we have incorporated functional unit scheduling in the simulator, the impact of adding additional latencies on each of the buses between the register update unit and the three integer adders, i.e., those modeled by the factors ruu_iadd1 (ID: f_4), ruu_iadd2 (ID: f_5) and ruu_iadd3 (ID: f_6) of Table 4.2, is also determined by the latencies of the other two buses. This indicates possible significant (two and three factor) interactions.

- As shown in (4.1) and (4.2), the number of extra pipeline stages to be inserted in the decode stage is modeled as a maximum function of three factors dec_ruu (ID: f_1), max_lsq_ruu (ID: f_2) and ruu_reg (ID: f_3). Such a nonlinear function can result in significant interactions among these three factors.

We use the notation $f_i \cdot f_j$ to denote the interaction between factors f_i and f_j, where '$\cdot$' represents the interaction operator. According to this terminology, the eight interactions defined in the previous paragraph can be written as follows:

$$\begin{aligned} \textit{two–factor}: &\quad f_1 \cdot f_2; f_1 \cdot f_3; f_2 \cdot f_3 \\ &\quad f_4 \cdot f_5; f_4 \cdot f_6; f_5 \cdot f_6 \\ \textit{three-factor}: &\quad f_1 \cdot f_2 \cdot f_3 \\ &\quad f_4 \cdot f_5 \cdot f_6 \end{aligned} \tag{4.3}$$

Run	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	y
1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	-1	y_1
2	-1	-1	-1	-1	1	-1	-1	-1	1	1	-1	1	1	1	1	-1	-1	-1	1	y_2
3	-1	-1	-1	1	-1	1	1	1	-1	1	1	-1	1	1	-1	1	-1	-1	1	y_3
4	-1	-1	-1	1	1	1	-1	1	1	-1	1	1	-1	-1	-1	-1	1	1	-1	y_4
5	-1	-1	1	-1	-1	-1	-1	1	1	-1	1	1	-1	1	-1	-1	-1	-1	1	y_5
6	-1	-1	1	-1	1	-1	1	1	-1	1	1	-1	1	-1	-1	1	1	1	-1	y_6
7	-1	-1	1	1	-1	1	-1	-1	1	1	-1	1	1	-1	1	-1	1	1	-1	y_7
8	-1	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	1	1	1	-1	-1	1	y_8
9	-1	1	-1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	1	-1	1	y_9
10	-1	1	-1	-1	1	1	-1	-1	1	1	1	-1	-1	1	-1	1	-1	1	-1	y_{10}
11	-1	1	-1	1	-1	-1	1	1	-1	1	-1	1	-1	1	1	-1	-1	1	-1	y_{11}
12	-1	1	-1	1	1	-1	-1	1	1	-1	-1	-1	1	-1	1	1	1	-1	1	y_{12}
13	-1	1	1	-1	-1	1	-1	1	1	-1	-1	-1	1	1	1	1	-1	1	-1	y_{13}
14	-1	1	1	-1	1	1	1	1	-1	1	-1	1	-1	-1	1	-1	1	-1	1	y_{14}
15	-1	1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	1	1	-1	1	y_{15}
16	-1	1	1	1	1	-1	1	-1	-1	-1	1	1	1	1	-1	-1	-1	1	-1	y_{16}
17	1	-1	-1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	1	y_{17}
18	1	-1	-1	-1	1	1	-1	1	-1	-1	-1	1	1	1	-1	1	1	-1	-1	y_{18}
19	1	-1	-1	1	-1	-1	1	-1	1	-1	1	-1	1	1	1	-1	1	-1	-1	y_{19}
20	1	-1	-1	1	1	-1	-1	-1	-1	1	1	1	-1	-1	1	1	-1	1	1	y_{20}
21	1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1	1	1	1	1	-1	-1	y_{21}
22	1	-1	1	-1	1	1	1	-1	1	-1	1	-1	1	-1	1	-1	-1	1	1	y_{22}
23	1	-1	1	1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1	-1	1	1	y_{23}
24	1	-1	1	1	1	-1	1	1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	y_{24}
25	1	1	-1	-1	-1	-1	1	1	1	1	1	1	1	-1	1	1	-1	-1	-1	y_{25}
26	1	1	-1	-1	1	-1	-1	1	-1	-1	1	-1	-1	1	1	-1	1	1	1	y_{26}
27	1	1	-1	1	-1	1	1	-1	1	-1	-1	1	-1	1	-1	1	1	1	1	y_{27}
28	1	1	-1	1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	y_{28}
29	1	1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	1	1	-1	-1	1	1	1	y_{29}
30	1	1	1	-1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	y_{30}
31	1	1	1	1	-1	1	-1	1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	y_{31}
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	y_{32}

Table 4.3: Resolution III design matrix M for 19 factors. Each of the 32 rows corresponds to a simulation run, while the columns represent the factors: each entry $M(j,i)$ contains the value to be used for factor f_i in run j. The two levels of each factor are coded as ± 1, and the label y is the response, CPI, of the system (microarchitecture).

Resolution III design

Using our knowledge of the behavior of the process, it is reasonable assume that all of the other interactions are negligible, allowing us to utilize a resolution III fractional factorial design [Mon00], which provides the logically minimum number of experiments to determine the main effects of the factors. For N factors, the number of experiments required is equal to the nearest highest power of 2, which turns out to be 32 for our work, since $N = 19$. The design is captured by a simulation matrix M of size $32 \times N$, as shown in Table 4.3. Each of the 32 rows of M corresponds to a simulation run, and y_i represents the response (CPI) obtained during i^{th} simulation. The two levels of each factor are encoded as {+1, –1}, and the idea is to estimate the effect of changing the level of the factor from "$+1$" to "-1". The columns, labeled c_1–c_{19}, correspond to the 19 factors f_1–f_{19} described in Table 4.2. Each level (± 1) is contained in exactly half of the simulation runs, indicating that every column (which corresponds to a factor) of M has 16 "+1"s and 16 "–1"s. Furthermore, no two columns are identical, i.e., each column has a distinct mix of "$\pm$1s".

Each of the columns of M results in a distinct *contrast* or effect, from which the main and interaction effects of the factors can be determined. The contrast/effect for column c_i, e_i, is computed as the difference of the responses where the entry of c_i is "+1" (maximum value of factor f_i) and those where the level is "–1" (minimum value of f_i), as shown below:

$$e_i = \sum_{j=1}^{j=32} M(j,i) \cdot y_j \tag{4.4}$$

The matrix M is constructed as follows. First, a two-level full-factorial is generated for five factors f_1–f_5 (refer to Table 4.2) involving all possible ($2^5 = 32$) combinations, as shown by columns labeled c_1–c_5 of M in Table 4.3. The values to be used for the remaining 14 factors (f_6–f_{19}), listed in columns c_6–c_{19} of Table 4.3, are obtained through component-wise multiplication of several combinations of the five full-factorial columns c_1–c_5, as defined by the following *incidence relation*.

$$
\begin{aligned}
f_1 & \quad (dec\_reg) & = \quad & c_1 \\
f_2 & \quad (ruu\_reg) & = \quad & c_2 \\
f_3 & \quad (max\_lsq\_ruu) & = \quad & c_3 \\
f_4 & \quad (ruu\_iadd1) & = \quad & c_4 \\
f_5 & \quad (ruu\_iadd2) & = \quad & c_5 \\
f_6 & \quad (ruu\_iadd3) & = \quad & c_1 \times c_2 \times c_4 \\
f_7 & \quad (ruu\_imult) & = \quad & c_3 \times c_5 \\
f_8 & \quad (ruu\_fadd) & = \quad & c_1 \times c_3 \times c_4 \\
f_9 & \quad (il1\_l2) & = \quad & c_1 \times c_3 \times c_5 \\
f_{10} & \quad (dl1\_l2) & = \quad & c_1 \times c_4 \times c_5 \\
f_{11} & \quad (extra\_fet) & = \quad & c_2 \times c_3 \times c_4 \\
f_{12} & \quad (fet\_dec) & = \quad & c_2 \times c_3 \times c_5 \\
f_{13} & \quad (fet\_itlb) & = \quad & c_2 \times c_4 \times c_5 \\
f_{14} & \quad (itlb\_l2) & = \quad & c_3 \times c_4 \times c_5 \\
f_{15} & \quad (ruu\_lsq) & = \quad & c_1 \times c_2 \times c_3 \times c_4 \\
f_{16} & \quad (ruu\_fmult) & = \quad & c_1 \times c_2 \times c_3 \times c_5 \\
f_{17} & \quad (lsq\_dl1) & = \quad & c_1 \times c_3 \times c_4 \times c_5 \\
f_{18} & \quad (dtlb\_l2) & = \quad & c_2 \times c_3 \times c_4 \times c_5 \\
f_{19} & \quad (lsq\_dtlb) & = \quad & c_1 \times c_2 \times c_3 \times c_4 \times c_5
\end{aligned} \tag{4.5}
$$

For instance, the entries of column c_6 in M, e_6, which contains the values to be used for factor f_6 (ruu_iadd3) of Table 4.2, is obtained by the multiplication of the corresponding entries of columns c_1, c_2 and c_4, i.e., the values that are used for factors f_1 (dec_reg), f_2 (ruu_reg) and f_4 (max_lsq_ruu). Such a set-up indicates that the effect of factor f_6 is aliased with that of the interaction of the three factors, f_1, f_2 and f_4, i.e., $f_1 \cdot f_2 \cdot f_4$. In other words, the contrast e_6, computed using (4.4), is the sum of both the main effect of f_6 and the interaction effect of $f_1 \cdot f_2 \cdot f_4$[2]. Assuming that the effect of the interaction $f_1 \cdot f_2 \cdot f_4$ is negligible, then e_6 can be solely attributed the main effect of factor f_6.

In reality, a resolution III design of size 32 can be used for up to 31 factors, in which case, it is called a saturated design, and these can only be used where all of the interactions are insignificant. However, since we only have 19 factors, some of the combinations, 12 of them, of the columns c_1–c_5 of M are left unused. The unused combinations can be treated as dummy factors, which have negligible (zero) effects, and these can be used to estimate a few selected interactions. Using the resolution III design prescribed by M and the associated incidence relation of (4.5), it is possible to estimate all of the interaction effects described earlier in section 4.3.2, in addition to the 19 main effects. For instance, the combinations, say $c_{20} = c_1 \times c_2$ and $c_{21} = c_1 \times c_2 \times c_3$, are not used in (4.5), indicating the contrasts e_{20} and e_{21} solely represent the effects of the two-factor interaction between f_1 and f_2, $f_1 \cdot f_2$, and the three-factor interaction among 1, 2 and 3, $f_1 \cdot f_2 \cdot f_3$, respectively, where f_{20} and f_{21} are dummy factors with zero (main) effects.

In addition, it can be noted that the two-factor interaction effects of $\{ruu\_iadd1$, ruu_iadd2, $ruu\_iadd3\}$, i.e., $\{f_4, f_5, f_6\}$, must be equal due to symmetry. Therefore, estimating one of them will suffice, and this is true for the corresponding main effects as well.

The advantage of fractional factorial resolution III designs over other screening designs such as Plackett and Burman (PB) [PB56], which is employed in [YLH03], is the well defined aliasing structure. This attribute can be used to estimate a few required interactions, as is done in this project, at the expense of a few additional simulations[3].

[2]The contrast includes many other higher order interaction effects, as defined by a complete incidence relation. We refer the reader to [Mon00] for further details about the aliasing structure of fractional factorial designs.

[3]For N factors, the number of experiments required in a PB design is equal to the

Finally, if more interactions need to be considered in the design, and the number of dummy factors is inadequate, then one option is to perform appropriate additional orthogonal runs [WH00]. If this is not sufficient, then a higher resolution (IV or more) design can be utilized, if the associated simulation cost[4] can be tolerated.

CPI regression models

From the 32 simulations performed using the resolution III design of Table 4.3 for each benchmark, we construct a regression model for CPI, based on least-squares approximation, where the variables are the bus latencies. More specifically, for fractional factorial designs, the contrasts obtained from the design using (4.4) directly correspond to the regression coefficients determined through least-squares minimization. Equation (4.6) shows one such a model, where β_is represent the regression coefficients computed from the 32 CPI values of the resolution III design. Each x variable in (4.6), say x_i, represents an encoding of the latency of factor f_i, l_i, where the minimum and the maximum latencies are coded as -1 and +1, respectively, and $\mathcal{I}$ is the set of interactions described in section 4.3.2 and (4.3).

$$\begin{aligned} x_i &= -1 + \left(\frac{2 \cdot l_i}{\min(f_i) + \max(f_i)} \right), \quad 1 \leq i \leq 19 \\ CPI &= \beta_0 + \sum_{i=1}^{19} \beta_i \cdot x_i + \sum_{(f_i \cdot f_j) \in \mathcal{I}} \beta_{i.j} \cdot x_i \cdot x_j + \\ & \quad \sum_{(f_i \cdot f_j \cdot f_k) \in \mathcal{I}} \beta_{f_i \cdot f_j \cdot f_k} \cdot x_i \cdot x_j \cdot x_k \end{aligned} \tag{4.6}$$

next highest multiple of four (20 for 19 factors in this work), unlike the nearest highest power of two in a resolution III fractional factorial design used in this work.

[4]A resolution IV fractional factorial design for the microarchitecture of this work has a minimum size of 64.

4.3.3 Floorplanning cost function

Our floorplanner procedure is based on simulated annealing (SA), and uses the CPI regression models built out of the simulation methodology described in section 4.3.2 in the cost function. We use PARQUET [AM01], a floorplanner available in the public domain for this purpose.

The cost function of the floorplanner is a weighted sum of topological objectives such as the chip area ($Area$) and the aspect ratio (AR), and the CPI estimated using the regression model, as shown below:

$$Cost = W_1 \cdot Area + W_2 \cdot AR + W_3 \cdot CPI \tag{4.7}$$

where the Ws represent the relative weights of the optimization terms.

4.4 Reducing simulation times

It is widely accepted that the SPEC benchmark suite [Hen00], along with the `reference` input sets, represents a realistic work-load that is executed on microprocessors, and therefore has become an accepted standard in microarchitecture research such as the optimization problem addressed in this chapter. However, executing the `reference` input sets to completion, in most cases, is prohibitive, due to the inherent slow nature of the cycle-accurate simulations; simulating one cycle of the target microarchitecture consumes about 3000–5000 cycles of the host machine. For this reason, although the resolution III design described in section 4.3.2 considerably reduces the number of simulations (from exponential to linear in the number of buses/factors), the run time of each simulation is still an issue.

Several techniques have been proposed in the past to reduce simulation times to practical levels, while attempting to reproduce the behavior of the `reference` input sets. These techniques can be broadly categorized into three groups: (i) Reducing the input sets, (ii) Truncated execution, and (iii) Sampling. The work of [YKS$^+$05] compares the accuracies of six such techniques, and the results of the work indicate that sampling techniques have much higher accuracies in tracking microarchitecture (`reference`) performance than the other two categories. However, as explained in section 4.1, the findings are specific to the two enhancements considered, and it is not clear whether the inaccuracies can be generalized for all microarchitectural optimizations. In discrete optimization problems such as CPI-aware floorplanning, a moderate perturbation in the weight (regression coefficient) of a factor (or an interaction) may not be sufficient to shift the optimal value of that factor by an integer above or below.

Specifically, changing the latency of a bus in a particular placement involves a significant change in the locations of the connecting blocks in the layout, to increase/decrease the bus length by appropriate amount, and this can potentially lead to a massive realignment of the positions of other blocks, resulting in a drastically different placement with a significant change in the value of the cost function, which includes the weighted sum of factor/interaction latencies. It is unlikely that small or moderate perturbations in factor regression coefficients can result in such a scenario, which changes the cost function by a significant amount, during optimization. Therefore, any simulation technique with a reasonable accuracy (or moderate inaccuracy) may be sufficient in problems such as CPI-aware floorplanning.

In this chapter, we compare a few approaches that can be used to speed up the simulation methodology described in section 4.3.2 for the CPI-aware floorplanning problem. Due to the high number of simulations (32 per benchmark) required for each technique, we limit our comparison to two techniques, namely, sampling and reduced input sets, as shown below:

- *Reduced Input sets:* The idea behind the reduced input sets is to alter the `reference` input sets so that the simulation times are reduced when using these reduced input sets, while endeavoring to retaining the characteristics of the unaltered `reference` input sets. The `test` and `train` input sets from SPEC, and the MinneSPEC [KL02] reduced input sets are a few examples. For this work, we choose MinneSPEC reduced input sets for evaluation.

- *Sampling:* In statistical sampling, only selected portions of the instruction sequence of a benchmark are measured. The program segments between the selected portions are fast-forwarded using functional simulation. These samples must be chosen carefully such that they accurately reflect the behavior of the *population*, i.e, the whole program. The sampling technique proposed in [WWFH03], called SMARTS, simulates periodically selected subsets of the instruction sequence. The sampling frequency and the length of each sample are used to control the simulation time. The statistics measured for the simulated samples are generalized for the whole program. In addition, SMARTS uses statistical sampling theory to estimate the CPI error of the sampled simulation results, as compared to the complete simulation. On the other hand, the approach of SimPoint [SPHC02] selects a few representative simulation points beforehand and then uses statistical based clustering to select a set that is representative of the whole program. At the end of the simulation, the results from each simulation point are weighed to compute the final statistics. The number of simulation points, and the length of each determines the simulation time.

  We choose SMARTS as a representative of sampling techniques for our comparisons, since, as noted in [YKS+05], there is little difference in the accuracies of SMARTS and SimPoint.

- In addition to the above mentioned techniques, we consider a third case, a hybrid approach that is obtained by combining the two techniques. Specifically, in this case, we apply SMARTS on the MinneSPEC reduced input sets, to further reduce the simulation times. Hence, we actually compare three techniques in this thesis.

Besides, statistical simulation [NS01, EBN00, OCF00], which is not addressed in this thesis, is another way of reducing the simulation times, thereby allowing an efficient exploration of the design search space. For a benchmark program, using a single detailed simulation, statistical tables are constructed for various program characteristics such as cache miss rates and register dependencies. This synthesis trace generated, essentially a *statistical image* of the benchmark, can be used to speed up the subsequent simulations of the benchmark.

4.5 Experimentation

4.5.1 Benchmarks

We choose a set of eight SPEC 2000 benchmarks for evaluations in this work. The benchmarks, along with the corresponding `reference` and MinneSPEC input instruction counts are shown in Table 4.4. The total simulation time limited the number of benchmarks that we could use. The benchmarks are chosen because of their distinct instruction mixes. For instance, mesa has a high percentage of conditional branches, while the benchmark gcc has a very large number of memory operations. All benchmarks are compiled at optimization level O3 using the SimpleScalar version of the gcc compiler.

4.5.2 Set up

The areas of the blocks shown in Figure 4.2 are estimated using [SKLP+01], and are shown in Table 4.5. The total area of the chip is about 2cm^2 at 90nm technology, with the level-2 cache ($l2$) consuming about 70% of the area, as shown in the table. Only the chip core that also includes the L1 caches is considered during floorplanning, and the L2 cache is wrapped around the core floorplan, just as is done in [SSH+03] and Alpha 21362 [Ban98]. For the bus latency ranges that are to be used in the resolution III design, the minimum value is chosen to be 0, depicting the best case placement of the

Benchmark	**Type**	**Instr. count (Billions)**	
		MinneSPEC	`reference`
gzip	Int	1.065	63
vpr	Int	0.217	110
gcc	Int	0.175	34
mesa	FP	1.297	305
art	FP	7.700	54
equake	FP	0.716	175
parser	Int	0.914	301
bzip2	Int	3.800	94

Table 4.4: Benchmarks from the SPEC suite, along with the `reference` and reduced instruction counts.

connecting blocks. The maximum value chosen is equal to the corner-to-corner latency of the chip core, which is found to be 9 clock cycles at 6GHz, based on the projections of [SSH$^+$05]. We present results for three clock frequencies, ranging from 4GHz to 6GHz. The regression model constructed for each benchmark and technique can be used for all of these frequencies, since the bus latency ranges are valid for all of the frequencies less than or equal to 6GHz.

