INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

DESIGN AND MODELING OF POWER-EFFICIENT COMPUTER ARCHITECTURES

DAVID BROOKS

A DISSERTATION

PRESENTED TO THE FACULTY

OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE

BY THE DEPARTMENT OF

ELECTRICAL ENGINEERING

November 2001

Abstract

Power dissipation and thermal issues are increasingly significant in modern processors. As a result, it is crucial that power/performance tradeoffs be made more visible to chip architects and compiler writers, in addition to circuit designers. Most traditional power analysis tools achieve high accuracy by calculating power estimates for designs only after the circuit design, layout, and floorplanning are complete. In addition to being available only late in the design process, such tools are often quite slow, which compounds the difficulty of running them for a large space of design possibilities.

This thesis presents a methodology for estimating power dissipation at a much earlier stage in the design cycle and at a much higher level. Wattch and PowerTimer are two working examples of the use of this methodology. Both tools provide a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level. These tools are 1000X or more faster than existing layout-level power tools, and yet maintain accuracy within 10% of their estimates as verified using industry tools on leading-edge designs. These tools can allow architects to explore and cull the design space early on and opens up the field of power-efficient computing to a wider range of researchers by providing a power evaluation methodology within the portable and familiar SimpleScalar framework.

This thesis also considers several applications of architectural-level power modeling to propose specific architectural-level power and temperature saving optimizations – value-based clock gating and dynamic thermal management. Value-based clock gating is a technique which exploits the dynamic bitwidth requirements of typical

applications to save power within arithmetic units and the memory hierarchy. We have demonstrated that this technique can save roughly 50% of the power in the integer execution units. With dynamic thermal management, temperature sensors and throttling techniques are combined to adaptively slow down the CPU for extended periods of particularly high-power code sequences. This allows the CPU package and power delivery system to be designed for a much lower maximum power rating, with minimal performance impact for typical applications.

The techniques presented in this thesis represent some of the first work in the area of high-performance, low-power processor design at the architectural level. We hope that this work, and the other research in the area of low-power architectural modeling and design, will help future generations of processor architectures to meet the many new challenges in this area.

Acknowledgements

I would like to gratefully acknowledge the guidance of my advisor, Professor Margaret Martonosi for all of the help during my four years at Princeton. I would also like to acknowledge the other Computer Engineering and Computer Science faculty at Princeton. Finally, I am very grateful to the other members of my reading committee: Professor Doug Clark and Dr. Pradip Bose.

I would like to thank Dr. Vivek Tiwari at Intel Corporation and Dr. Pradip Bose at IBM for being extremely supportive of my dissertation work at Princeton and during internships at those companies.

I would like also like to thank my family, especially my parents, Morris and Christine Brooks, and my sister, Persephone Brooks-Bilson, for their support and encouragement over the years. Thanks also go to Pai-hui Iris Hsu for her friendship during our years in graduate school.

This work has been supported by research funding from the National Science Foundation, a donation from Intel Corp, and an IBM University Partnership award. I would also like to thank the National Science Foundation for receiving an NSF Graduate Fellowship, Princeton University for the Gordon Wu Fellowship.

Contents

Chapter 1

Introduction and Contributions

Power-aware computing has traditionally been the primary focus of designers of portable and battery-powered computing systems and has in the past largely been considered a low-level circuit design issue. In the past several years, we have seen two major shifts in the focus of power-aware computing that have greatly increased the amount of research interest in this field.

First, the need for power-efficient designs is no longer solely associated with portable computing systems. Power dissipation has rapidly become a first-order design constraint in virtually every type of computing system including hand-held devices, set-top entertainment systems, desktop computers, and the most performance-hungry compute servers. As clock rates and die sizes increase, power dissipation is predicted to soon become the key limiting factor on the performance of single-chip microprocessors [41; 89]. Already, current high-end microprocessors are beginning to reach the limits of conventional air cooling techniques. In addition to battery life and cooling concerns the difficulties of delivering large and highly-varying amounts of current onto the chip are also significant.