We assume that the operating frequency of the chip is constrained only by the bus delays, and the maximum of the delays of the buses is the minimum possible clock period when wire-pipelining is not employed. The corresponding maximum frequency, obtained by minimizing the maximum of wire lengths of the global wires in the floorplanner, is determined to be about 2.4GHz, and this forms the baseline unpipelined design.

The comparison metric is the execution time, T_{exec}, which, as shown in (4.8), is the product of the number of instructions (N_{inst}) in the benchmark, the CPI (CPI), and the

Block	**Area (cm^2)**	**Block**	**Area (cm^2)**
$il1$	0.097	dec	0.019
$dl1$	0.101	$bpred$	0.038
$ialu1$	0.006	fet	0.017
$ialu2$	0.006	$fadd$	0.016
$ialu3$	0.006	$fmult$	0.023
$imult$	0.012	$itlb$	0.003
ruu	0.125	$dtlb$	0.003
lsq	0.035	$l2$	1.471
reg	0.022		

Table 4.5: Areas of the microarchitecture blocks shown in Figure 4.2 for the configuration of Table 4.1. The total area of the processor is 2.03cm^2.

corresponding clock cycle time evaluated as the reciprocal of the clock frequency ($\frac{1}{f}$). It can be noted that this is the same equation as (1.1), with the clock period T_clk written as the reciprocal of the frequency.

$$T_{exec} = \frac{N_{inst} \cdot CPI}{f} \tag{4.8}$$

In addition, for all of the simulations using the SMARTS technique, both in `reference` and MinneSPEC input sets, we use the default values that are listed in [WWFH03] for the sampling parameters (a sampling interval of 1000, a sample size of 1000 and a warmup size of 2000 instructions).

4.5.3 Results

We present the results in two parts in this section. The first part demonstrates the efficacy of our proposed CPI-aware floorplanning methodology against a naive and an

Coefficient	**Factor**	**Value**
β_0	$mean$	100.00
β_1	dec_reg	0.86
β_2	ruu_reg	7.77
β_3	max_lsq_ruu	0.47
β_4	ruu_iadd1	11.24
β_5	ruu_iadd2	11.24
β_6	ruu_iadd3	11.24
β_7	ruu_imult	0.16
β_8	ruu_fadd	3.83
β_9	$il1\_l2$	0.64
β_{10}	$dl1\_l2$	1.48
β_{11}	$extra\_fet$	2.05
β_{12}	fet_dec	2.12
β_{13}	fet_itlb	0.59
β_{14}	$itlb\_l2$	0.16
β_{15}	ruu_lsq	5.64
β_{16}	ruu_fmult	1.40
β_{17}	lsq_dl1	8.66
β_{18}	$dtlb\_l2$	1.21
β_{19}	lsq_dtlb	3.67
$\beta_{1.2}$	$f_1 \cdot f_2$	-0.63
$\beta_{1.3}$	$f_1 \cdot f_3$	-0.53
$\beta_{2.3}$	$f_2 \cdot f_3$	-0.55
$\beta_{1.2.3}$	$f_1 \cdot f_2 \cdot f_3$	0.54
$\beta_{4.5}$	$f_4 \cdot f_5$	4.51
$\beta_{4.6}$	$f_4 \cdot f_6$	4.51
$\beta_{5.6}$	$f_5 \cdot f_6$	4.51
$\beta_{4.5.6}$	$f_4 \cdot f_5 \cdot f_6$	3.75

Table 4.6: Normalized regression coefficients, averaged over all of the eight benchmarks. The final eight rows correspond to the interaction terms shown in (4.3).

existing approach. For the purpose of this demonstration, we use SMARTS to speed up the `reference` simulations, both in the resolution III design strategy of section 4.3.2, and in validating the floorplanning results. Next we present the results of the comparison of the three techniques, SMARTS included, that can be used to speed up the resolution III design strategy.

Validation of the proposed floorplanning strategy

For each of the eight SPEC benchmarks of Table 4.4, 32 cycle-accurate simulations are performed on the `reference` input sets using SMARTS, as prescribed by the resolution III design described in section 4.3.2. Although the floorplan can be optimized for each of the individual benchmarks, in practice, a processor must be optimized so that it performs well over a range of benchmarks. In other words, one must generate a single floorplan for the processor that is, on average, optimal over all benchmarks. For this purpose, the CPI regression coefficients are averaged over the eight benchmarks to generate a new set of regression models that are used in the optimization process to generate a single floorplan.

Table 4.6 shows the coefficients of the average case regression model. These are normalized such that the $mean$, β_0, is 100. It can be seen that the buses from the register update unit (ruu) to the integer adders ($iadd1, iadd2, iadd3$) have the highest impact on CPI when pipelined[5]. The main reason for this is that programs typically have a high number of integer instructions such as branch operations that involve significant dependencies. The memory bus lsq_dl1 has the next highest magnitude, followed by ruu_reg. In addition, some of the factors/buses such as $itlb\_l2$ have negligible (coefficient) magnitudes, and therefore can be freely pipelined without any significant impact on CPI.

[5]However, it can also be seen in Table 4.6 that these factors have significant interactions, and these tend to further magnify the impact of pipelining the corresponding buses.

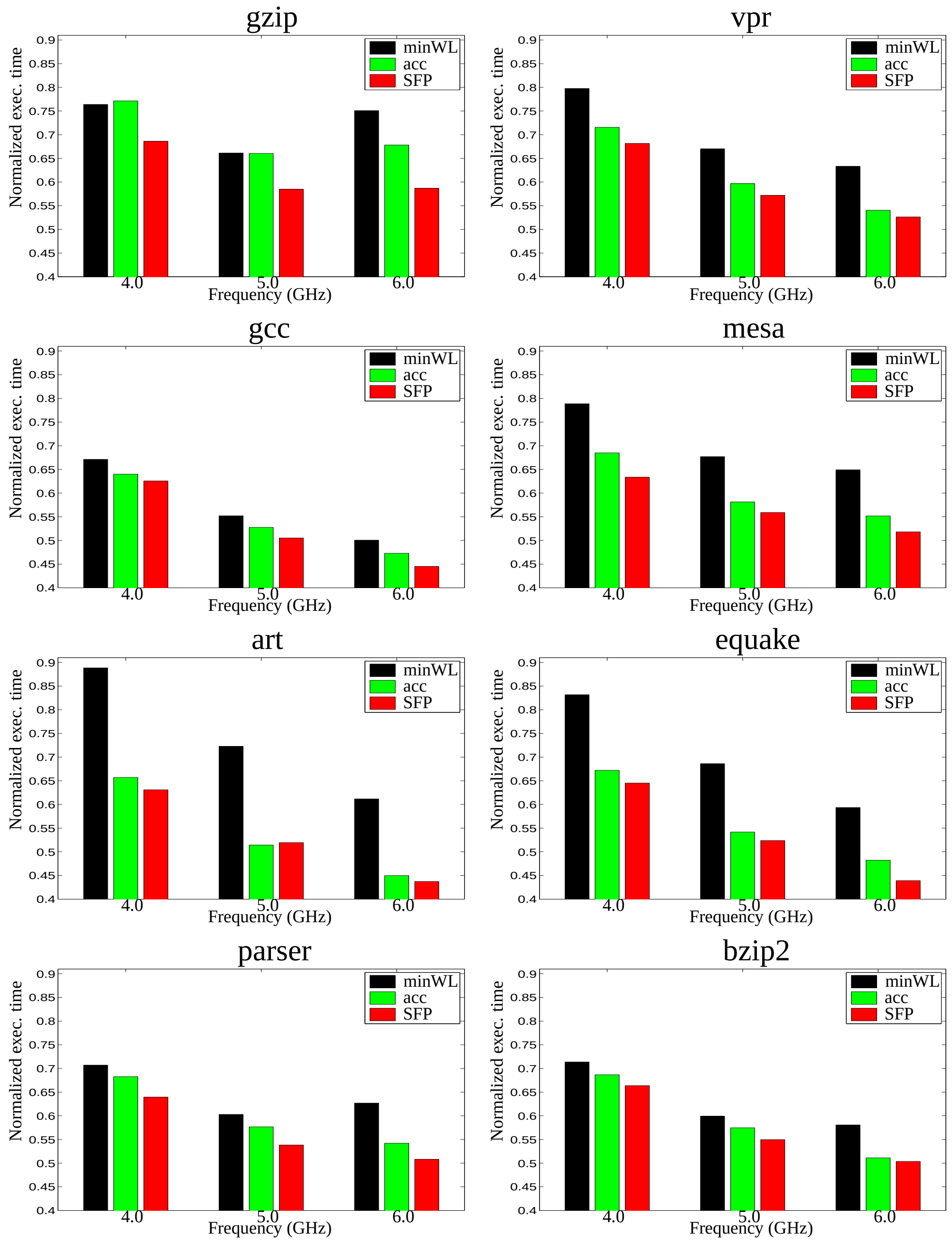


Figure 4.3: Floorplanning results for eight benchmarks for three different frequencies. The execution times are normalized to the baseline case, where wire-pipelining is employed and the frequency cannot exceed 2.4GHz.

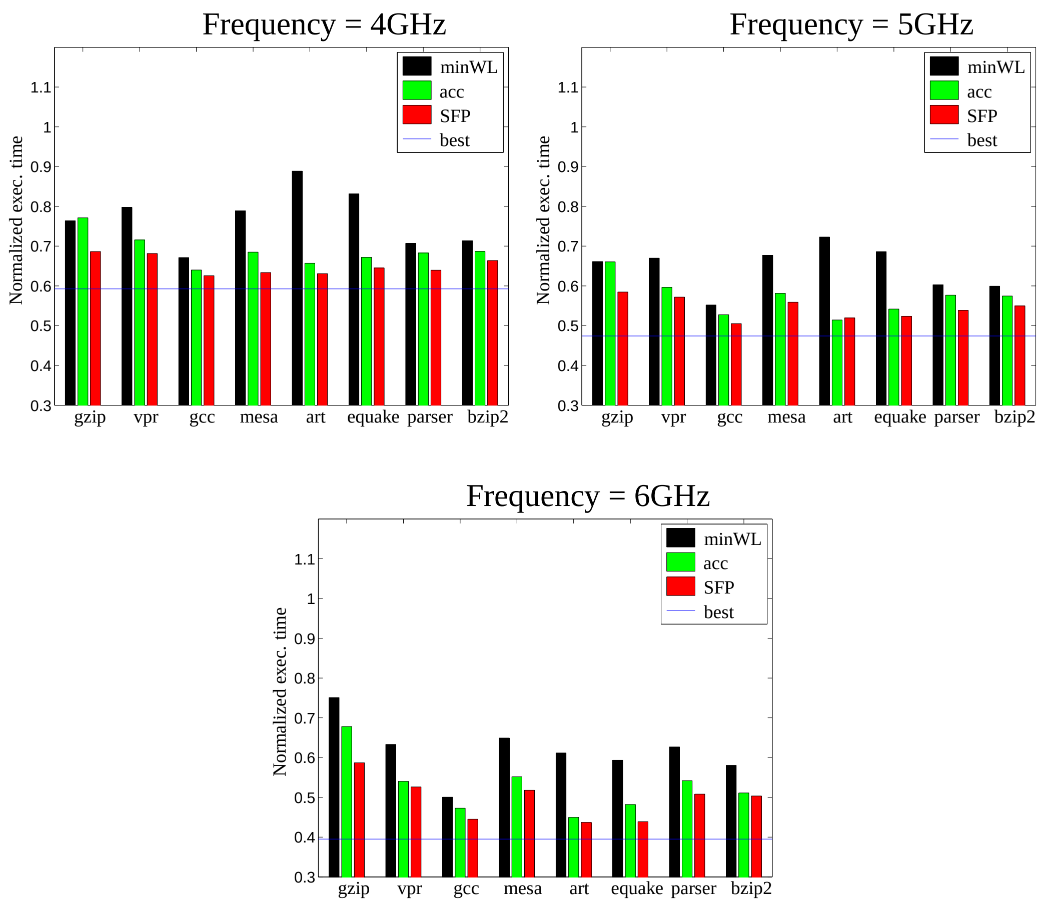


Figure 4.4: The floorplanning results of Figure 4.3, plotted for each frequency. The horizontal line in each graph represents the reduction in the clock period, over the baseline 0.42ns (corresponding to a frequency of 2.4GHz), obtained with wire-pipelining, and this represents the lower bound on the achievable execution times. However, this lower bound is not be achievable, since operating at frequencies higher than 2.4GHz makes it necessary to pipeline some of the buses, which increases the associated CPIs, thus affecting the execution times, determined using (4.8).

The regression model thus obtained is used to guide our Statistical FloorPlanner (SFP). We compare the results of SFP with those of traditional floorplanning, where the cost function shown in (4.7) includes the total wire length, instead of CPI or any microarchitecture related issue, and we refer to this floorplanner as *minWL*. In addition, we also compare our results with the access frequency-based floorplanning of [EMW$^+$04], which will be referred to as *acc* henceforth. Based on [Ekp04], we have implemented the algorithm in [EMW$^+$04] that gathers the bus access information by incorporating access counters for each bus in SimpleScalar. These access frequencies are used to weight the buses in floorplanning, and we replace the CPI term in (4.7) with the weighted sum of bus latencies.

Figure 4.3 presents the results obtained from floorplanning for the eight benchmarks. The graphs plot the execution times of the programs for three different frequencies ranging from 4GHz to 6GHz. As mentioned earlier, the baseline processor with no wire-pipelining operates at 2.4GHz. All execution times are normalized to those of this baseline processor. The bars SFP and acc represent the cases, respectively, for our proposed floorplanner and the access-ratios based floorplanning of [EMW$^+$04], where the floorplan is optimized for the general case by averaging the regression coefficients (or access-frequencies in acc) across the eight benchmarks.

Firstly, since all of the bars in the graphs are well below 1.0, the execution times are less than those obtained on the baseline processor, which indicates that wire-pipelining does increase the performance of a microarchitecture. In addition, our proposed floorplanner, SFP, as well as acc outperform minWL by a large margin for each benchmark over all frequencies. Next, SFP performs better than acc for almost all frequencies, and the execution time reductions tend to be more at higher frequencies where the amount of pipelining required is typically higher. For instance, at a frequency of 6GHz, improvements of about 16% and 11% over acc are obtained for the benchmarks gzip and equake, respectively, while these are about 11% and 6% at 4GHz. On an average, as compared to acc, SFP reduces the execution time by 6% for the eight benchmarks. In addition, it can be observed that the execution times decrease as the frequency increases for all benchmarks for both acc and SFP. However, this is not true for minWL, where in some instances, an increase in the frequency raises the execution time, such as gzip as the frequency is increased from 5 to 6GHz.

The graphs of Figure 4.4 plot the results for each frequency, as opposed to Figure 4.3 where the results are shown for each benchmark separately, to demonstrate the sensitivity of the execution times of the eight benchmarks to wire-pipelining. Similar to Figure 4.3, the y-axes of the graphs show the normalized execution times. For each frequency, the horizontal line depicts the frequency speedup achieved by employing wire-pipelining, essentially the ratio of the corresponding and baseline (2.4GHz) frequencies of 2.4GHz. This line represents the theoretical bestcase for the execution times, assuming that the circuit can still be operated at the high clock frequency without pipelining the buses, i.e., the frequency speedup translates to an equivalent reduction in the execution time. In reality, however, the bestcase may not be achievable since at least a few buses may need to be pipelined in order to operate at the high clock frequency, which increases the associated CPI. From the figure, it may be observed that all of the bars are over the corresponding horizontal lines, indicating an increase in the CPIs due to wire-pipelining. In addition, the ratios of the best and observed execution times gets higher as the frequency increases. However, there is considerable variation in the sensitivity to wire-pipelining across various benchmarks. For instance, for benchmarks such as mesa, SFP results in about 7%, 17% and 28% higher execution times than the (unachievable) bestcases, for the frequencies of 4, 5 and 6GHz, respectively. On the other hand, for gcc, the corresponding increases are lower, about 5%, 6% and 12%, respectively.

Comparison of simulation techniques

In addition to the SMARTS technique that is used to generate the results of the previous section, we perform 32 simulations, according to the resolution III design of

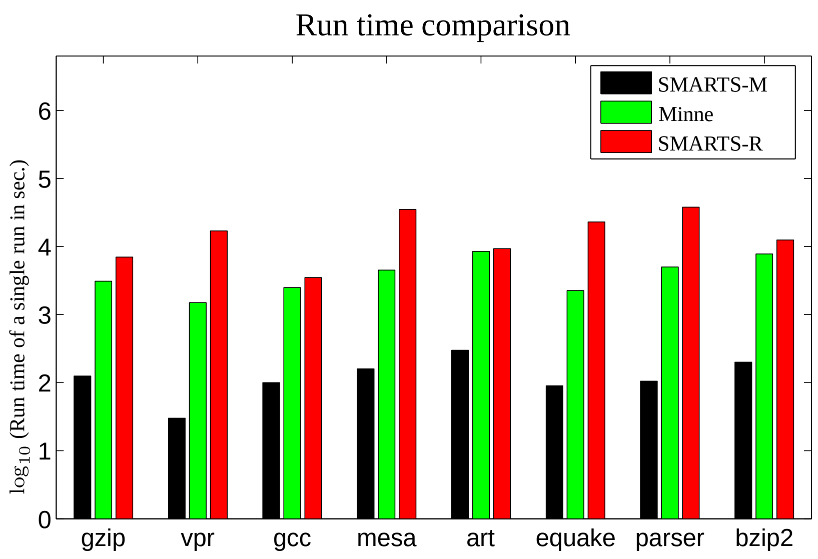


Figure 4.5: Run time comparison of a single simulation for the three techniques. The y-axis plots the run time of a single run for each technique and benchmark on a logarithmic scale.

section 4.3.2, for each of the other two techniques described in section 4.4 for every benchmark. Furthermore, the floorplan is optimized for each benchmark separately, unlike the average case that is used in the previous section, to study the impact of the techniques on individual optimizations.

The three floorplanning scenarios that are compared are labeled as shown below:

- **Minne:** MinneSPEC reduced input sets are simulated to completion.
- **SMARTS-R:** SMARTS is applied on the `reference` input sets.
- **SMARTS-M:** SMARTS is applied on the MinneSPEC input sets.

In addition, we use a common platform to compare the three cases: the evaluations are performed on the `reference` input sets, with SMARTS speeding the simulations. In doing so, we are biasing the evaluations towards the SMARTS-R technique. We note that our objective is to examine how the reduced input sets compare with the sampling techniques, not exactly to measure the accuracy of these techniques in tracking the `reference` performance. The reason behind choosing "sampling–on–`reference`" as the common framework is that sampling techniques replicate the `reference` behavior with very high accuracies, as indicated in [YKS$^+$05], and the speedups evaluated on this platform are likely to represent those observed when the `reference` input sets are run to completion.

Figure 4.5 plots the run times of a single simulation for the three techniques in logarithmic scale. It can be seen from the figure that SMARTS-R has the longest simulation times among the three techniques. However, the SMARTS-M case has simulation times that are more than two orders less than the other two approaches, specifically SMARTS-R, while the simulation times in Minne are somewhere between those of the other two techniques. As an example, for the benchmark gzip, the simulation times associated with SMARTS-M, Minne, and SMARTS-R are 125, 3100, and 7000 seconds, respectively.

Figure 4.6 presents the results obtained from floorplanning for the four scenarios described earlier in this section, for each of the eight benchmarks, and for four frequencies, 3-6GHz. For each benchmark, all execution times plotted in the graphs are normalized to that of the corresponding baseline case, where the frequency is 2.4GHz.

The graphs of the figure show that, for most benchmarks, there is not much difference in the execution times obtained for the three cases, Minne, SMARTS-R and SMARTS-M, and the differences in execution times are with in 1%. This indicates that the reduced input sets compare well with the sampling technique for the CPI-aware floorplanning problem. In fact, by employing sampling (SMARTS) on the reduced input sets (MinneSPEC), we can drastically reduce the simulation times without much loss in the performance. For instance, for the benchmark vpr, on an average, each simulation run for SMARTS-R takes about 5 hours. However, almost the same performance improvements (as seen in SMARTS-R) can be obtained when MinneSPEC reduced inputs

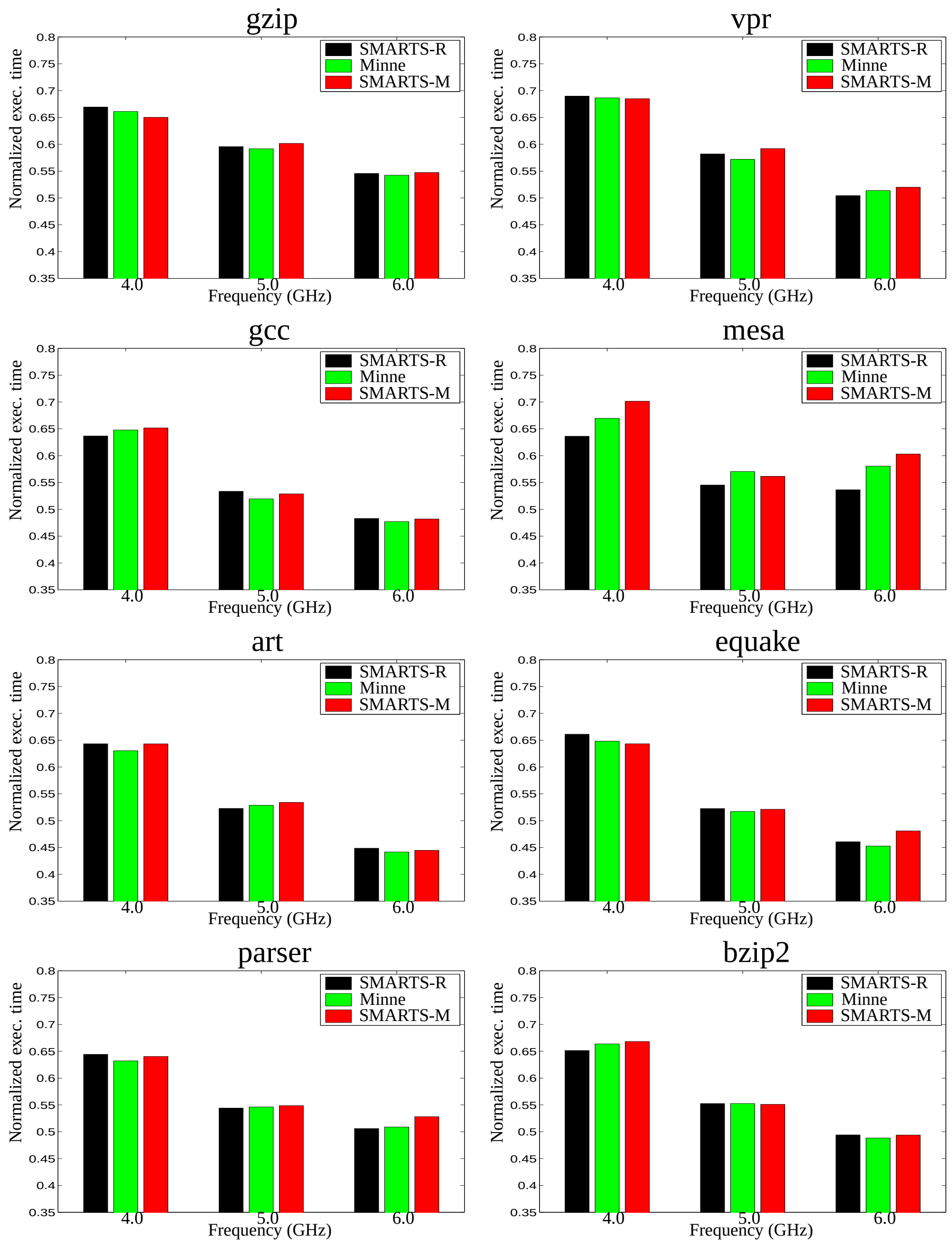


Figure 4.6: The floorplanning results of the comparison of the three simulation techniques, for eight benchmarks for three different frequencies. Just as is done in Figure 4.3, the execution times are normalized to the baseline case, where no wire-pipelining is employed and the frequency cannot exceed 2.4GHz.

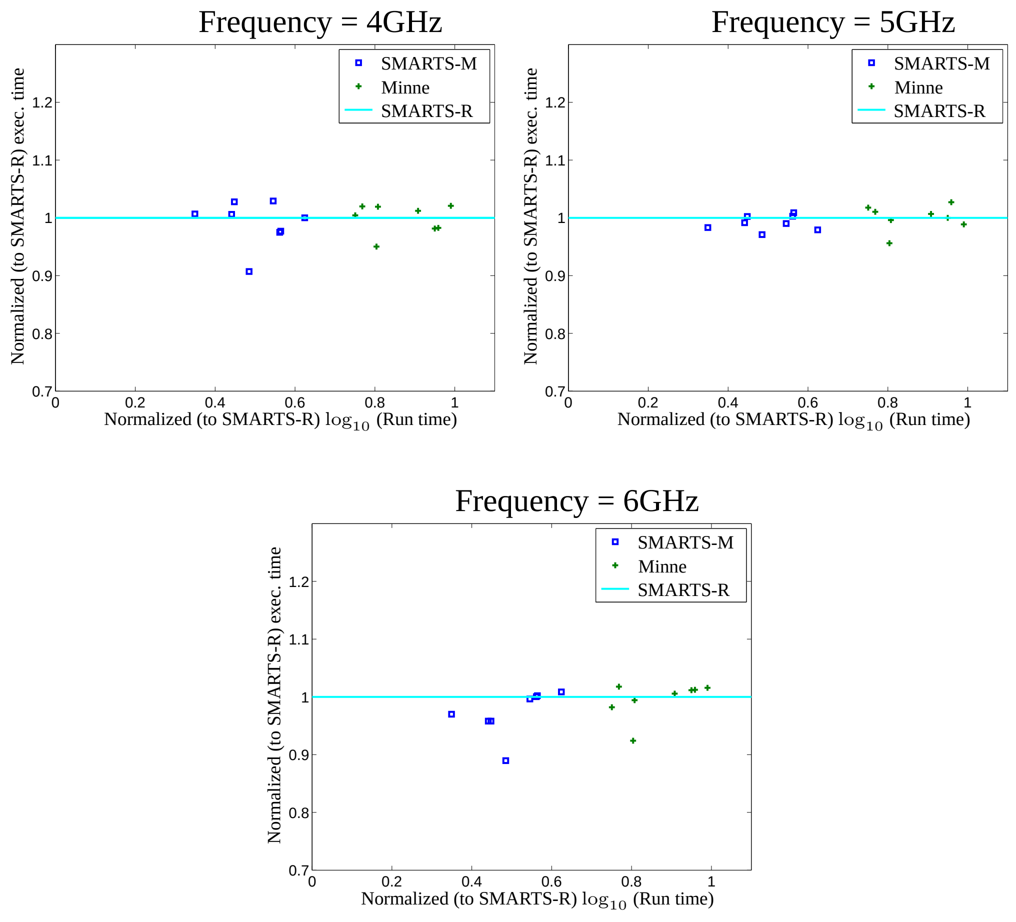


Figure 4.7: Run time vs Execution time comparison for the three techniques, at each of the three frequencies. For each benchmark, the execution times obtained using SMARTS-M and Minne are normalized to those of SMARTS-R, shown as the horizontal line in the graphs. Similarly the run times are normalized those of SMARTS-R (on a logarithmic scale) as shown on the x-axes.

are used to generate the factor/interaction weights, and in this case (Minne), the time required for each simulation is about 30 minutes, a nearly $10\times$ speedup over SMARTS-R. The simulation times can further be decreased with a negligible reduction in the performance by sampling the reduced input sets, i.e, SMARTS-M, where each simulation runs for about 30 seconds, approximately $600\times$ faster than SMARTS-R.

The graphs of Figure 4.7 provide a better picture to examine the tradeoffs between the run times and the execution times associated with the three techniques, and the results are plotted for each of the three frequencies separately, just as is done in Figure 4.4. The graphs show the run time of a single run (refer to Figure 4.5) using the three techniques for each of the eight benchmarks on the x-axes. These are normalized on the logarithmic scale to those of the SMARTS-R technique for each benchmark, where a (normalized) run time of one for a benchmark corresponds to that of SMARTS-R for that benchmark. The y-axes of the graphs plot the execution times obtained using Minne and SMARTS-M, and once again, these are normalized to those of the SMARTS-R technique for each benchmark. The horizontal line in a graph at one represents the (normalized) execution time of the SMARTS-R technique for each benchmark at the associated frequency. In addition, the 16 entries in each graph correspond to the execution times obtained using Minne and SMARTS-M for various benchmarks: one square (SMARTS-M) and one plus (Minne) for each benchmark.

It can be observed from the figure that, as expected and seen earlier in Figure 4.6, almost all of the entries (both squares and pluses) are close to the horizontal line. In addition, a fraction of the small differences seen may be attributed to random variations due to the nondeterministic nature of the floorplanner. This may also be the reason behind the fact that for some cases, Minne and SMARTS-M outperform the SMARTS-R technique, as indicated by the presence of some of the entries above the horizontal line in the graphs.

Table 4.7 shows pairwise comparisons of the three techniques in terms of the magnitudes of the regression coefficients obtained from the resolution III design of section 4.3.2 for the three techniques, SMARTS-R, SMARTS-M, and Minne. For each pairwise combination of these three, the value shown for each benchmark is the average Euclidean distance between the corresponding main and interaction regression coefficient vectors, similar to the metric used in [YLH03]. If $X = \langle x_1, \cdots, x_n \rangle$ and $Y = \langle y_1, \cdots, y_n \rangle$ are two regression (coefficient) vectors, i.e., sets of βs described in (4.6), then the average Euclidean distance between X and Y is determined as follows:

$$E_{xy} = \sqrt{\frac{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}{n}} \tag{4.9}$$

We only include the main and interaction coefficients, and omit the mean β_0, in the distance estimation. Each weight vector is normalized to 100, i.e., maximum of the coefficients (other than β_0) in each vector is 100. In such a case, the maximum bound on the average Euclidean distance is 100, with the minimum being 0. The idea of this distance metric is to observe how the regression coefficients (or factor/interaction weights) generated in the three cases compare with each other. Each value in Table 4.7 indicates how much the magnitude of a coefficient obtained using an approach differs, on an average, from the corresponding value obtained using the compared technique. For instance, for the benchmark vpr, a value of 6.73 shown in column labeled **SMARTS-R Vs Minne** indicates that, on an average, any regression coefficient obtained using Minne differs by about seven from that of the corresponding coefficient determined through SMARTS on the `reference` input sets (SMARTS-R).

- **Minne Vs SMARTS-M:** The distance is negligible for most of the benchmarks, as shown in the table, which suggests that sampling with SMARTS on the reduced input sets tracks the behavior of the whole input sets with high accuracy. This is an interesting observation, since, it shows that, for applications which employ reduced input sets, the simulations can further be speeded up by applying sampling.

- **Minne Vs SMARTS-R:** The distances are relatively higher than those seen in the **Minne Vs SMARTS-M** comparison, presumably because of change in the

Benchmark	SMARTS-R Vs Minne	SMARTS-R Vs SMARTS-M	Minne Vs SMARTS-M
gzip	4.30	4.22	0.45
vpr	6.73	6.58	2.38
mesa	26.36	27.29	2.92
art	23.81	22.87	4.79
mesa	12.78	11.35	1.65
equake	12.73	12.89	0.81
parser	9.23	9.30	0.67
bzip2	0.86	0.83	0.56

Table 4.7: Pairwise distance comparison of the three techniques. For each pair of techniques, the values shown represent the average differences between the regression coefficients of the model shown in (4.6), computed using (4.9). The maximum coefficient in each technique is normalized to 100, i.e., the maximum possible distance is 100.

input sets, in tune with the conclusions of [YKS+05]. However, other than mesa and gcc, the distances are still moderate, and it is unlikely that such moderate changes in the factor and interaction regression coefficients shift the optimal operating points by significant amounts, given the discrete nature of the cost function that is minimized in floorplanning. For mesa, the reason behind the large differences is the contrasting instruction mixes of the corresponding MinneSPEC and `reference` input sets: MinneSPEC input set for mesa has negligible floating point instruction count, while the `reference` input set has about 9% floating point instructions.

This is also the reason behind the relatively higher differences in the execution times obtained for mesa for the three techniques shown in Figure 4.6. Although the differences in the distances are also higher for gcc, it is less sensitive to wire-pipelining, as shown in Figure 4.4, where the execution times for gcc exceed the theoretical limits (horizontal lines in the graphs) only by small amounts. We believe this is a reason for the lack of impact of such differences on the floorplanning results.

- **SMARTS-R Vs SMARTS-M:** The distances follow the same trend as seen in the **SMARTS-R Vs Minne** comparison above, since both Minne and SMARTS-M use reduced input sets. The distances are slightly lower than those observed in **SMARTS-R Vs SMARTS**, however. This may be because both SMARTS-R and SMARTS-M employ SMARTS, the inaccuracies, however small they are, associated with the SMARTS technique creep into both of them, having identical effects.

Therefore, due to the similarities in the results obtained, there are no accuracy-run time tradeoffs that can be explored. From Figure 4.7, it is clear that, SMARTS-M, while achieving the same performance speedups as the other two techniques, represents the best approach with least simulation times, in the order of a few hundreds of seconds. Given the small run times of SMARTS-M, it may also be possible to employ a more accurate or a higher resolution design than that it is described in section 4.3.2. However, as will be demonstrated in the next section, it is unlikely that a higher resolution design can result in a better floorplanning solution.

4.5.4 Validation of the resolution III design

As pointed out in Section 4.3.2, the underlying assumption in using the resolution III design of Table 4.3 to generate the results in Section 4.5.3 is that all of the interactions except those shown in (4.3) are negligible. To test the validity of this assumption, we utilize a two-level resolution V design, where all of the main and two-factor interactions can be estimated, and compare the resultant regression models with those of (4.6). However, such a design requires a minimum of 512 simulations, which makes the implementation impractical if the simulations have long run times, even with speed up techniques such as **SMARTS-R** and **Minne** described in the previous section.

We therefore utilize the **SMARTS-M** technique, which has short simulation times and is shown to be a good substitute for the `reference` input sets for CPI-aware floor-

Benchmark	**Res. III Vs Res. V**
gzip	1.60
vpr	2.32
gcc	4.61
mesa	9.14
art	4.82
equake	5.31
parser	2.35
bzip2	2.66

Table 4.8: Distance comparison of the Resolution III and V designs for the **SMARTS-M** technique. The values shown represent the average differences between the regression coefficients of the model shown in (4.6), computed using (4.9). Just as done in Table 4.7, the maximum coefficient in each technique is normalized to 100, i.e., the maximum possible distance is 100.

planning in the Section 4.5.3, for the comparison. Table 4.8 lists, for each benchmark, the distance between the corresponding regression coefficients of (4.6) obtained using the resolution III and V designs, computed using the metric shown in (4.9). We note that only the terms of (4.6), i.e., the main effects and the eight interaction effects of (4.3), are compared in generating the distance, although a model with many more interaction terms than in (4.6) can be constructed using a resolution V design. Since the coefficients, particularly of the main effects, obtained with the resolution III design are aliased with interactions that are assumed to be negligible, a comparison with the corresponding coefficients determined using the resolution V design will be sufficient to check if the assumption is valid.

Specifically, if the coefficients are equal (or almost equal), then the interactions must be negligible. It can be seen from the table that the distances are small, indicating that

Benchmark	**Res. III design**		**Res. V design**	
	Avg. err. (%)	**Max. err. (%)**	**Avg. err. (%)**	**Max. err. (%)**
gzip	1.434	6.553	0.621	3.130
vpr	2.678	-9.911	1.628	-11.171
gcc	2.514	18.134	3.063	-20.959
mesa	5.492	24.444	2.896	-9.352
art	4.024	14.187	0.941	5.679
equake	3.309	-21.762	0.811	3.867
parser	1.591	7.397	0.863	3.034
bzip2	1.787	8.438	0.869	4.029

Table 4.9: Error in the estimated CPIs as compared to the simulated numbers over a set of 512 combinations of the minimum and the maximum values for the bus latencies. The absolute values of the individual errors are used in the calculations of the average errors.

the coefficients compare well, and this validates the assumption of negligible interaction effects. Furthermore, the numbers are much smaller than those seen in Table 4.7, which presents a pairwise comparison of the three simulation techniques described in Section 4.4. It is unlikely that the small increase in the accuracies, when the resolution V design is used, can significantly impact the floorplanning optimization process.

In addition, we compare the estimated CPI values with the simulated numbers for the set of 512 latency combinations that are part of the resolution V design. Table 4.9 shows the maximum and the absolute average of the error in estimation for both resolution III and V designs, for each benchmark. On the whole, the models do well in predicting the CPIs, except for a few outliers. For instance, for bzip2, resolution III and V designs result in errors of about 1.7% and 0.9%, respectively, on an average. It can also be observed that the resolution V design does better than resolution III in estimation, the differences are not significant however. This can partly be explained by the fact that the combinations for which the CPIs are estimated to obtain the errors actually correspond to the 512 simulations prescribed by the resolution V design, i.e., the same set of combinations are used in the resolution V case to generate the regression functions.

Since the 512 simulations are actually part of a two-level design, the combinations only have the extreme (minimum and maximum) values for the bus latencies. As stated in Section 4.3.2, the assumption is that the response (CPI) has a linear dependence on the bus latencies. To check if there are significant nonlinearities, we generate a set of 1000 random combinations and perform simulations for each benchmark. The combinations contain any values between the range of bus latencies, i.e., between the two levels of the resolution III and V designs addressed in the previous paragraphs. Table 4.10 shows the average and maximum of the estimated errors for both the resolution III and V cases. It can be seen that the errors have higher magnitudes than those of Table 4.9, which suggests the presence of nonlinearities. However, the reduction in the accuracy is not significant. For instance, the average error increases from about 1% in Table 4.9 to about 4%, which is not big enough to have a significant impact. It may also be noticed that the two cases, resolution III and V, result in almost identical errors, unlike the comparison presented in Table 4.9.

4.6 Conclusion

This chapter proposed a methodology based on a statistical design of experiments approach to identify the CPI critical buses in a microarchitecture. Using this approach, the essence of the large exponential search space is captured using a small number of simulations, linear in the number of buses. In addition, our approach also considers the impact of interactions of the bus latencies. The performance impact of bus latencies is quantified by constructing a CPI regression model, and the approach is applied at the

Benchmark	Res. III design		Res. V design	
	Avg. err. (%)	Max. err. (%)	Avg. err. (%)	Max. err. (%)
gzip	6.343	-20.962	6.425	-21.940
vpr	6.314	-20.889	5.312	-21.077
gcc	3.011	20.958	3.696	-14.930
mesa	6.108	19.854	5.548	-20.173
art	8.287	-23.583	8.004	-22.581
equake	6.182	-21.801	6.291	-21.945
parser	2.467	-9.193	2.468	10.258
bzip2	4.387	-18.392	4.467	-18.371

Table 4.10: Error in the estimated CPIs as compared to the simulated numbers over a set of 1000 different combinations of random values for the bus latencies. The absolute values of the individual errors are used in the calculations of the average errors.

floorplanning level. A comparison of the results with an existing approach, which uses bus access frequencies to weight the criticality of the bus latencies, indicates that our proposed methodology produces better performance for a number of frequencies.

In addition, we compared three techniques, namely, SMARTS, MinneSPEC reduced input sets, and a hybrid of both, that can speed up the simulations of the `reference` inputs of the SPEC benchmarks for the CPI-aware floorplanning problem. We use a distance metric to compare the regression models generated using the three techniques in the simulation methodology. This comparison suggests that MinneSPEC sets and SMARTS generate significantly different sets of regression coefficients. However, this variation in the magnitudes did not affect that subsequent optimization, and the performance improvements seen in both cases are almost identical. Therefore, there is no correlation between the contrasting regression models generated and the actual delivered performance. The best technique for this optimization is the hybrid version of both, i.e., SMARTS on the MinneSPEC reduced input sets, and this case drastically quickens the simulation process by several orders, besides generating high quality designs.