The second major shift is that researchers in power-aware design have begun to focus on power and energy savings at higher levels in the design hierarchy including the logic design [90], microarchitecture [40; 52; 59; 87], software controlled voltage scaling

[72; 92], and the compiler [32]. Specialized circuit techniques have been the main strategies for low-power design in the past, and these will continue to be important areas in the future. Unfortunately, these techniques alone are not sufficient; higher-level strategies for reducing power consumption are increasingly crucial. Architectural and software techniques—in addition to lower-level circuit techniques—must play a major role in creating power-efficient computer systems.

1.1 Contributions

These two major shifts have greatly increased the amount of research interest and the potential for reducing power and energy consumption in computer systems. However, when we consider the importance of power-aware computing in more complex systems, as well as power savings techniques at the architectural and software levels, power modeling becomes a significant challenge. Because of this, my thesis research has had two major contributions.

- First, we have addressed the problem of *architectural-level power modeling* by developing methodologies for estimating power on top of architectural performance simulators. This work has led to the development of two tools: Wattch [21], a publicly available tool based on SimpleScalar [24], and PowerTimer [22], a tool used within IBM Research.
- Second, my research has utilized these modeling tools to develop *techniques for reducing power* in high-performance computing systems. These techniques have focused on reducing both average power and maximum power.

1.1.1 Architectural-level Power Modeling

Estimating the power dissipation of a computing system generally requires transistor-level circuit schematics and a detailed circuit simulation environment such as SPICE [55]. This poses two major problems for modeling power at the architectural level. First, circuit-level simulation is extremely slow, requiring orders of magnitude more time per instruction than architectural-level performance simulation. Second, architectural level studies are generally performed in the planning stages of the design before the circuit and RTL designs have begun to take shape.

To overcome these problems, intelligent abstractions must be developed. In this research, we developed analytical power models for common hardware structures that are present in typical microprocessors. These structures include register files, caches, content-addressable memories, and interconnect. These power models are parameterizable, allowing structures with various sizes and attributes to be instantiated. Finally, the power models are tightly integrated into a traditional architectural-level performance simulator. Cycle-level activity and utilization statistics from the performance simulator are combined with the power models of the hardware structures to provide power estimates. This framework provides accurate power estimates on a per-cycle basis with approximately a 30% simulation time overhead over performance simulation alone. I have also developed a similar framework using industrial simulators and power models extracted from a commercial microprocessor at IBM. Finally, my research has focused on the validation of both of these frameworks and will continue to do so as improvements are made to the modeling methodology. Establishing a solid power modeling methodology is critical for allowing architects to rapidly explore large design spaces and to consider methods to reduce power dissipation in the planning stage of the design.

1.1.2 Techniques for Power-Aware Design

A primary goal of developing accurate and efficient power modeling methodologies is to assist in the development of techniques for power-efficient design. In this thesis, I will describe several techniques including value-based clock gating and dynamic thermal management. Value-based clock gating seeks to disable the upper portion of functional units based on dynamic information gathered about the values being executed. This technique capitalizes on the disparity between the bitwidth requirements of address and data calculations. This disparity increases when considering processors with wide datapaths, and we demonstrated that in 64-bit processors this technique can reduce the power dissipation of the functional units by over 50%, which can lead to full chip power savings of roughly 5-10%. I have investigated practical implementations of this technique as well as extensions into the memory hierarchy.

We have also investigated the benefits of dynamic thermal management. This is a method to reduce the cost of thermal packaging of microprocessors by reducing the effective maximum power dissipation of the processors. This technique is based on the observation that the maximum chip power dissipation is achieved only under extreme circumstances that do not typically occur in most applications. With the use of on-chip thermal sensors, the operating system or microarchitecture can use various techniques to dynamically trade small amounts of performance for reduced power dissipation when these unusual circumstances occur. We demonstrate that for many applications the thermal packaging requirements can be reduced substantially while maintaining performance.

1.2 Organization

Chapter 2 will discuss a methodology for architectural-level power modeling and the energy models in the context of the *Wattch* tool. Chapter 3 describes the *PowerTimer*

tool that uses this methodology in the context of industrial simulators and energy models. Chapter 4 discusses the model validation of these two tools. This chapter also seeks to quantify the robustness of the relative accuracy of these models. Chapter 5 discusses value-based clock gating, a technique targeting average power dissipation. Chapter 6 discusses dynamic thermal management, a technique targeting maximum power dissipation in high-performance microprocessors. Chapter 7 will summarize the major contributions of this thesis, discuss areas of future research, and offers conclusions.