Chapter 5

Floorplanning of a Pentium architecture

In Chapter 4, the proposed strategy for throughput-aware floorplanning is experimented on the DLX architecture using the SimpleScalar simulator. In this chapter, we extend the technique for the widely used Intel Pentium architecture (P6) [HP96]. For this purpose, we use Asim [EAB$^+$02], a cycle-accurate simulation framework developed at Intel Corporation. The objective of this study is to validate the efficacy of our proposed floorplanning methodology in an industry simulation framework.

The rest of the chapter is organized as follows. Section 5.1 overviews the Pentium architecture and the simulator Asim, while Section 5.2 presents the floorplanning methodology. The results of the simulation methodology and floorplanning are demonstrated in Section 5.3 and we conclude the chapter in Section 5.4.

5.1 Preliminaries

5.1.1 Pentium architecture

Pentium architecture (P6) models an out–of–order superscalar machine that implements a Complex Instruction Set Computing (CISC) instruction set. However, during execution, programmer visible instructions are split into equal sized operations called μops, which gives the appearance of a RISC machine at this level. The word arrangement is little–endian and like the DLX architecture employed in Chapter 4, there are separate Level 1 caches for instructions and data and a single unified cache at Level 2. The P6 architecture has been utilized for several processors developed by Intel Corporation such as Pentium III and Pentium M.

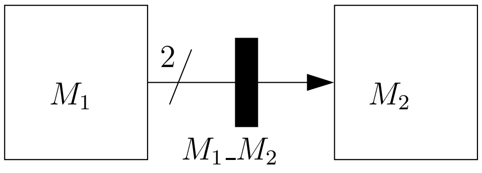


Figure 5.1: A bus $M_1\_M_2$, from module M_1 to module M_2. The bandwidth of the bus is two and the latency is one, as indicated by the single dark rectangle on the bus.

5.1.2 Overview of Asim

Asim provides a cycle-accurate simulation framework for microprocessor performance modeling. The basic components of the simulator are modules and feeders, where the modules represent the functional blocks of a processor and the feeders provide the instructions for simulations. The framework also provides a template for multiple implementations of the modules, thereby allowing the capability of simulating different architectures. In addition, Asim includes a port network that models the buses connecting the blocks of a processor. Such a representation enables modeling of a microarchitecture system much closer to the actual hardware, than SimpleScalar, where there is no explicit way of communication between the functional blocks, and thus making it easier to account for the additional bus latencies in the simulator.

For instance, consider any two modules (blocks) of a processor, M_1 and M_2, and suppose there is a bus of width two from M_1 to M_2, say, $M_1\_M_2$, with a latency of one, as shown in Figure 5.1. The bus can be declared as a write port in the module definition of M_1 while it becomes a read port in M_2. Both declarations use the same identifier "$M_1\_M_2$" and this binds the read and write ports. The bandwidth of $M_1\_M_2$ is set in the input block, i.e., M_1, and the destination module, M_2, sets the latency, which in the example of Figure 5.1 is one. In such a scenario, changing the latency of a bus can be easily done just by modifying the latency of the corresponding read port in the simulator code.

5.2 Simulation methodology

5.2.1 Microarchitecture blocks and buses

Due to the complexity and the high number of blocks and buses associated with the Pentium architecture, we decided to approach the floorplanning problem in phases, first applying the methodology on subsystems of the processor before a comprehensive implementation for the entire microarchitecture. In the initial phase reported in this thesis, we consider the backend execution and memory cores shown in Figure 5.2. The blocks rs, rob, int and fp represent the backend, and ldb, stb, dcu and $dtlb$ form the memory unit of the architecture. Comparing with the blocks of DLX processor shown in Figure 4.2, we have the load store queue, lsq of DLX split into separate queues for loads and stores, shown as ldb and stb, respectively, in Figure 5.2. Similarly, there are separate queuing systems for the reorder buffer, which maintains the instruction pool dispatched from the frontend, and reservation stations, where instructions are scheduled and issued to execution and memory units, and these are labeled as rob and rs in Figure 5.2. It can be noted that the rob and rs are combined to form the register update unit, ruu, in the DLX architecture of Figure 4.2. The blocks int and fp correspond to the integer and floating point execution clusters comprising adders and multipliers. Finally, dcu and $dtlb$ are the data cache and TLB of the microprocessor.

There are 16 buses (ports) in this subsystem, as shown in Figure 5.2. The buses include the instruction issue ports from rs, the cache access and the data forwarding paths.

5.2.2 Bus latency modeling

As noted in Section 5.1.2, the buses (of Figure 5.2) are modeled as ports in Asim, and the latencies can be varied by calling appropriate member functions of the port objects. However, arbitrarily increasing the latencies can pose functional correctness issues, as

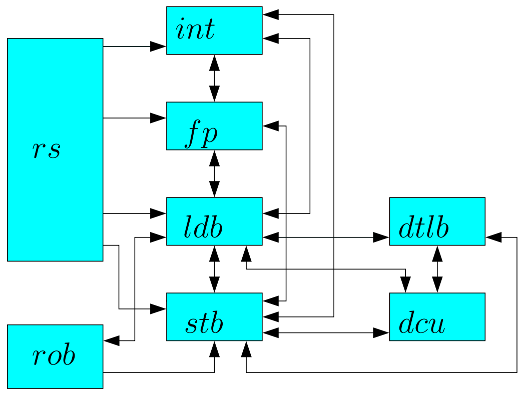


Figure 5.2: The backend execution and memory functional blocks of the P6 architecture. The blocks rs, rob, int and fp represent the backend, and ldb, stb, dcu and $dtlb$ form the memory units of the architecture. The arrows represent the ports (buses), 16 of them, connecting the blocks.

observed in Chapter 2, which addresses the correctness problems at circuit-level, and Chapter 4, where the latency modeling issues for the DLX architecture are dealt. There are several dependencies associated with bus latencies that need to be considered for correct execution, such as:

- The load buffer, ldb, monitors and retains a few previous store instructions issued by the reservation rs as part of the wake-up logic that unblocks load instructions that are previously stopped for possible Read after Write (RAW) hazards. It turns out that the number of the previous stores retained is dependent upon the latency of the port between rs and ldb.

- When the load buffer, ldb, dispatches a load instruction to the data cache and tlb for memory access, a warning signal is sent to the scheduler of rs to speculatively issue any dependent instructions of the load operation. If a cache hit occurs for the load operation, then the execution continues. However, the speculatively dispatched dependent instructions must be squashed if the memory access for the load instruction results in a cache miss or a page fault. If the latency of the signal is less than the cache hit latency, the dependent instructions can be issued before the data is available, even when a cache hit occurs. In such a scenario, the instructions are always squashed irrespective of a hit of miss in the cache. The existing mechanism works for a miss but not for a hit. For correct execution, the latency of the warning signal from ldb to rs must be greater than the cache hit latency, which indicates a dependency between the two paths.

The dependencies are modeled and the additional bus latencies are parameterized in the simulator. Each of the 16 buses represents a factor that can influence the throughput, IPC, of the microarchitecture.

5.2.3 Design of experiments strategy

Although the resolution III design of Table 4.3 is accurate enough for the DLX architecture, as explained in Section 4.5.4, preliminary experimentation indicated that there may be several significant, particularly two-factor, interactions. Estimation of all of the two-factor interaction effects requires a design with a higher resolution, such as V, as noted in Section 4.5.4. However, such a design involves a high number of experiments: for 16 factors, a typical two-level resolution V design has a size of 512 [DM70], like the design used in Section 4.5.4.

We instead employ a two-step approach, shown in Figure 5.3, that can reduce the number of simulations to a reasonable level. The rationale of this approach is that, in general, not all factors may significantly impact the response of a system. It has been a common practice to use a simple, screening, design to filter out the factors that have negligible impact and construct a better and detailed model in the second phase for the significant ones. For instance, the work of [FP00] uses this two-step technique for optimizing electromagnetic devices. The efficacy of such an approach lies in the concept of *factor sparsity*, i.e., the fewer the number of significant factors, the lower is the number of experiments required.

The objective of our proposed two-step approach is to construct a performance model that includes all of the two-factor interaction terms along with the main effects, for the identified few significant factors. We use the statistical software JMP [SLC01] for this purpose. The software can be used to generate a variety of statistical designs, perform statistical significance tests, and construct regression models using techniques such as Analysis of Variance (ANOVA) [Lil00].

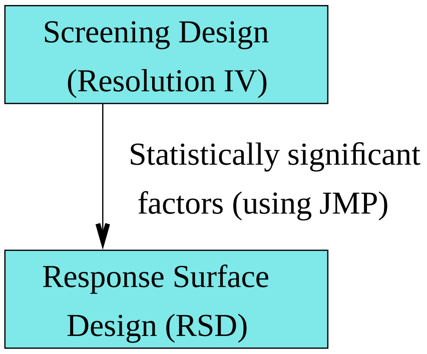

Screening Design (Resolution IV)
Statistically significant factors (using JMP)
Response Surface Design (RSD)

Figure 5.3: The outline of the two-step Design of Experiments approach. The statistical significance tests are performed using the software JMP.

The first step of the approach involves a screening design to identify the significant and insignificant factors. For this purpose, we use a two-level resolution IV design, where the main effects are completely separated from the two-factor interaction effects, as described in Section 3.5.2. A resolution III design, such as that of Table 4.3, may not be effective since the magnitude of a main effect can be masked by potentially significant two-factor interaction terms that are aliased with the main effect. For 16 factors, a standard resolution IV has a size of 64[1]. A two-level resolution IV design can be obtained by simply folding over the entries of a resolution III design such as that of Table 4.3. For instance, complementing all the entries of Table 4.3 gives a set of 32 additional distinct simulations, which when combined with the original 32 rows of the table gives a resolution IV design (of size 64).

[1]Alternatively, a Plackett and Burman (PB) design of resolution IV, which has a size of 40, can be used.

The screening design is applied on the set of benchmarks, and the results obtained from the simulations are used to find important factors. The identification of significant factors is done using a systematic procedure that employs ANOVA and the F-test, a test for determining statistical significance, as described in Section 3.5.3. The test involves indexing the appropriate F-distribution with the F-ratio obtained for each factor, using the simulation results. The result of the look up is a p-value, which is an indicator of the importance of the factor: the lower the value, the higher is the significance. As noted in Section 3.5.3, the general rule of thumb followed is to classify all factors with a p-value higher than 0.05 (5%) as statistically insignificant, and we do this in our work.

The factors that are found to be statistically insignificant are ignored and a Response Surface Design (RSD) is applied for the retained significant factors. The design is of resolution V, therefore all of the two-factor interactions involving the (retained) factors can be estimated. In addition, it also permits the estimation of the quadratic components of the main effects, there by enabling the modeling of nonlinearities in the impact of the factors on the throughput. In other words, the throughput of the microprocessor is captured as a response surface involving the identified important factors. Since, a three-level design is required to estimate the quadratic components as mentioned in Section 3.5.2, the design includes a third level for each factor, and this corresponds to the mean of the minimum and maximum values of the latency range of the factors. We use JMP to generate the design and perform simulations for each benchmark as prescribed by the generated RSD.

5.2.4 Regression model

The result of the simulation methodology outlined in the previous section is a regression function for CPI, where the variables are the significant factors obtained using the screening design. If, say, n out of the original 16 factors are found to be important, then the following regression function can be constructed for CPI from the results of the simulations obtained using the RSD generated using JMP.

$$CPI \;=\; \beta_0 + \sum_{i=1}^{n} \beta_i \cdot x_i + \sum_{i=1}^{n} \beta_{i.i} \cdot x_i^2 + \sum_{\forall i<j\leq n} \beta_{i.j} \cdot x_i \cdot x_j \qquad (5.1)$$

Where the x_is represent the encodings of the latencies of the n buses, computed along the same lines of those in (4.6). The term β_0 represents the mean or intercept of the function, each β_i and $\beta_{i.i}$ correspond to the coefficients of the linear and quadratic components, respectively, of the main effect of factor i, while each $\beta_{i.j}$ is the magnitude of interaction between factors i and j. Overall, there are $2n$ main effect terms and $\binom{n}{2}$ two-factor interaction terms in the model.

5.3 Results

5.3.1 Experimental Setup

We use a set of 24 SPEC 2000 benchmarks for the experimentation. For the floorplanning step, we use an Intel internal floorplanning tool that is also a simulated annealing based implementation similar to Parquet employed in Chapter 4 for DLX, and we choose a frequency of 4GHz for the experimentation. Owing to the slow simulation speed of Asim, we only simulate each benchmark for one million instructions. The idea is to first validate the efficacy of the simulation methodology of Section 5.2 for a small portion of the benchmarks before graduating to utilizing complete simulations. Each simulation, of size one million instructions, takes about 10-20 minutes to complete. For the latency range for each factor, we choose zero and two clock cycles as the minimum and maximum values, and therefore one for the mean latency value which is used in the RSD of Section 5.2.3, besides the minimum and the maximum, to estimate the quadratic components of the main effects. It is unlikely that the latencies exceed more than two clock cycles since the methodology is applied on a relatively smaller portion of the processor, i.e., the backend and memory cores, as compared to the whole area of the DLX processor in Chapter 4.

5.3.2 Results of the two-step methodology

First, 64 simulations are performed for each of the 24 benchmarks, as part of the two-level resolution IV screening design generated using JMP. The results of the F-test indicate that 10 of the 16 factors can be ignored in building the regression model. Figure 5.4 shows the six significant factors (buses), where the dotted line indicates the factor which has the highest impact on the throughput. In addition, it turns out that almost all of the benchmarks have the same set of critical factors. It can also be observed that most of the buses are related to the instruction issue and forwarding paths, and the data forwarding bus between int and stb is the most critical of these buses.

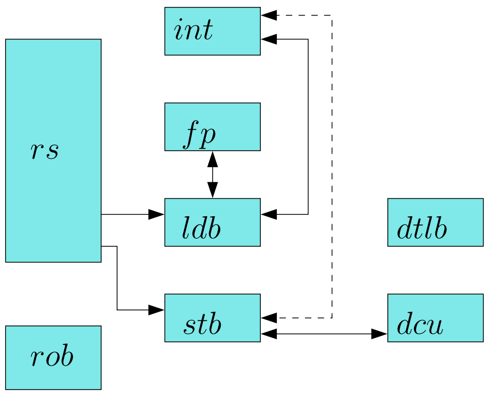


Figure 5.4: The six statistically significant buses of the architecture. The dotted line between int and stb represents the most important of the six buses.

The next step involves the implementation of the Response Surface Design (RSD) for the six factors of Figure 5.4. This design is generated using JMP, and requires 45 simulations per benchmark. Therefore, the two-step approach of Section 5.2.3 prescribes a total of 109 (64 for the screening design and 45 for RSD) simulations per benchmark. It can be noted that this is significantly less than the 512 simulations per benchmark required in a potential one-step approach that utilizes a resolution V design for all of the 16 factors.

The results of the simulations are then used to generate regression models, one for each benchmark, and a single regression function is constructed by averaging over the 24 benchmarks: Each regression coefficient of (5.1) is obtained by averaging the corresponding coefficients of the 24 regression models.

Two of the coefficients of the regression function, corresponding to the main effects of the buses between rs and ldb, and rs and stb have negative magnitudes. This is an unexpected result since it indicates that inserting additional flip-flops on the two buses can actually increase the throughput, i.e., reduce the number of clock cycles of execution. To understand such an anomalous behavior, we analyze and compare the block activity traces and instruction issue patterns across several relevant bus latency configurations. The analysis suggests that the complex scheduling and the out-of-order issuing schemes employed in the P6 architecture proved to be significantly sensitive to the bus latencies. Increasing the latencies sometimes can change the issue patterns in a beneficial way that will reduce the number of clock cycles required for execution. Although SimpleScalar models an out-of-order execution core, such an unexpected trend is not observed, and the system behavior is much more linear, i.e., the throughput reduces with insertion of latencies on any of the buses, compared to the out-of-order execution core of the Pentium processor.

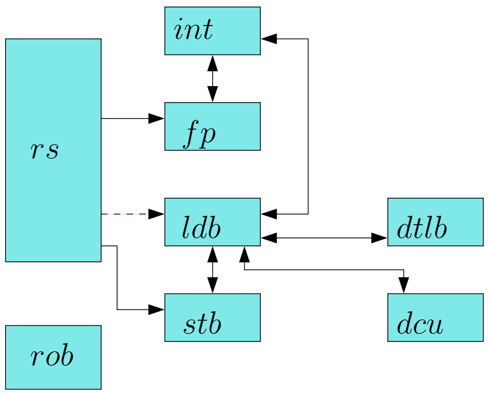


Figure 5.5: The eight statistically significant buses of the in-order execution version of the P6 architecture. The dotted line between rs and ldb represents the most important of the eight buses.

For an even better understanding of the Pentium system behavior, we implement an in-order version of the execution core of the P6 architecture with in the Asim framework. The goal is to identify if the observed nonlinearities are due to the out-of-order nature of the execution. In general, an in-order machine has a lot less unpredictability in instruction scheduling as compared to an out-of-order processor. In particular, changing bus latencies tends to impact the scheduling algorithms of an in-order core with a lesser magnitude than those of an out-of-order system due to the relatively lower flexibility in issuing instructions: Even if an instruction is ready to be issued, it can only be scheduled after all of the previously queued instructions are dispatched.

We implement the same two-step simulation methodology utilized for the out-of-order core, for the in-order execution version. The results of the screening design indicate that there are eight statistically significant factors, and these are shown in Figure 5.5. It can be observed that most of the buses are related to the forward paths, similar to those seen for the out-of-order case in Figure 5.4. The most important factor for the in-order machine is the bus between the reservation station rs and the load buffer ldb.

For eight buses, the response surface design has a minimum size of 80, compared to 45 for six buses in the out-of-order scenario. Therefore, the total number of simulations increases to 144 per benchmark for the in-order processor. All of the regression coefficients obtained through the application of the RSD have positive magnitudes, indicating a normally expected behavior. The results therefore suggest that the scheduling schemes are less sensitive to bus latency variations in the in-order version of the P6 architecture, compared to the original out-of-order core.

5.3.3 Validation of the simulation methodology

We validate the regression models obtained using the two-step approach of Figure 5.3 for the in-order and out-of-order versions over a random set of 256 bus latency configurations. For this purpose, 256 simulations are performed and the results are com-

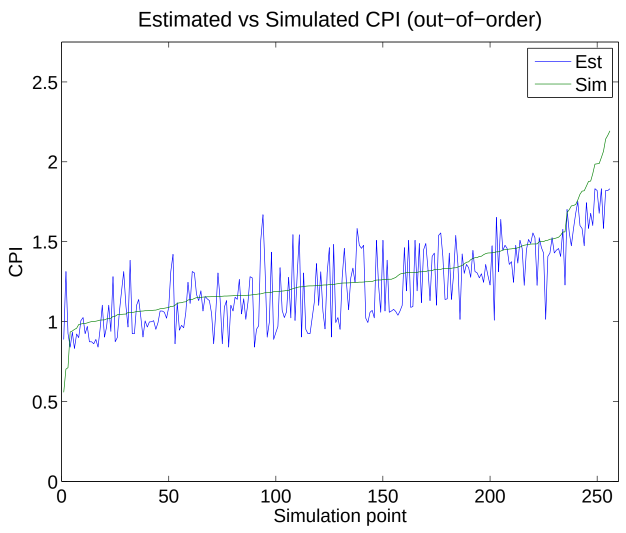


Figure 5.6: Comparison of the estimated and simulated results for a randomly generated set of 256 simulations for the benchmark ammp, for the out-of-order P6 machine. The CPIs are sorted with respect to the simulated results.

pared with those estimated from the models, and we do this for one benchmark in this work. It is, however, a good idea to validate the models for each benchmark, as is done for the DLX architecture in Section 4.5.4, although the huge computation cost, 256 simulations per benchmark, introduces feasibility concerns.

Figures 5.6 and 5.7 show the validation results for the SPEC benchmark “ammp” for the out-of-order and in-order scenarios, respectively. The graphs plot the estimated and simulated CPIs of the 256 simulations, sorted with respect to the simulated numbers. The figures indicate that the percentage differences between the estimated and simulated results is lower in the in-order scenario than the out-of-order case. The average absolute errors are about 6% and 12% for the in-order and out-of-order cases, respectively, while the corresponding maximum errors are 28% and 86%, respectively.. Overall, the graphs do not exhibit any particular trends, the estimated values exceed and fall behind the simulated values arbitrarily. In addition, it can be observed that, as expected, the in-order CPIs are much higher than those of the out-of-order case.

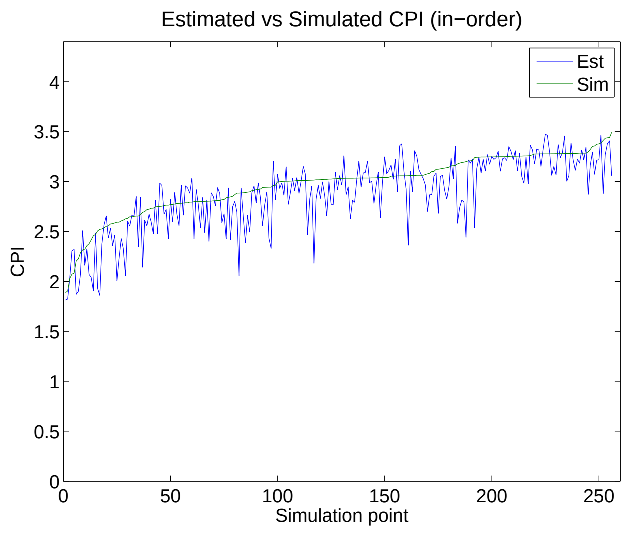


Figure 5.7: Comparison of the estimated and simulated results for a randomly generated set of 256 simulations for the benchmark ammp, for the in-order P6 machine. The CPIs are sorted with respect to the simulated results.

5.3.4 Floorplanning results

The floorplanner uses the CPI regression models constructed from the two-step simulation methodology of Section 5.2.3 in the cost function of the simulated annealing based engine. The cost function is similar to that of (4.7), and we compare our Design Of Experiments (DOE) based floorplanning approach with the regular floorplanning methodology, where total wire length minimization is considered instead of CPI. Figures 5.8 and 5.9 compare the floorplanning results obtained using the proposed DOE approach and the regular approach, for the out-of-order and in-order scenarios, respectively. Each plot has 24 points, one each for a benchmark.

The graphs show that our proposed approach results in lower CPIs, therefore better throughputs, than the regular floorplanner, in both the out-of-order and in-order cases. However, the improvements are much higher in the in-order case than out-of-order, which is not surprising given better validation results seen for the in-order scenario in

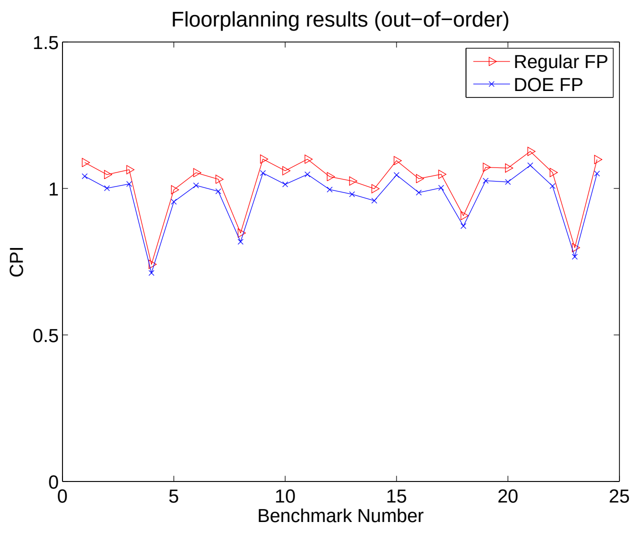


Figure 5.8: Floorplanning results for the out-of-order case: The CPIs obtained using our proposed approach are compared to those computed through regular, minimum wire length, floorplanning.

Section 5.3.3. On an average, a reduction of about 11% in the CPI is observed for the in-order case, while it is about 5% for the out-of-order machine.

5.4 Conclusion

This chapter extends the Design of Experiments (DOE) based floorplanning methodology of Chapter 4 to the Pentium architecture (P6) that implements an out-of-order machine. For this purpose, we use Asim, a simulation framework of Intel corporation. In this thesis, we focus only on the backend and memory bus latencies in the performance modeling. We use a two-step approach for simulation methodology, where insignificant factors are filtered out in the first step using a screening design of resolution IV. A detailed Response Surface Design (RSD) is applied on the remaining significant factors to construct a quadratic model that includes all of the two-factor interactions and quadratic components of the main effects. Simulation results indicate unexpected behavior where

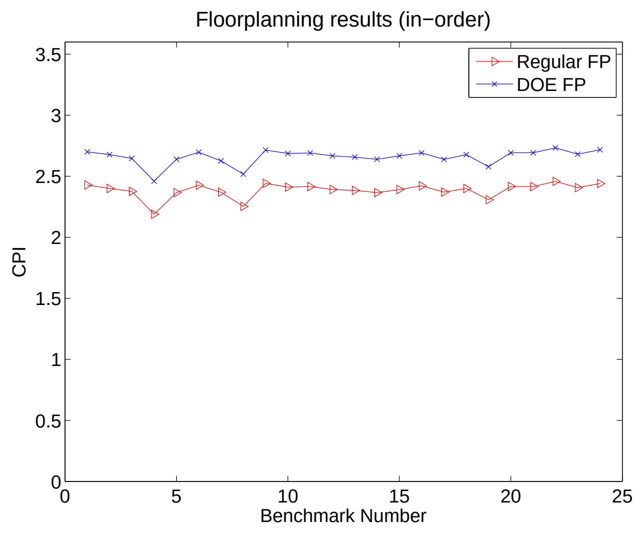


Figure 5.9: Floorplanning results for the in-order case: The CPIs obtained using our proposed approach are compared to those computed through regular, minimum wire length, floorplanning.

throughput increases (or CPI reduces) when some bus latencies increase. Our analysis suggests that that the reason for this is that the complex scheduling schemes of the out-of-order execution core are highly sensitive to changes in bus latencies.

For a better understanding, we implemented an in-order version of the execution core and applied the two-step methodology on this new processor. The results indicate predictable behavior, unlike the out-of-order case. The floorplanning results show that the proposed floorplanning approach outperforms a regular, minimum wire length, methodology, and the approach works better for the in-order scenario than the out-of-order version.

Chapter 6

Thermally-aware floorplanning

The microarchitecture-aware floorplanning problem addressed in Chapters 4 and 5 specifically focuses on throughput/performance optimization. As pointed out in section 1.3.2, floorplanning can also impact the operating temperature of a chip, through the vertical and horizontal heat transfer mechanisms between the components, the spreader and the heat sink of the chip. In this chapter, we extend the methodology to include temperature in the objectives of floorplanning.

The chapter is arranged as follows. Section 6.1 introduces the problem and presents some previous work. The importance of modeling the power-throughput interaction is addressed in Section 6.2, while 6.3 explains the concept of temperature estimation. Section 6.4 lists the thermal metrics considered in this work and the overall flow of the floorplanning methodology is outlined in Section 6.5. We conclude the chapter in Section 6.7 after presenting some experimental results in Section 6.6.

6.1 Introduction

Due to rapid increases in on-chip power and integration densities, operating temperatures have become an important concern in high performance integrated circuits in nanometer technologies. A high temperature can affect the reliability of a circuit, thus reducing its lifetime [SABR04], through phenomena such as electromigration and Negative Temperature Bias Instability (NBTI). With every process generation, circuit performance becomes more sensitive to thermal effects due to the decreasing limits on the maximum junction temperature [Sem01]. In addition, the temperature dependence of the leakage power results in an undesirable positive feedback, commonly referred to as *thermal runaway*, which could even lead to catastrophic chip failures. While advanced [GBCH01] packaging solutions can result in enhanced heat removal capabilities, the costs associated with these solutions are typically prohibitive. Therefore, it is important to develop temperature-conscious design techniques that alleviate on-chip thermal problems.

As noted in Section 1.3.2, on-chip temperature distributions depend not only on the total power dissipation, but also on the spatial distribution of the power sources and the material properties of the medium that permit vertical and horizontal heat transfer in a chip. Physical design methods, such as floorplanning and placement, can impact the thermal profile of a chip by altering the spatial distribution of power sources, indicating a scope for improvement through better heat spreading that evens the temperature distribution on the chip. In addition, physical design optimizations can complement other thermal- and power-aware design [BCB$^+$00] techniques implemented at a higher, architecture level such as Dynamic Thermal Management (DTM) [BM01].

The topic of thermally-aware floorplanning/placement has attracted some attention in the last few years, both at the circuit and microarchitecture levels. The primary difference between circuit and architecture level treatments is the level of knowledge about the spatial distribution of power. At the architectural level, the circuit is defined only in terms of large functional blocks and coarse estimates of power are available, while at the circuit level [HXV$^+$05,CWZ05,GS03], the power consumptions of individual macro cells or blocks are all well known, and more accurate estimations are possible. However, there are many more flexibilities at the architectural level that permit significant design changes that reduce the overall power and temperature distribution.

This thesis focuses on the interactions between microarchitecture design and physical design, in particular, floorplanning, to explore performance-temperature tradeoffs. As seen in Chapter 4, the choice of a floorplan can significantly affect the throughput/performance of a processor due to the presence of wires with multicycle delays. Such fluctuations in the CPI can change the activity patterns of the blocks, resulting in variations in the power densities. In other words, floorplanning can affect the temperature profile not only through heat spreading but also because the spatial and temporal distributions of power densities vary due to wire-pipelining. A good floorplanning strategy must therefore consider such interaction between CPI and power (and hence temperature) and jointly optimize both the performance and temperature objectives.

A few recent works [HKM05, WYYC05, SSH+05, EHB+04] propose techniques for thermally-aware microarchitecture floorplanning. While these indicate a welcome progress, they suffer from two drawbacks:

- They do not model the CPI-power interaction in the floorplanning step and assume that the block power consumptions are layout independent. Specifically, the power densities that are obtained for a zero-bus-latency scenario, which typically represents the worst case for dynamic power (and the best case for the throughput, IPC), are assumed to be valid for all floorplans irrespective of the amount of pipelining required by the buses, and this can result in overestimation of the temperature.

- They attempt to minimize the steady-state temperature of a chip. However, steady-state can only occur when the power dissipation is constant, which may not be true in general since programs tend to exhibit phases of varying activities [IM03]. In such a case, a transient modeling [WC02] provides a better picture of the thermal behavior of the chip: the execution times of the standard benchmarks that are used in simulations, such as SPEC [Hen00] utilized in this thesis, are typically in the range of seconds, which are significantly larger than typical thermal time constants, making it imperative to model transients. In addition, transient modeling also captures an accurate depiction of the dependence of leakage current on temperature.

  A better strategy may be to focus on minimizing the peak transient temperature over the entire execution time of a program. Furthermore, besides the peak temperature, it is useful to capture the temporal average of the temperature distribution, since many reliability mechanisms depend on this.

Although some of the previous approaches do consider the temperature transients, the emphasis is on modeling the impact of temperature on leakage power, only a small portion of the execution time is considered for analysis, and the goal of floorplanning is to minimize the steady-state temperature.

We propose a methodology for multiobjective microarchitecture floorplanning, where the objectives are minimizing the temperature (both average and peak), based on transient analysis, and maximizing the performance (or minimizing the CPI). Our approach models the impact of wire-pipelining (i.e., changes in the CPI, on power densities in the floorplanning step) and temperature-leakage power dependencies. For the purposes of a complete transient analysis that considers the entire execution times of the programs, we use a larger timestep than those employed in the limited-time analyses of [HKM05, WYYC05, SSH$^+$05, EHB$^+$04]. Since the floorplanning that we address involves big microarchitecture blocks, which have larger time constants than ordinary cells, the temperatures change at a slow rate, in which case, a large timestep, which reduces the analysis time by a tremendous amount, can be chosen without much loss in accuracy.

6.2 Dependence between power and throughput

Figure 6.1 plots the instantaneous dynamic power consumptions of a block, averaged over every 10000 clock cycles, for two different latency configurations, “c1” and “c2”, that result in CPIs of 0.91 and 1.11, respectively, for the SPEC benchmark, “gcc”. The two instances represent two distinct floorplans of the DLX architecture shown in Figure 4.2 and Table 4.1. It can be observed from the figure that there is considerable variation in the dissipated power between the two configurations. The case “c2”, which has a higher CPI, takes more clock cycles than "c1", a program event is likely to occur in "c2" at a time later than in "c1". Moreover, the magnitude of such a "time shift" may vary across different events, it is even possible that a particular instruction of a program is executed in "c2" at an earlier time than in "c1". Such variations in the block activities can significantly impact the power consumption profile of the processor, as seen in Figure 6.1. It is therefore important to consider this dependence between the throughput and dynamic power in the floorplanning optimization.

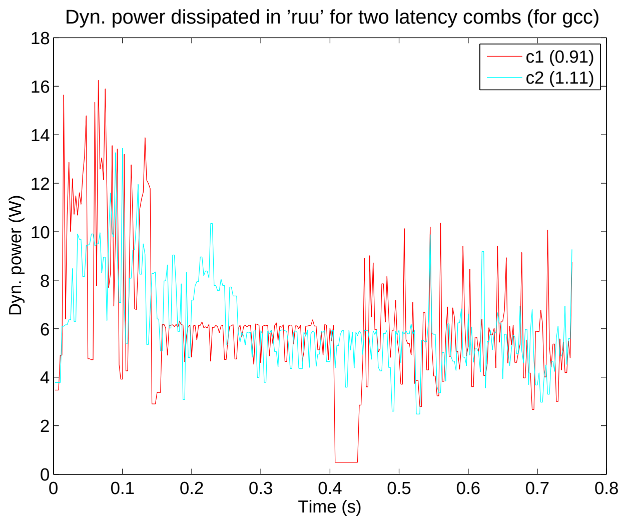


Figure 6.1: The instantaneous dynamic power consumption of the register update unit (ruu) block for two different bus latency configurations, each corresponding to a different floorplan of the same microarchitecture.

6.3 Thermal estimation

A key component of a thermally-aware design methodology is a framework to estimate the temperature distribution of a chip. In the thermal analysis context, a chip can be viewed as a multi-layered grid network, essentially a discretization of the chip geometry, where the nodes of the network correspond to the centers of the grids, and the connections between the nodes represent the heat flow paths in the chip. In such a set-up, the power sources $\vec{P}$ are located at the nodes of the network and based on the duality of electricity and heat transfer, the temperature distribution of the network is governed by the following differential equation:

$$\vec{C} \cdot \frac{d\vec{T}}{dt} + G \cdot \vec{T} = \vec{P} \tag{6.1}$$

where G is the thermal conductance matrix of the network, $\vec{T}$ is the temperature distribution of the nodes of the network. The first term on the LHS of (6.1) represents the transient behavior of the temperature, with $\vec{C}$ modeling the thermal capacitances. Several techniques for thermal analysis have been proposed in the past, some of which can be found in [ZGS06].

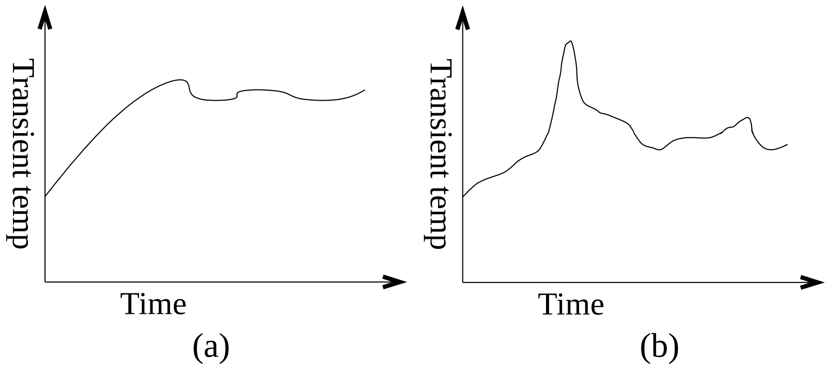


Figure 6.2: The results of two transient analyses of a circuit under two different implementations.

6.4 Average temperature

Figure 6.2 shows two possible transient scenarios for a circuit, where the maximum transient temperature of the circuit is plotted against time elapsed. Although the curve of Figure 6.2(a) has a lower peak than that of Figure 6.2(b), Figure 6.2(b) offers a better average, where the curve is below that of Figure 6.2(a) for a majority of the time. As noted in [SABR04], the reliability or mean time to failure (MTTF) decreases exponentially with temperature. Therefore, Figure 6.2(b) may represent a higher reliable case than Figure 6.2(a). In such a scenario, attempting to minimize the peak temperature can result in suboptimal thermal profiles. Nevertheless, a higher peak, seen in Figure 6.2(b), is not desirable due to the constraints it places on the package hardware. Therefore, a better approach may be to consider both the peak and the average temperatures in the optimization objectives, and we do this in our floorplanning methodology.

6.5 Floorplanning flow

Figure 6.3 shows the flow of the proposed temperature-aware microarchitecture floorplanning methodology. It can be observed that the flow is an extension of the methodology depicted by Figure 4.1 in Chapter 4 to include temperature in the floorplanning objectives. The approach accepts a microarchitecture block configuration, a set of buses, benchmarks and a target frequency as inputs and generates a floorplan of the blocks that is both optimal in both CPI and temperature.

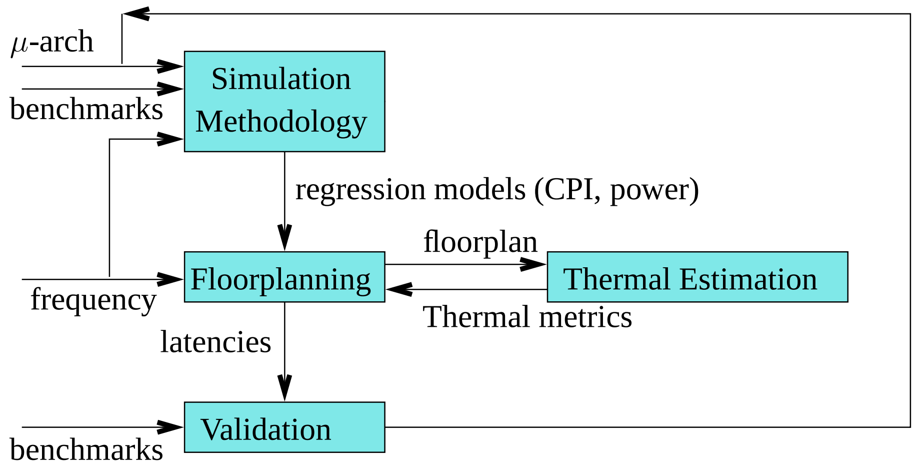


Figure 6.3: Thermal-aware floorplanning: design flow.

Unlike [$\mathrm{SSH}^+05, \mathrm{EHB}^+04$] and also in the throughput-aware floorplanning problem addressed in Chapter 4, where the purpose of the simulations is to characterize the variations in the CPI in terms of changes in the bus latencies, the objective of the simulation strategy of Figure 6.3 is to model the variations in both the throughput and the power densities, and thus capture the CPI-power dependence. The variations are encapsulated in the form of regression functions, similar to those of (4.6), with the bus latencies as variables, both for CPI and power.

The floorplanner, based on a simulated annealing (SA) framework, uses the regression models to optimize a cost function that is similar to that of Section 4.3.3. However, besides the CPI, it also includes the thermal terms, both the peak and average temperatures, as described in section 6.4.

After every SA move, the floorplanner estimates the block power densities from the regression models and passes them along with the corresponding floorplan to the thermal simulator, which in turn returns the thermal metrics that are part of the cost function. The performance and thermal profile of the resultant layout can then be determined from cycle-accurate simulations. In addition, the entire design flow of Figure 6.3 may be repeated for several microarchitectural block configurations to identify the optimal configuration.