Chapter 2

Power Modeling – Wattch

Research in the area of high-performance, power-efficient computer architectures is still in the preliminary stages. A major obstacle for such research has been the lack of infrastructure that analyzes and quantifies the power ramifications of different architectural choices. Creating such infrastructure requires balancing the need for low-level detail and accuracy against the need for higher-level abstractions offering simulator speed and portability.

This chapter will present Wattch, a framework for analyzing and optimizing microprocessor power dissipation at the architectural-level. Wattch's power estimates are based on a suite of parameterizable power models for different hardware structures and on per-cycle resource usage counts generated through cycle-level simulation. In Chapter 3, I will also discuss PowerTimer, a tool that uses a a similar core methodology, although its energy models are derived from circuits used in an existing commercial, high-performance microprocessor design.

Wattch is 1000X or more faster than existing layout-level power tools, and yet maintains accuracy within 10% of their estimates as verified using industry tools on leading-edge designs. We have performed a validation study on Wattch and PowerTimer. This analysis is presented in Chapter 4 where we present several validations of Wattch's accuracy and discuss the robustness of these simulators to error.

Wattch is intended to be a complement to existing lower-level tools; it allows architects to explore and cull the design space early on, using faster, higher-level tools. It also opens up the field of power-efficient computing to a wider range of researchers by providing a power evaluation methodology within the portable and familiar SimpleScalar framework.

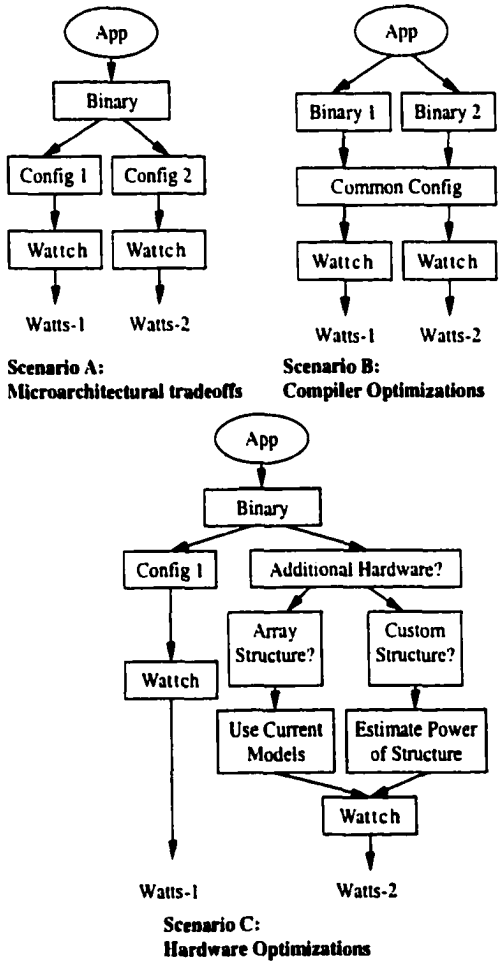

Figure 2.1: Three scenarios for using architectural-level power analysis.

Figure 2.1 shows three possible usage flows for Wattch. Scenario A applies to cases where the user is interested in comparing several design configurations that are achievable simply by varying parameters for hardware structures that we have

modeled. Scenario B is for software or compiler development, where a single hardware configuration is used and several programs are simulated and compared. The third usage scenario highlights Wattch's modularity. Additional hardware modules can be added to the simulator. In some cases, these hardware models follow the template of a hardware structure we already handle. For these cases (i.e., array structures) the user can simply add a new instantiation of the model into the simulator. For other types of new hardware, the model will not fit any already developed, but it is relatively easy to plug new models into the Wattch framework. In Section 2.3, we demonstrate case studies in which the power simulator can be used to perform these three types of power analysis.