For the purpose of simulations, we use SimpleScalar that is augmented with the Wattch [BTM00] technique for power estimation. In addition, we utilize the same two-level resolution III design of Table 4.3 that is used in Chapter 4 for the simulation strategy . Furthermore, as both power and CPI depend on the same set of variables, i.e., bus latencies, a single design can be used to characterize both responses.

The thermally-aware floorplanning approaches of [$\mathrm{SSH}^+05, \mathrm{EHB}^+04$], although do not model the dependence of power on bus latencies, propose simulation strategies to capture the throughput impact of bus latencies. The method of [SSH^+05] constructs linear regression models using simulations by varying each latency independently, whereas [EHB^+04], which is an extension of the CPI-aware floorplanning work of [EMW$^+$04] that is compared in Section 4.5.3, uses latency-independent models to capture the CPI/IPC variations. While these may work well for CPI since a reasonably accurate relative ordering of variables is sufficient as shown in Section 4.5.3[1], such one-at-a-time approaches may not effectively track absolute variations, required in the case of power, as compared to the DOE approach [Czi99] used in this work.

The reason for the requirement of "absoluteness" is that the power and temperature may not have a perfect correlation [SSH$^+$03], and power-criticality does not necessarily imply temperature-criticality. This lack of fidelity[2], coupled with the dependence of leakage current on temperature, indicates that any error in power estimation can result in significant inaccuracies in the temperature computations.

Reducing simulation times

To speed up the simulations, we utilize SMARTS, one of the techniques compared in Chapter 4, which works well both for throughput and power/energy, particularly for the SPEC benchmarks. In this way, we reuse the CPI regression models obtained using the **SMARTS-R** technique of Section 4.5.3.

Power/CPI regression models

As mentioned in Section 4.4, the SMARTS technique involves fastforwarding program segments between successive samples chosen for detailed simulation. However, the transient modeling requires that the block power densities be collected periodically for every timestep. For this, we extrapolate the power data collected for each sample for the succeeding fastforwarded portion. While we do not offer a proof, the concept of periodic sampling is inherently based on this assumption, and there is empirical evidence

[1]Section 4.5.3 however shows that our proposed DOE based approach outperforms the technique of [EMW$^+$04]

[2]A well known case where the property of fidelity holds is Elmore delay modeling: although the estimated delays may be inaccurate, the metric accurately tracks the variations in the delays.

that it works well at least for average power/energy estimation [WWFH03].

The total execution time obtained from a simulation is then segmented into slots of size equal to the transient analysis timestep. Therefore, the data collected from the simulation can be arranged as an array P indexed by the timestep and the block number, i.e., the entry $P(a, b)$ of the array corresponds to the power consumption of block b (one of the 17 blocks of Figure 4.2) during timestep a. Since 32 simulations performed (per benchmark), there are 32 such tables. For each entry $P(a, b)$ (per benchmark), a regression model is constructed from the 32 values [Mon00], based on least-squares approximation, where the variables are the bus latencies. Equation (4.6), essentially the same equation listed in (4.6), shows one such a model, constructed to estimate the power dissipation at entry $P(a, b)$, where β_is represent the regression coefficients computed from the 32 values obtained for the correspond entry (a, b).

$$\begin{aligned} x_i &= -1 + \left(\frac{2 \cdot l_i}{\min(i) + \max(i)} \right), \quad 1 \leq i \leq 22 \\ P(a, b) &= \beta_0 + \sum_{i=1}^{22} \beta_i \cdot x_i + \sum_{(ij) \in \mathcal{I}} \beta_{ij} \cdot x_i \cdot x_j + \\ & \quad \sum_{(ijk) \in \mathcal{I}} \beta_{ijk} \cdot x_i \cdot x_j \cdot x_k \end{aligned} \tag{6.2}$$

A CPI regression model is similarly constructed for each benchmark from the statistics gathered from the 32 simulations. In addition, although we construct separate regression functions for CPI and power, since the associated variables are the same, a direct relation between the power and the CPI estimates can be obtained by composition of the regression functions.

6.5.1 Temperature estimation

We use HotSpot [SSH$^+$03] in this work for thermal analysis. In this approach, the nodes of the multi-layered thermal network described in section 6.3 are the centers of the blocks of the microarchitecture. The tool also provides a framework for transient modeling, and accepts a floorplan, the length of the timestep, and the block power dissipations averaged over each timestep as inputs. The differential equation (6.1) is solved at each timestep to estimate the new set of temperatures (with the initial conditions being those of the previous timestep). The leakage power component of the succeeding timestep can then be updated using the new temperatures.

Choice of timestep

In general, the smaller the timestep, the higher is the accuracy of the transient analysis. It is clearly impractical to perform the analysis for every clock cycle of execution, and the authors of HotSpot suggest a size of about 10000 clock cycles at a frequency of 3GHz, i.e., a timestep of about $3.3\mu s$. Although this reduces the analysis time by a significant factor, it still makes it prohibitive to incorporate transient analysis into the iterative scheme of the floorplanning step, where thousands of floorplans are evaluated.

To solve this issue, we choose an interval of one million clock cycles, which amounts to about a few hundreds of microseconds for gigahertz frequencies, and this can possibly affect the accuracy of the computations. However, since the focus of the optimizations involves relatively larger microarchitecture blocks (than the macro cells considered in circuit level optimizations), the thermal RC constants tend to be higher, typically in the range of tens of milliseconds, and this indicates a minimal loss of accuracy since each time constant still involves a high number of timesteps. For instance, ruu, a medium sized block of the microarchitecture of Figure 4.2, has a time constant of about 120ms. As noted in [SSH$^+$03], the temperatures rise slowly, and it takes more than 100,000 clock cycles to observe an increase of as small as 0.1°C in the temperature. In addition, we use a single iteration to solve the differential equation of (6.1) during each timestep of the analysis.

6.5.2 Floorplanning cost function

The floorplanner, PARQUET, uses the power and CPI regression models built out of the simulation methodology described in section 4.3.2 in the cost function.

The cost function C is a weighted sum of, besides the chip area ($Area$) and the aspect ratio (AR), the average (T_{avg}), and the peak (T_{peak}) transient temperatures, as shown below:

$$C = W_1 \cdot Area + W_2 \cdot AR + W_3 \cdot CPI + W_4 \cdot (T_{avg} + T_{peak}) \tag{6.3}$$

where the Ws represent the relative weights of the optimization terms. If N_t is the number of timesteps in the transient analysis and T_i is the maximum of the block temperatures at timestep i, the average and the peak temperatures are determined as follows:

$$T_{avg} = \frac{1}{N_t} \sum_i T_i \text{ and } T_{peak} = \max_i T_i \quad (i = 1, 2, \cdots, N_t)$$

6.6 Experimentation

6.6.1 Experimental set up

We use the same set of benchmarks that are utilized for validating the throughput-aware floorplanning strategy of Chapter 4, shown in Table 4.4. In addition, just as done in Chapter 4, only the chip core that also includes the L1 caches is considered during floorplanning, and the L2 cache is wrapped around the core floorplan, just as is done in [SSH$^+$05] and Alpha 21362 [Ban98]. We choose a frequency of 4GHz for our experiments, and therefore, a timestep of 250μs.

For each of the eight SPEC benchmarks of Table 4.4, 32 cycle-accurate simulations are performed, as prescribed by the resolution III design of Table 4.3. We also generate a single floorplan for the processor that is, on average, optimal over all benchmarks. For this purpose, the CPI and power regression coefficients are averaged over the eight benchmarks to generate a new set of regression models that are used in the optimization process to generate a single floorplan.

We integrate HotSpot with Wattch to enable thermal analysis during simulations. *Although we use SMARTS to speed up the simulation strategy of section 6.5, detailed cycle-accurate simulations, without fastforwarding any program portions, for the entire execution times of the benchmarks are performed for validating the floorplanning solutions.* In addition, we use a relatively smaller timestep of 10000 clock cycles, as compared to that of 1000000 cycles used during optimization, for transient analysis, i.e., the power data are averaged over every 10000 clock cycles and are provided to the HotSpot solver to determine the set of temperatures.

We compare our proposed thermal floorplanning technique with two other approaches. The long run times of the simulations is the main obstacle that limits the number of comparisons that can be made. The floorplanners compared are listed below:

- **cpiFP:** IPC/CPI only floorplanning, the cost function of the floorplanning does not consider any thermal issues.

- **therFP:** Our proposed temperature-aware floorplanning, where the cost includes CPI and both the average and peak transient temperatures, along with the core area and aspect ratio.

- **skadFP:** A temperature-aware floorplanning approach based on [SSH$^+$05]: the block power densities are assumed to be independent of the bus latencies. In addition, the cost includes only the peak transient temperature, along with the CPI, area and aspect ratio[3].

[3]We choose to include the peak transient temperature in our implementation of [SSH$^+$05] for convenience. Moreover, although the original implementation attempts to minimize the steady-state temperature, the authors use peak transient temperature as a metric of their validation process.

For **therFP** and **skadFP**, we choose a weight of 0.4 for both CPI and temperature, and 0.1 for area and aspect ratio, i.e., $w_1 = w_2 = 0.1, w_3 = w_4 = 0.4$ in (6.3). For the CPI-only floorplanner **cpiFP**, we have $w_1 = w_2 = 0.1, w_3 = 0.8, w_4 = 0$. The idea is to provide a greater emphasis on the primary issues, the CPI and the temperature, while still attempting to limit the total area.

6.6.2 Impact of initial temperature

A key issue in transient modeling is the setting of the initial temperature, which serves as the reference point of the analysis. It is possible that the transients of a chip converge to a steady-state irrespective of the initial ambient conditions. However, the steady-state is likely to occur after a significantly long transient phase, particularly for microprocessors which have large time constants, and the nature of this phase can be affected by the initial temperature. The impact is further aggravated by the mutual dependence between the leakage power and the temperature: if the processor begins execution at a high temperature, high leakage will be seen, which may drive up the transients.

However, since floorplanning focuses on optimizing the heat transfer mechanisms of the chip through appropriately spreading the “hot spots” across the chip, it is likely that the optimization is not significantly impacted by the choice of initial setting. While it may still be a good idea to optimize over a range of possible initial temperatures, it requires multiple transient analysis evaluations, one for each temperature, for each choice considered and this blows up the simulated annealing runtime.

In this work, we perform the floorplanning optimization at a single initial temperature of 40°C. We assume this choice reflects the commonly observed ambient conditions.

Although only one initial temperature is utilized in our floorplanning strategy, we analyze the impact of variations in the initial conditions on the temperature profiles of the floorplans obtained using the three optimization approaches. Specifically, we

Case	**Core WS (%)**	**Core AR**
cpiFP	5.33	1.15
skadFP	7.60	1.02
therFP	6.21	1.03

Table 6.1: Comparison of white space (WS) and aspect ratio (AR) for the three floorplanners.

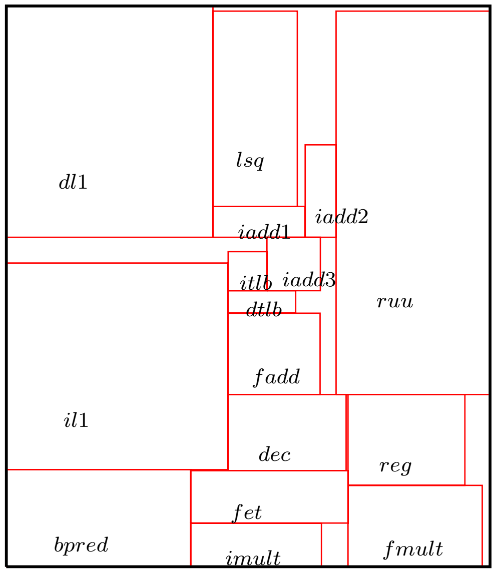
capture the transients of the three floorplans for a number of initial temperatures, ranging from 40°, the setting used in the floorplanning step, to a high temperature of 120°. To this purpose, for each floorplan, we collect the block dynamic power dissipations averaged every timestep (10000 cycles) for all of the benchmarks[4]. Using these traces, transient analysis is performed at a number of initial temperatures, and we compare the temperature metrics obtained at each of the temperatures.

6.6.3 Results

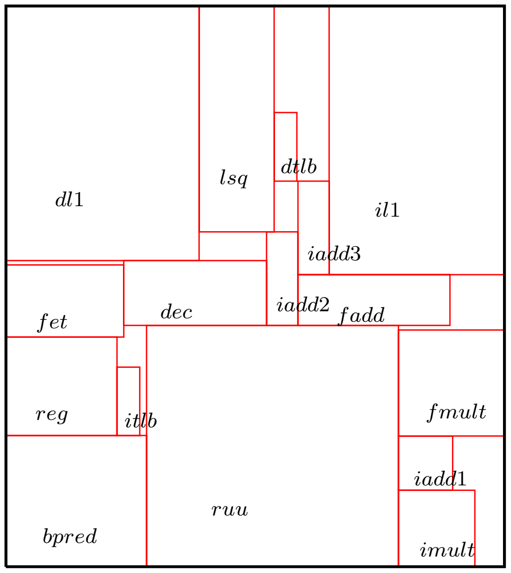
The cores of the floorplans obtained with the three approaches, namely, cpiFP, therFP, and skadFP, are shown in Figure 6.4. The L2 cache, $l2$, not shown in the figures, is wrapped around the cores to complete the floorplans. The white spaces (WS) and the aspect ratios (AR) of the floorplans obtained using the three approaches, shown in Table 6.1, imply that all of the three result only in a small increase in the area. For instance, a core WS of about 6% in **therFP** indicates an overall increase of 1.5% in the chip area (equivalent to 2.03cm^2). Besides, both **skadFP** and **therFP** produce floorplans of almost perfect AR.

Figures 6.5, 6.6 and 6.7 plot the instantaneous temperatures obtained for the three floorplanning scenarios **therFP, skadFP** and **cpiFP**, respectively, at four different initial temperatures, 40, 60, 80 and 120°C.

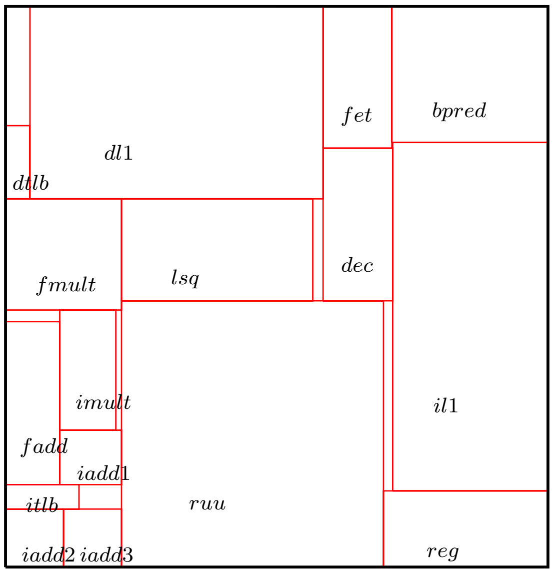
[4]We remind that the dynamic power can vary across floorplans, if the bus latencies differ.



(a) cpiFP



(b) therFP



(c) skadFP

Figure 6.4: The cores, which exclude the L2 cache, of the floorplans obtained using the three approaches.

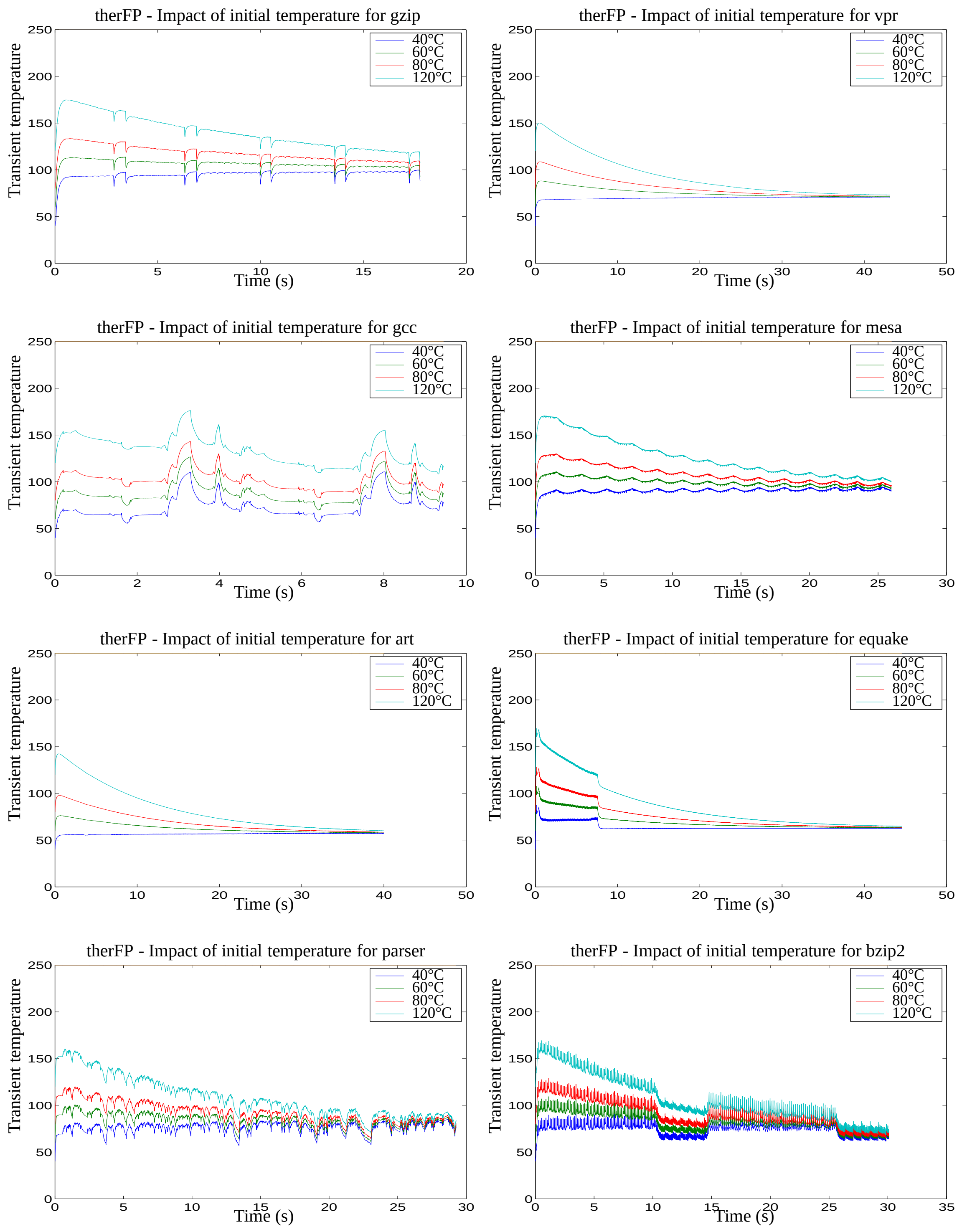


Figure 6.5: Transient profiles for the floorplan obtained using our proposed floorplanner, therFP, at four different initial conditions.

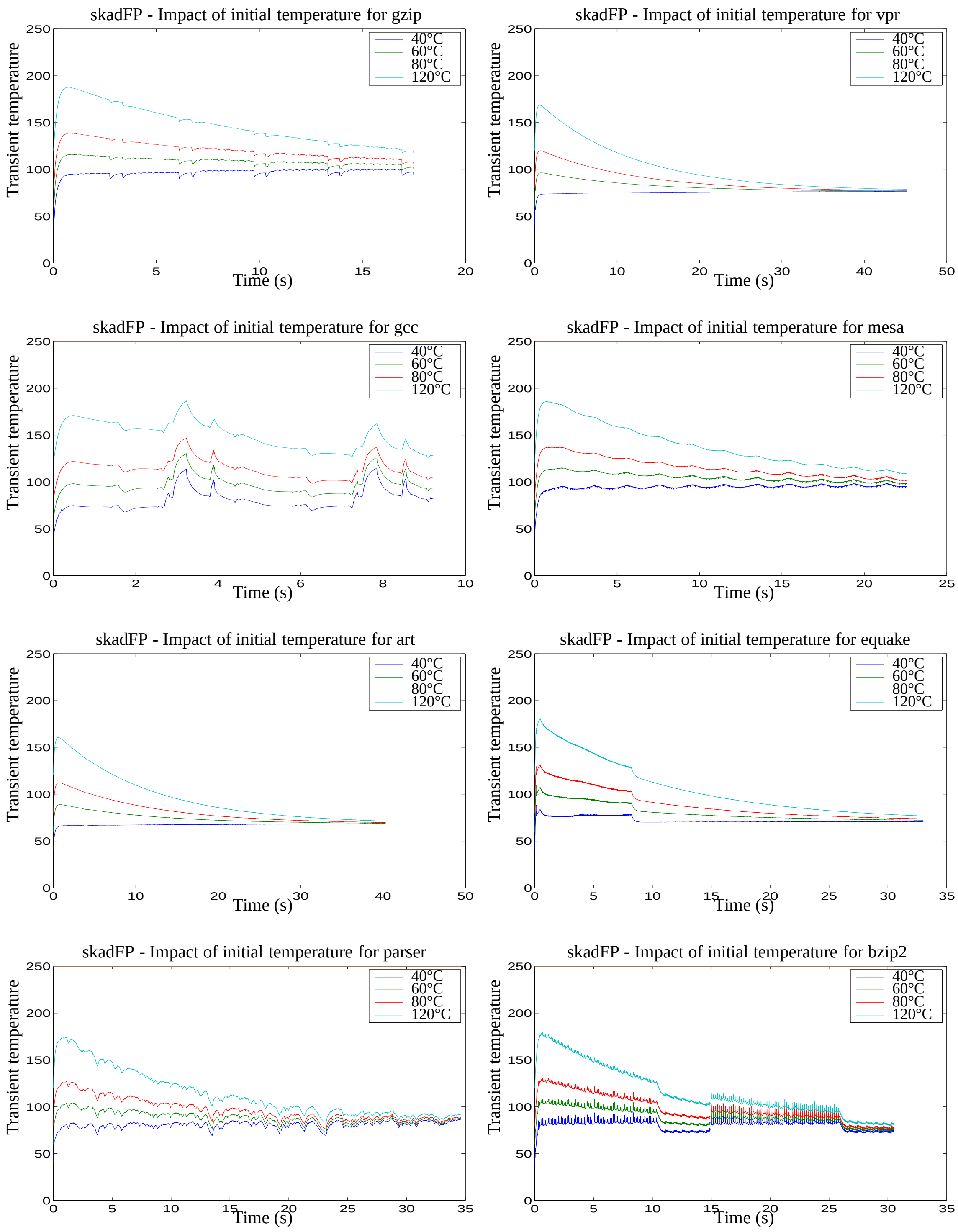


Figure 6.6: Transient profiles for the floorplan obtained using skadFP at four different initial conditions.

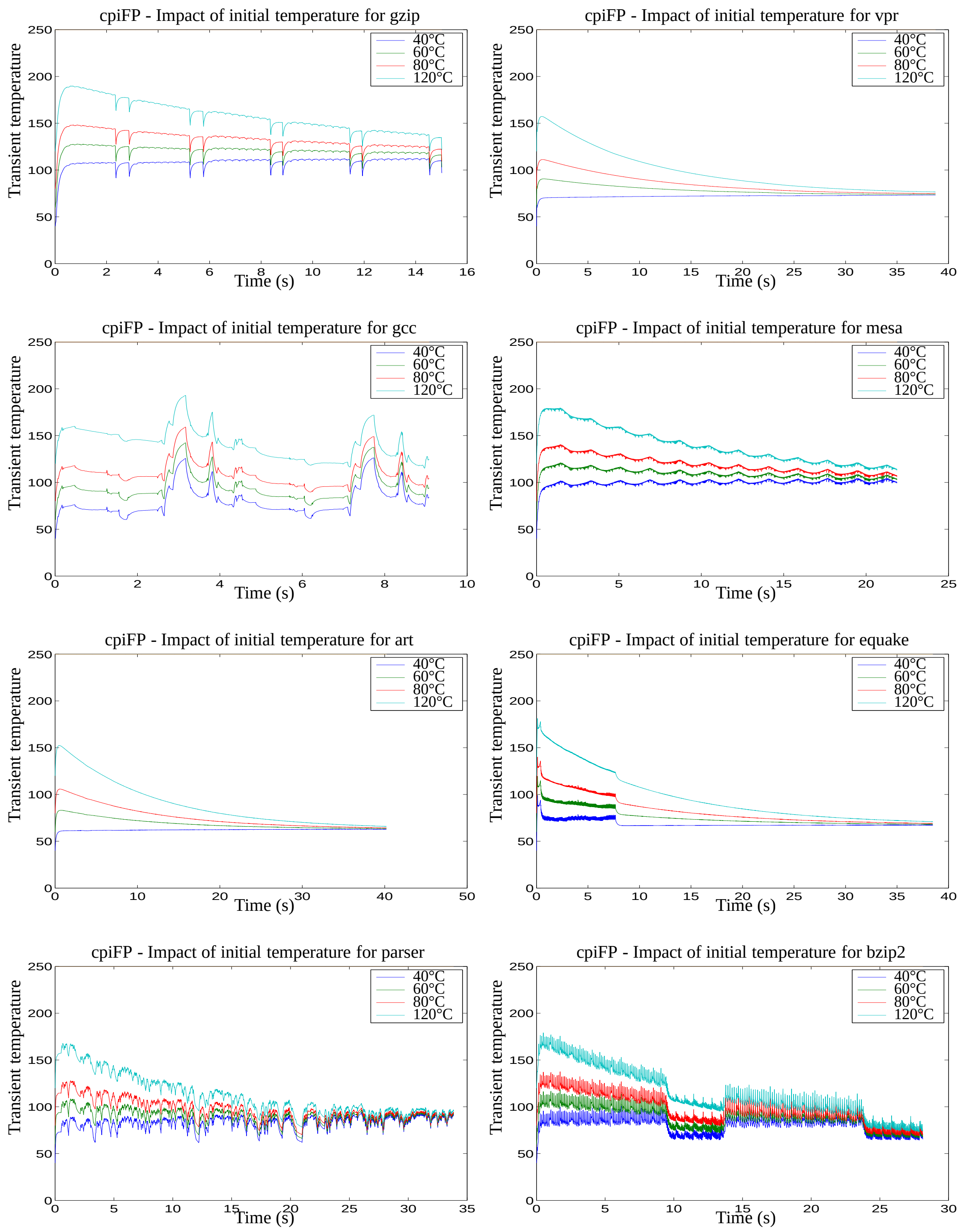


Figure 6.7: Transient profiles for the floorplan obtained using cpiFP at four different initial conditions.

It is apparent from the graphs that the transient curves tend to reach a steady-state like pattern, i.e., the gaps between the curves reduce over time, particularly towards the end of the execution, and this is true for all of the benchmarks. Besides, in most instances, the peak occurs quite early into the execution, and increases with the initial temperature. However, even with the increased leakage seen at high temperatures, the power dissipations are not enough to sustain the high temperatures after these early moments, and the temperatures gradually drop before eventually converging. The steady-state like patterns, however, occur after more than a few seconds, and such long times indicate that the differences in the average temperatures across initial temperatures can be significant. Overall, the graphs show that steady-state temperature is not an ideal metric for measuring thermal performance, since it is quite possible that two instances that have different peaks (and averages) reach the same steady-state.

It can also be observed that some benchmarks such as gcc and bzip2 exhibit uneven activity levels with several low and high temperature phases, in which case, the average and peak of the temperatures may not have a perfect correlation, i.e., a curve with a higher peak than another can have a lower average. On the other hand, benchmarks such as art maintain steady-states or monotonically decreasing temperatures for a major part of the execution time.

Figure 6.8 plots the peaks of the transient curves of Figures 6.5, 6.6 and 6.7 for the four initial temperatures. The graphs show that, for all benchmarks, our proposed floorplanner **therFP** obtains good reductions in the peak temperatures than both **cpiFP** and **skadFP**, particularly for those that exhibit high temperatures such as gcc. For instance, for the benchmark gcc, at 40°C, the floorplan generated by **therFP** reduces the peak by about 14°C as compared to **cpiFP**, while it is about 5°C for **skadFP**, and at the initial temperature of 80°C, the reductions are about 16°C and 6°C, respectively.

Moreover, **therFP** outperforms **skadFP** despite not explicitly attempting to minimize the peak temperature as is done in **skadFP**. This is true for all of the initial temperatures, even though the floorplanning is performed at a single point, i.e., 40°C, indicating

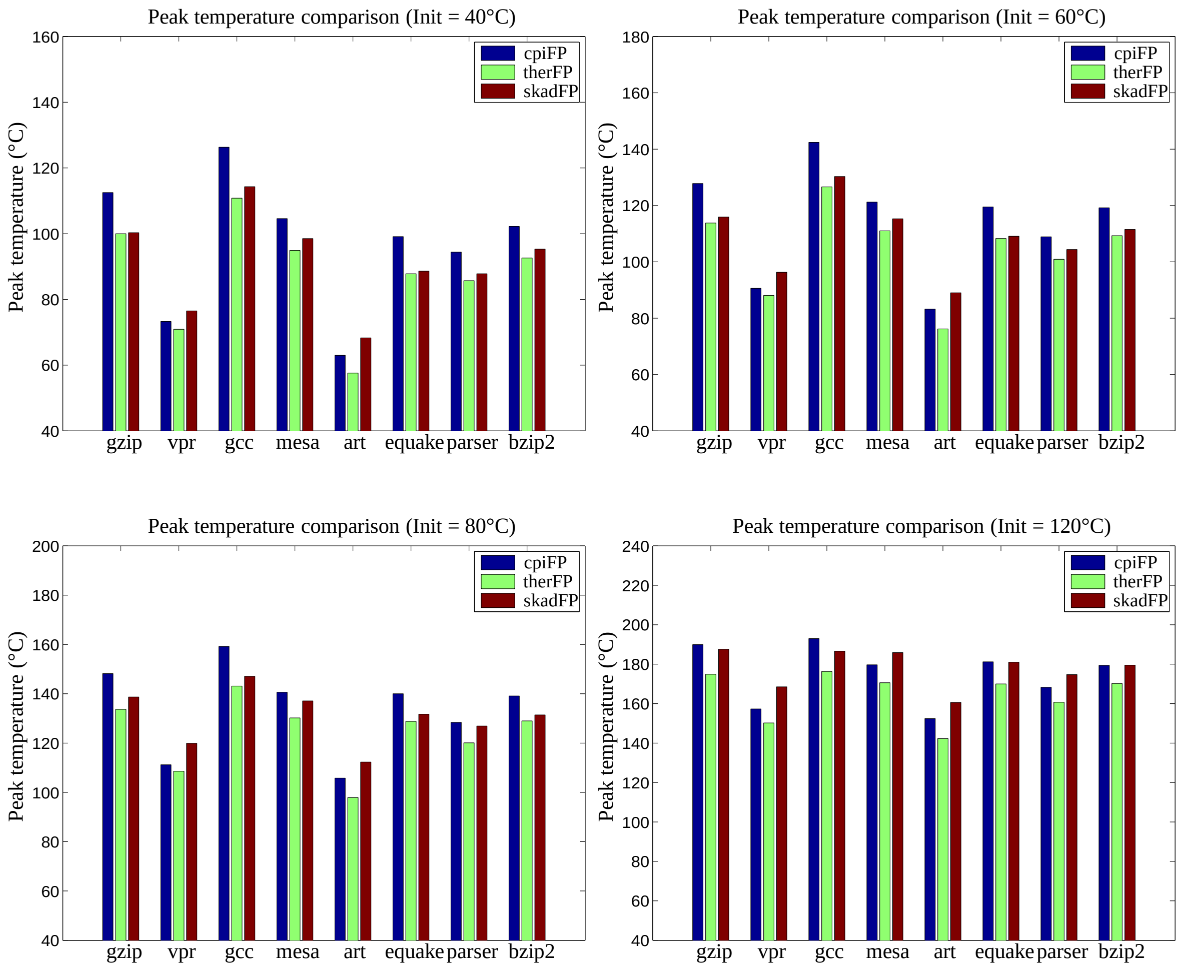


Figure 6.8: Comparison of the peak temperatures for the three floorplanning scenarios at different initial temperatures.

that the initial state has not significantly affected the optimization. A possible reason for this is that the shape of the profile does not change much across different initial settings, as can be seen in Figures 6.5, 6.6 and 6.7; if a technique keeps the temperatures low at some initial temperature, it is likely that the trend is maintained for other initial temperatures as well.

Figure 6.9 compares the average transient temperatures obtained using the three approaches at the four initial temperatures of 40, 60, 80 and 120°C. The plots indicate that **therFP** outperforms both **cpiFP** and **skadFP** by significant amounts for all benchmarks. Reductions of about 9°C and 6°C are obtained over **cpiFP** and **skadFP**, respectively, for gcc, at 40°C. Similar trends are observed for the most part of other initial temperatures.

In addition, since the floorplans are optimized for the average cases and not specifically for each benchmark, the optimization potential for each benchmark may not be fully exploited. Furthermore, benchmarks that have low power profiles such as art and vpr do not offer much scope for optimization, the resultant improvements tend to be small, and in fact, **skadFP** worsens the thermal profiles obtained for art and vpr, where both the average and the peak temperatures are higher than those of **cpiFP**, as shown in Figures 6.8 and 6.9.

Table 6.2 lists the critical block, i.e., the block for which the peak temperature occurs, for a number of scenarios, for each benchmark. It can be seen that the integer adder blocks, $iadd1, iadd2$ and $iadd3$ are critical for a majority of the cases. In general, as was also observed in Section 4.5.3, integer adders see a lot of activity due to the associated instruction mix and tend to be active for a significant percentage of the total execution time. In addition, blocks that implement random logic such as adders typically have high power densities, since the percentage switching transistors during active state is much higher than arrayed structures such as caches.

However, for some benchmarks such as art, which is a floating-point benchmark, it turns out that the register file reg is the critical block. It can also be observed that the criticality shifts to the register file or the register update unit ruu at high temperatures,

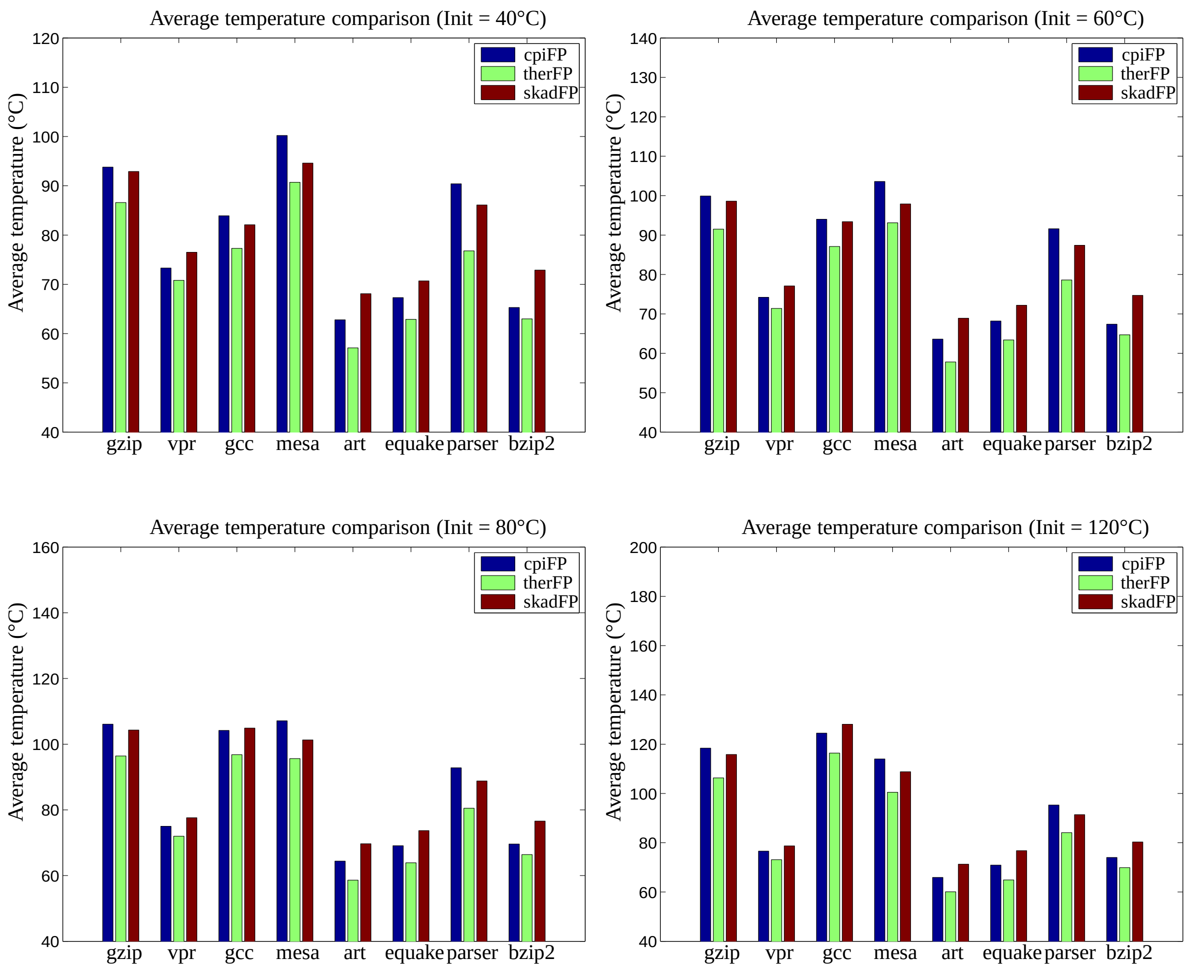


Figure 6.9: Comparison of the average temperature metric for the three floorplanning scenarios at different initial temperatures.

Init. T	**Case**	**gzip**	**vpr**	**gcc**	**mesa**	**art**	**equake**	**parser**	**bzip2**
40°C	cpiFP	$iadd2$	$iadd2$	$iadd2$	$iadd2$	reg	$iadd2$	$iadd2$	$iadd2$
	therFP	$iadd1$	$iadd1$	$iadd1$	$iadd1$	$iadd1$	$iadd1$	$iadd1$	$iadd1$
	skadFP	ruu	reg	$iadd3$	ruu	reg	$iadd3$	ruu	$iadd3$
60°C	cpiFP	$iadd2$	$iadd2$	$iadd2$	$iadd2$	reg	$iadd2$	$iadd2$	$iadd2$
	therFP	$iadd1$	$iadd1$	$iadd1$	$iadd1$	reg	$iadd1$	$iadd1$	$iadd1$
	skadFP	ruu	reg	$iadd3$	ruu	reg	$iadd3$	reg	$iadd3$
80°C	cpiFP	$iadd2$	$iadd2$	$iadd2$	$iadd2$	reg	$iadd2$	$iadd2$	$iadd2$
	therFP	$iadd1$	$iadd1$	$iadd1$	$iadd1$	reg	$iadd1$	$iadd1$	$iadd1$
	skadFP	reg	reg	$iadd3$	reg	reg	reg	reg	$iadd3$
120°C	cpiFP	$iadd2$	reg	$iadd2$	$iadd2$	reg	$iadd2$	$iadd2$	$iadd2$
	therFP	$iadd1$	$iadd1$	$iadd1$	ruu	reg	$iadd1$	$iadd1$	$iadd1$
	skadFP	reg	reg	$iadd3$	reg	reg	reg	reg	reg

Table 6.2: The critical block with the peak temperature for the eight benchmarks at different initial settings (**Init. T** in the table).

such as 120°C. The reason for this is that the units ruu and reg are arrayed structures and dissipate more leakage than execution units such as adders. In such a scenario, due to the exponential dependence of leakage on temperature, the leakage becomes so high in ruu and reg that they become hotter than adders.