2.1 Related Work

Related research falls into two categories. First, we touch on some relevant work on architectural-level techniques for reducing power consumption, and second, we discuss related strategies for estimating power consumption at the architectural level.

Prior work in architectural-level techniques for power reduction has mainly focused on caches [7; 50; 52; 87]. This focus can be attributed to two factors. First, embedded microprocessors, historically the main focus of low-power design, frequently devote a large portion of their power budget to caches, in some cases up to 40% [62]. Second, since caches are regular structures, they are somewhat easier to model than other units, and thus, it can be easier to quantify power savings in caches.

Some work on architectural-level power reduction has addressed other areas of the processor. For example, Manne et al. showed how branch prediction confidence estimators can be used to control branch speculation in order to reduce power consumption [59]. This work presents results in terms of the amount of needless speculative work saved per pipeline stage, as an indicator of power savings. Other prior work has

discussed the power benefits of value-based clock gating in integer ALUs [17] which will be discussed in Chapter 4. In both of these prior papers, a simple measure of the proposed strategy's power effectiveness can be offered by quantifying some type of work that is "saved". In one case, for example, this is the number of fruitless speculative cycles that were saved; in the other case, it is the number of result bits that need not be computed. While such work-saved measures are accurate and very useful for individual techniques, apples-to-apples comparisons of different power-saving techniques require a single common power metric. This is our motivation for creating an architectural-level power simulator.

There has also been related work in architectural-level power estimation tools developed about the same time as Wattch and PowerTimer. Published at ISCA in 2000 simultaneously with Wattch, SimplePower [94] is a tool in which capacitance data was generated from switch-level simulation of the functional unit designs; thus, the models are not easily parameterizable. This simulator is primarily focused on single-issue embedded microprocessors, and does not model out-of-order hardware, so it is difficult for us to compare the speed or accuracy of our approach with this related work.

Cai et al. in 1999 and 2000 have proposed two models for architectural power-estimation based on SimpleScalar. First, the Cai-Lim model [27], proposes power-density based estimates which are combined with activity factors observed within SimpleScalar. The second model, Tempest [33], introduces mixed-mode simulation which can either use power-density based estimates or analytical estimates.

Zyuban et al. in 2000 have also proposed a SimpleScalar based model which uses analytical estimates to and explore multi-clustered architectures for power-performance efficiency [99; 101].

Numerous low-power research studies, as well as next-generation power/performance modeling efforts including the PowerAnalyzer project at the University of Michigan

and the University of Colorado and the Liberty/MESCAL project at Princeton have used portions of the above simulators in their code base.

Lower-level power tools such as *PowerMill* [88] and *QuickPower* [61] operate on the circuit and Verilog level. While providing excellent accuracy, these types of tools are not especially useful for making architectural decisions. First, architects typically make decisions in the planning phase before the design has begun, but both of these tools require complete HDL or circuit designs. Second, the simulation runtime cost for these tools is unacceptably high for architecture studies, in which the tradeoffs between many hardware configurations must be considered. The point of our work is not to compete with these lower-level tools, but rather to expose the basics of power modeling at a higher-level to architects and compiler writers. In a manner analogous to the development of tools for cycle-level architectural performance simulation, tools for architectural-level power simulation will help open the power problem to a wider audience.

Section 2.2 provides a detailed description of our power modeling methodology. An in-depth description of our validation strategy is described in Chapter 4. Section 2.3 provides three case studies detailing how Wattch can be used to perform microarchitectural tradeoff studies for low-power designs, compiler tradeoffs for power, and hardware optimizations for low-power. In Section 2.4 we discuss future possibilities for research in the area of power-efficient architectures and provide conclusions.

2.2 Wattch Power Modeling Methodology

The foundations for our power modeling infrastructure are parameterized power models of common structures present in modern superscalar microprocessors. These power models can be integrated into a range of architectural simulators to provide power estimates. In this work we have integrated these power models into the SimpleScalar

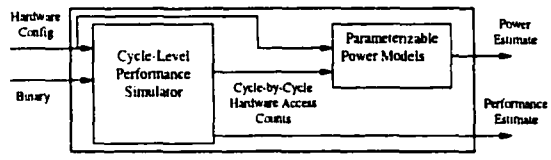

Figure 2.2: Overall Structure of Wattch.

architectural simulator [24].

Figure 2.2 illustrates the overall structure of Wattch and the interface between the performance simulator and the power models. In the following section we describe the power models in detail. We have performed both low-level and high-level validations of these models; we present these validation results in Section 4.2.

2.2.1 Detailed Power Modeling Methodology

The main processor units that we model fall into four categories:

- Array Structures: Data and instruction caches, cache tag arrays, all register files, register alias table, branch predictors, and large portions of the instruction window and load/store queue.
- Fully Associative Content-Addressable Memories: Instruction window/reorder buffer wakeup logic, load/store order checks, and TLBs, for example.
- Combinational Logic and Wires: Functional Units, instruction window selection logic, dependency check logic, and result buses.
- Clocking: Clock buffers, clock wires, and capacitive loads.

In CMOS microprocessors, dynamic power consumption (P_d) is the main source of power consumption, and is defined as: $P_d \approx CV_{dd}^2 af$. Here, C is the load capacitance, V_{dd} is the supply voltage, and f is the clock frequency. The activity factor, a, is a fraction between 0 and 1 indicating how often clock ticks lead to switching activity on average. Our model estimates C based on the circuit and the transistor sizings as

described below. V_{dd} and f depend on the assumed process technology. In this work, we use the .35um process technology parameters from [69].

The activity factor is related to the benchmark programs being executed. For circuits that pre-charge and discharge on every cycle (i.e., double-ended array bitlines) an a of 1 is used. The activity factors for certain critical subcircuits (i.e., single-ended array bitlines) are measured from the benchmark programs using the architectural simulator. The vast majority of the nodes that have a large contribution to the power dissipation fall under one of these two categories. For subcircuits in which we are unable to measure activity factors with the simulator (such as the internal nodes of the decoder) we assume a base activity factor of .5 (random switching activity). Finally, our higher-level power modeling selectively clock-gates unneeded units on each clock cycle, effectively lowering the activity factor.

The power consumption of the units modeled depends very much on the particular implementation, particularly on the internal capacitances for the circuits that make up the processor. We model these capacitances using assumptions that are similar to those made by Wilton and Jouppi [96] and Palacharla, Jouppi, and Smith [69] in which the authors performed delay analysis on many of the units listed above. In both of the above works, the authors reduced each of the above units into stages and formed RC circuits for each stage. This allowed them to estimate the delay for each stage, and by summing these, the delay for the entire unit.

For our power analysis we perform similar steps, but with two key differences. First, we are only interested in the capacitance of each stage, rather than both R and C. Second, in our power analysis the power consumption of *all* paths must be analyzed and summed together. In contrast, when performing delay analysis, only the expected critical path is of interest. Table 2.1 summarizes our capacitance formulas, and the descriptions below elaborate on our approach.

Array Structures Our array structure power model is parameterized based on the number of rows (entries), columns (width of each entry), and the number of read/write ports. These parameters affect the size and number of decoders, the number of wordlines, and the number of bitlines. In addition, we use these parameters to estimate the length of the pre-decode wires as well as the lengths of the array's wordlines and bitlines.

For the array structures we model the power consumption on the following stages: decoder, wordline drive, bitline discharge, and output drive (or sense amplifier). Here we only discuss in detail the wordline drive and bitline discharge. These two components form the bulk of the power consumption in the array structures. Figure 2.3 shows a schematic of the wordlines and bitlines in the array structure.

Node	Capacitance Equation
Regfile Wordline Capacitance =	$C_{diff}(WordLineDriver)$ + $C_{gate}(CellAccess) * NumBitlines$ + $C_{metal} * WordLineLength$
Regfile Bitline Capacitance =	$C_{diff}(PreCharge)$ + $C_{diff}(CellAccess) * NumWdlines$ + $C_{metal} * BLLength$
CAM Tagline Capacitance =	$C_{gate}(CompareEn) * NumberTags$ + $C_{diff}(CompareDriver)$ + $C_{metal} * TLLength$
CAM Matchline Capacitance =	$2 * C_{diff}(CompareEn) * TagSize$ + $C_{diff}(MatchPreCharge)$ + $C_{diff}(MatchOR)$ + $C_{metal} * MLLength$
ResultBus Capacitance =	$.5 * C_{metal} * Num.ALU * ALUHeight)$ + $C_{metal} * (RegfileHeight)$

Table 2.1: Capacitance equations of critical nodes.