Finally, Figure 6.10 depicts the performance degradation, i.e., increase in the CPIs, obtained in **therFP** and **skadFP** due to the inclusion of thermal issues in the cost function, besides performance. On an average, both **therFP** and **skadFP** result in almost identical CPIs, about 6% more than **cpiFP**, where no thermal metrics are considered in the cost.

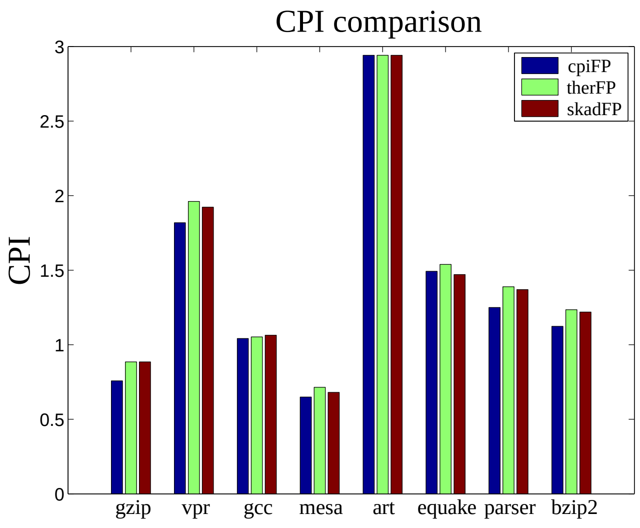


Figure 6.10: Comparison of the CPI metric for the three floorplanning scenarios.

6.7 Conclusion

Thermal issues have become an important concern in microprocessors designed in nanometer technology nodes. This chapter presented a strategy for thermally-aware floorplanning for microprocessors, where the optimization objectives also include the throughput (IPC) issues. The approach also models the IPC-power interaction, and uses a complete transient analysis that captures a thermal profile of a chip in a better way than the steady-state approach, during the floorplanning optimization. The results indicate good improvements both in the average and peak temperatures when compared to an approach derived from a previous work.

Chapter 7

Conclusion

7.1 Summary

This thesis has focused on two important issues of interconnect pipelining and operating temperature that are associated with the high gigahertz frequencies utilized in high-performance integrated circuits, particularly microprocessors, designed in the nanometer technologies. First, *wire-pipelining*, when applied on a circuit to realize multicycle communication, can change the path latencies in a nonuniform way and result in a functionally different version of the circuit. An even more important concern is the potential reduction in the throughput due to the additional latencies introduced into the circuit. Next, the high frequencies, combined with high integration densities, have resulted in high chip temperatures and this poses a significant challenge to the circuit design community, due to the nonlinearly increasing cooling costs.

The problems that we have addressed can be categorized into circuit- and microarchitecture-level issues and our contributions for the problems can be summarized as follows:

- At the circuit-level, we have proposed a solution to correct the functionality of a wire-pipelined circuit. The solution finds the minimal value of the throughput slowdown and also includes a minimum area formulation to minimize the increase in the number of additional flip-flops that are required to be inserted in arriving at a solution. The technique is applied on the ISCAS and the ITC benchmark circuits and the results indicate that wire-pipelining improves performance for most of the circuits.

- At the microarchitecture-level, we have focused on the interactions between architecture design and physical design. Specifically, physical design plays a pivotal role in determining the throughput, measured as the average number of instructions executed per cycle (IPC), and the thermal profile of a microprocessor, through its influence on deciding the latencies of the buses and the heat transfer mechanisms of the processor, respectively. Furthermore, there is also dependence between the throughput and temperature, as the power consumption levels of the chip vary with program execution times.

We have presented floorplanning methodologies to optimize the throughput and the thermal attributes of a microprocessor. The vital ingredient of the methodologies is a design of experiments based approach to limit the number of cycle-accurate simulations required to characterize the throughput and the dynamic power patterns of the architecture. The regression models built from the simulations drive the floorplanner that optimizes for the throughput and the temperature objectives. In addition, our approach uses transient analysis and minimizes the peak and average of the transients as opposed to the steady-state temperature, and also analyze the impact of variations in the initial temperature on the floorplanning optimization. We apply the methodologies on the DLX architecture and a Pentium architecture, in which case only the throughput objective is considered, and the results indicate good improvements in the throughput and reductions in temperatures when compared to existing approaches.

Additionally, we compare a few simulation speed up techniques, namely, sampling and reduced input sets in the context of throughput-aware floorplanning for the DLX architecture. Our results suggest that, although the techniques exhibit variations in the regression models, the differences do not impact the floorplanning optimization.

7.2 Future directions

The regime of wire-pipelining offers exciting opportunities and challenges for further research. Future extensions of the topics addressed in this thesis may include the following:

- **Buffer explosion:** It is important to consider the number of repeaters inserted on the wires in the wire-pipelining regime during optimizations. As shown in [SMCK04], the number of buffers and flip-flops required to optimize the wires of a circuit increases exponentially as the technology advances, .

- **Local interconnect:** As the circuit complexity and clock frequencies increase, at some point, the individual blocks become big enough, and the local wire delays can exceed a clock cycle. In such a scenario, it may be wise to reduce the granularity by splitting each block into sub-blocks.

- **Applications of DOE:** The idea of statistical design of experiments (DOE), which is utilized in this thesis, is particularly useful for domains such as microarchitecture research, where even though there are a small number of factors, each simulation runs for a long time. Several other problems that can use DOE for optimization can be thought of. One such application is in the domain of chip multiprocessing (CMP). Multicore machines tend to have large die sizes and communicate through long multicycle buses. The theory of DOE can be used to build performance models for these multicycle buses, and incorporate them in the floorplanning step. In addition, there is also scope for novel floorplanning schemes for multicore machines, such as hierarchical placement, i.e., inter- and intra- processor placement, and 3D arrangement of devices.

BIBLIOGRAPHY

[AM01] S. N. Adya and I. L. Markov. Fixed-outline floorplanning through better local search. In *Proceedings of the IEEE International Conference on Computer Design*, pages 228–334, October 2001.

[BA97] 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.

[Ban98] P. Bannon. Alpha 21364: A scalable single-chip SMP, 1998. Available at `http://www.digital.com/alphaoem/microprocessorforum.htm`.

[BASB01] F. R. Boyer, E. M. Aboulhamid, Y. Savaria, and M. Boyer. Optimal design of synchronous circuits using software pipelining techniques. *ACM Transactions on Design Automation of Electronic Systems*, 6(4):516–532, October 2001.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1929–1934, May 1989.

[BC97] D. S. Bormann and P. Y. K. Cheung. Asynchronous wrapper for heterogeneous systems. In *Proceedings of the IEEE International Conference on Computer Design*, pages 307–314, October 1997.

[BCB$^+$00] D. M. Brooks, P. W. Cook, P. Boze, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, and M. Gupta. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, August 2000.

[BJS90] M. S. Bazaara, J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley and Sons, New York, New York, 2nd edition, 1990.

[BM01] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the ACM International Symposium on High-Performance Computer Architecture*, pages 171–182, February 2001.

[Bor00] S. Borkar. Obeying Moore's law beyond 0.18 micron. In *Proceedings of the International ASIC/SOC Conference*, pages 26–31, September 2000.

[BTM00] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of ACM International Symposium on Computer Architecture*, pages 83–94, June 2000.

[Cha84] D. M. Chapiro. *Globally-asynchronous locally-synchronous systems*. PhD thesis, Stanford University, Stanford, California, October 1984.

[CJRR03] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis. Microarchitecture evaluation with physical planning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 32–35, June 2003.

[CKG06] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 657–662, July 2006.

[CKM] A. Caldwell, A. B. Kahng, and I. Markov. Capo: A large-scale fixed-die placer. Available at `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/Capo/`.

[CL95] J. Cong and K. S. Leung. Optimal wiresizing under Elmore delay model. *IEEE Transactions on Computer Aided Design of Integrated Circuits*, 14(8):321–336, March 1995.

[CLR89] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachussets, 1st edition, 1989.

[CM04a] M. R. Casu and L. Macchiarulo. A new approach to latency insensitive design. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 576–581, June 2004.

[CM04b] M. R. Casu and L. Macchiarulo. Floorplanning for throughput. In *Proceedings of the ACM International Symposium on Physical Design*, pages 62–67, April 2004.

[CMSV01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer Aided Design of Integrated Circuits*, 20(9):1059–1076, September 2001.

[Coc02] P. Cocchini. Concurrent flip-flop and repeater insertion for high performance integrated circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 268–273, November 2002.

[Con01] J. Cong. An interconnect-centric design flow for nanometer technologies. *Proceedings of the IEEE*, 89(4):505–528, April 2001.

[CRS00] F. Corno, M. S. Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results. *IEEE Design and Test of Computers*, 17(3):44–53, July 2000.

[CWZ05] J. Cong, J. Wie, and Y. Zhang. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 634–639, November 2005.

[CYTD03] C. Chu, E. F. Y. Young, D. K. Y. Tong, and S. Dechu. Retiming with interconnect and gate delay. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 221–226, November 2003.

[Czi99] V. Czitrom. One-factor-at-a-time versus designed experiments. *The American Statistician*, 53(2):126–131, May 1999.

[DIG99] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cycle cost to time ratio problems. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 37–42, June 1999.

[DM70] N. R. Draper and T. J. Mitchell. Construction of a set of 512-run designs of resolution ≥ 5 and a set of even 1024-run designs of resolution ≥ 6. *The Annals of Mathematical Statistics*, 41(3):876–887, June 1970.

[EAB$^+$02] J. Emer, P. Ahuja, E. Borch, A. Klauser, C. Luk, S. Manne, S. S. Mukherjee, H. Patil, and S. Wallace. Asim: A performance model framework. *IEEE Computer*, 35(2):68–76, February 2002.

[EBN00] L. Eekhout, K. De Bosschere, and H. Neefs. Performance analysis through synthetic trace generation. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 1–6, April 2000.

[EHB$^+$04] M. Ekpanyapong, M. B. Healy, C. Ballapuram, S. K. Lim, and H. S. Lee. Thermal-aware 3D microarchitectural floorplanning. Technical Report GIT-cercs-04-37, Georgia Institute of Technology, Atlanta, April 2004.

[EHG$^+$97] D. Edelstein, J. Heidenreich, R. Goldblatt, W. Cote, C. Uzoh, N. Lustig, P. Roper, T. McDevitt, W. Motsiff, A. Simon, J. Dukovic, R. Wachnik, H. Rathore, R. Schulz, L. Su, S. Luce, and J. Slattery. Full copper wiring in a sub-0.25 micro-m CMOS ULSI Technology. In *IEDM Technical Digest*, pages 773–776, December 1997.

[Ekp04] M. Ekpanyapong. Private communication, 2004.

[Elm48] W. C. Elmore. The transient response of damped linear network with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[EMW$^+$04] M. Ekpanyapong, J. R. Minz, T. Watewai, H. S. Lee, and S. K. Lim. Profile-guided microarchitectural floorplanning for deep submicron processor design. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 634–639, June 2004.

[FP00] G. Frederic and B. Pascal. Screening and response surface method applied to the numerical optimization of electromagnetic devices. *IEEE Transactions on Magnetics*, 36(4):1163–1167, January 2000.

[GBCH01] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, 5(1):1–9, May 2001.

[Ger99] S. H. Gerez. *Algorithms for VLSI design automation*. John Wiley and Sons, New York, New York, 1st edition, 1999.

[GS03] B. Goplen and S. S. Sapatnekar. Efficient thermal placement of standard cells in 3D ICs using a force directed approach. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 86–89, November 2003.

[HAT02] S. Hassoun, C. J. Alpert, and M. Thiagarajan. Optimal buffered routing path constructions for single and multiple clock domain systems. In *Pro-*

ceedings of the IEEE International Conference on Computer Aided Design, pages 247–253, November 2002.

[Hen00] J. L. Henning. SPEC CPU 2000: Measuring CPU performance in the new millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[HKM05] Y. Han, I. Koren, and C. A. Moritz. Temperature aware floorplanning. In *Notes of Workshop on temperature-Aware Computer Systems*, June 2005.

[HP96] J. Hennessy and D. Patterson. *Computer architecture: A quantitative approach*. Morgan Kaufmann Publishers, San Francisco, California, 2nd edition, 1996.

[HP97] J. Hennessy and D. Patterson. *Computer organization and design: A hardware/software interface*. Morgan Kaufmann Publishers, San Francisco, California, 2nd edition, 1997.

[HXV$^+$05] W-L. Hung, Y. Xie, N. Vijaykrishnan, C. Addo-Quaye, T. Theocharides, and M. J. Irwin. Thermal-aware floorplanning using genetic algorithms. In *Proceedings of IEEE International Symposium on Quality Electronics Design*, pages 634–639, March 2005.

[IM03] C. Iski and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings of the ACM International Symposium on High-Performance Computer Architecture*, pages 281–291, February 2003.

[Int98] Intel Corporation. Pentium processor manual, 1998. Available at `http://www.x86.org/intel.doc/686manuals.htm`.

[JCK06] J. Julvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proceedings of the IEEE Interna-*

tional Conference on Computer Aided Design, pages 448–455, November 2006.

[Jou90] N. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of ACM International Symposium on Computer Architecture*, pages 364–373, May 1990.

[JYK$^+$05] A. Jagannathan, H. H. Yang, K. Konigsfed, D. Milliron, M. Mohan, M. Romesis, G. Reinman, and J. Cong. Microarchitecture evaluation with floorplanning and interconnect pipelining. In *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference*, pages 32–35, Jan. 2005.

[KL02] A. J. KleinOsowski and D. J. Lilja. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *IEEE Computer Architecture Letters*, 1, June 2002.

[Law66] E. L. Lawler. Optimal cycles in doubly weighted directed linear graphs. In *Proceedings of the International Symposium on Theory of Graphs*, pages 209–213, July 1966.

[Lil00] D. J. Lilja. *Measuring computer performance*. Cambridge University Press, New York, New York, 1st edition, 2000.

[Loe] A. Loebel. MCF Version 1.2 - A network simplex implementation . Available at `http://www.zib.be`.

[LPMS97] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the International Symposium of Microarchitecture*, pages 330–335, December 1997.

[LRS83] C. E. Leiserson, F. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proceedings of the Third Caltech Conference on VLSI*, pages 87–116, March 1983.

[LS83] C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, 1(1):41–67, Spring 1983.

[LSLH04] C. Long, L. J. Simonson, W. Liao, and L. He. Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 640–645, June 2004.

[LZ03] C. Lin and H. Zhou. Retiming for wire pipelining in system-on-chip. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 215–220, November 2003.

[LZKC02] R. Lu, G. Zhong, C. K. Koh, and K. Y. Chao. Flip-flop insertion for early interconnect planning. In *Proceedings of the IEEE Design, Automation and Test in Europe Conference*, pages 690–695, March 2002.

[MLH$^+$00] R. McInerney, K. Leeper, T. Hill, H. Chan, B. Basaran, and L. McQuiddy. Methodology for repeater insertion management in the RTL, layout, floorplan and fullchip timing databases of the Itanium^{TM} microprocessor. In *Proceedings of the ACM International Symposium on Physical Design*, pages 99–104, April 2000.

[Mon00] D. C. Montgomery. *Design and analysis of experiments*. John Wiley and Sons, New York, New York, 5th edition, 2000.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. In *Electronics Magazine*, volume 38, pages 114–117, April 1965.

[NCLS05] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar. Microarchitecture-aware floorplanning using a statistical design of experiments approach. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 579–584, June 2005.

[NCLS06] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar. Comparing simulation techniques for microarchitecture-aware floorplanning. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 80–88, March 2006.

[Noo04] V. Nookala. A method for correcting the functionality of a wire-pipelined circuit. Master's thesis, University of Minnesota, Minneapolis, Minnesota, January 2004.

[NS01] S. Nussbaum and J. E. Smith. Modeling superscalar simulators via statistical simulation. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 15–24, September 2001.

[OCF00] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor designs. In *Proceedings of ACM International Symposium on Computer Architecture*, pages 71–82, May 2000.

[PB56] R. Plackett and J. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33(4):305–325, June 1956.

[SABR04] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 70–80, June 2004.

[Sch02] L. Scheffer. Methodologies and tools for pipelined on-chip interconnect. In *Proceedings of the IEEE International Conference on Computer Design*, pages 152–157, October 2002.

[Sei94] J. N. Seizovic. Pipeline synchronization. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits*, pages 87–96, November 1994.

[Sem01] Semiconductor Industry Association. International Technology Roadmap for Semiconductors, 2001. Available at `http://public.itrs.net`.

[She95] N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, Massachussets, 2nd edition, 1995.

[SKLP$^+$01] M. Steinhaus, R. Kolla, J. Larriba-Pey, T. Ungerer, and M. Valero. Transistor count and chip-space estimation of simplescalarbased microprocessor models. In *Notes of the Workshop on Complexity-Effective Design*, June 2001.

[SLC01] J. Sall, A. Lehman, and L. Creighton. *JMP start statistics*. Duxbury Press, Pacific Grove, California, 2nd edition, 2001.

[SMCK04] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick. Repeater scaling and its impact on CAD. *IEEE Transactions on Computer Aided Design of Integrated Circuits*, 23(4):451–463, April 2004.

[SPHC02] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–97, October 2002.

[SSH$^+$03] K. Skadron, M. Stan, W. Huang, S. Velusamy, and K. Sankaranarayananand D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of ACM International Symposium on Computer Architecture*, pages 2–13, June 2003.

[SSH$^+$05] K. Skadron, M. Stan, W. Huang, S. Velusamy, and K. Sankaranarayananand D. Tarjan. A case for thermal-aware floorplanning at the microarchitectural level. *The Journal of Instruction-Level Parallelism*, 8:8–16, October 2005.

[SSL$^+$92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/URL M92/41, The University of California, Berkeley, March 1992.

[SZH04] V. Seth, M. Zhao, and J. Hu. Exploiting level sensitive latches in wire pipelining. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 283–290, November 2004.

[Tra97] Transaction Processing Performance Council. TPC Benchmark C standard specification Revision 3.3.2, 1997. Available at `http://www.tpc.org`.

[TTBN00] A. Tabbara, B. Tabbara, R. K. Brayton, and A. R. Newton. Integration of retiming with architectural floorplanning. *Integration, the VLSI Journal*, 29(1):25–43, March 2000.

[vG90] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 2, pages 865–868, May 1990.

[WC02] T. Wang and C. C. Chen. 3-D thermal-ADI: A linear-time chip level transient thermal simulator. *IEEE Transactions on Computer Aided Design of Integrated Circuits*, 21(12):1434–1445, December 2002.

[WH00] C. F. J. Wu and M. Hamada. *Experiments: Planning, analysis, and parameter design optimization*. John Wiley and Sons, New York, New York, 1st edition, 2000.

[WWFH03] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of ACM International Symposium on Computer Architecture*, pages 84–97, June 2003.

[WYYC05] Y. W. Wu, C. Yang, P. Yuh, and Y. Chang. Joint exploration of architectural and physical design spaces with thermal consideration. In *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, pages 123–126, August 2005.

[YKS$^+$05] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins. Characterizing and comparing prevailing simulation techniques. In *Proceedings of the ACM International Symposium on High-Performance Computer Architecture*, pages 266–277, February 2005.

[YL02] J. J. Yi and D. J. Lilja. Improving processor performance by simplifying and bypassing trivial computations. In *Proceedings of the IEEE International Conference on Computer Design*, pages 462–467, October 2002.

[YLH03] J. J. Yi, D. J. Lilja, and D.M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *Proceedings of the ACM International Symposium on High-Performance Computer Architecture*, pages 281–291, February 2003.

[YVKI00] W. Ye, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 340–345, June 2000.

[ZGS06] Y. Zhan, B. Goplen, and S. S. Sapatnekar. Electrothermal analysis and optimization techniques for nanoscale integrated circuits. In *Proceedings of the ACM Asia-South Pacific Design Automation Conference*, pages 219–222, January 2006.

[ZLA00] H. Zhou, I. M. Liu, and A. Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(7):819–824, July 2000.