Modeling the power consumption of the wordlines and bitlines requires estimating the total capacitance on both of these lines. The capacitance of the wordlines include

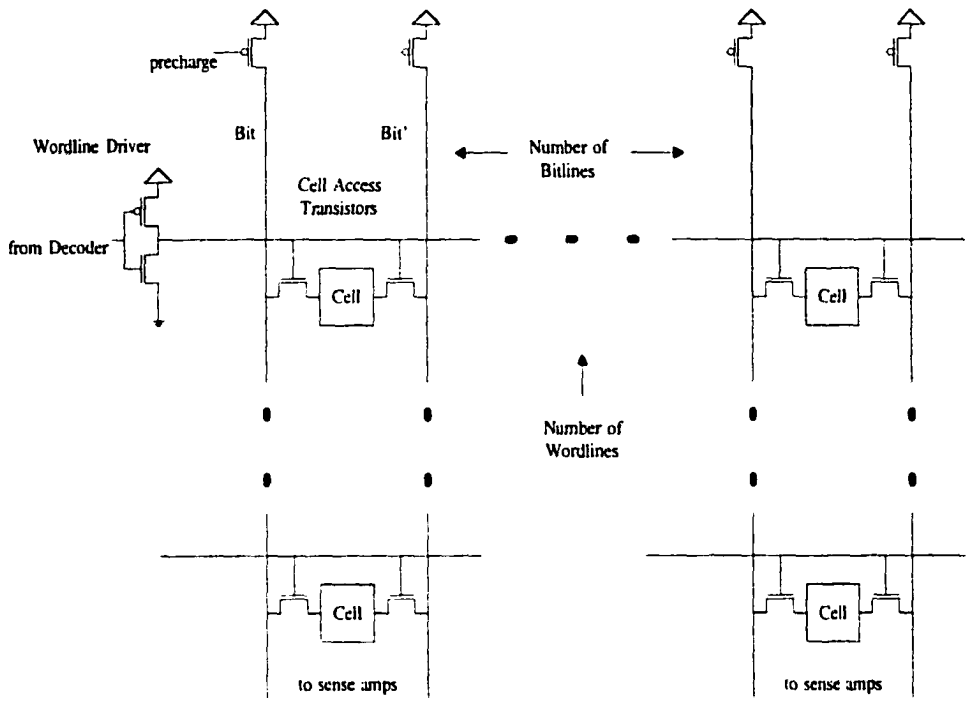

Figure 2.3: Schematic of wordlines and bitlines in array structure.

three main components. These three components are the diffusion capacitance of the wordline driver, the gate capacitance of the cell access transistor times the number of bitlines, and the capacitance of the wordline's metal wire.

The bitline capacitance is computed similarly. The total capacitance is equal to the diffusion capacitance of the pre-charge transistor, the diffusion capacitance of the cell access transistor multiplied by the number of word lines, and the metal capacitance of the bitline. The models that we have created provide the option to use single-ended or double-ended bitlines. In this work we assume that register file array structures use single-ended bitlines and that cache array structures use double-ended bitlines. Equations for the wordline and bitline capacitance are shown in Table 2.1.

Multiple ports on the array structure will increase the power consumption in three ways. First, there will be more capacitance on the wordlines because each

additional port requires an additional transistor connection. Second, each additional port requires up to two additional bitlines (bit and bit'), each of which must precharge/evaluate on every cycle. Finally, each core cell becomes larger which leads to longer word and bitlines, incurring additional wire capacitance.

Transistor sizing plays an important role in the amount of capacitance within the various structures. We use the transistor sizings of [69; 96] wherever sizes are noted. Generally, transistors in array structures are kept relatively small to reduce the area. In our model, certain critical transistors are automatically sized based on the model parameters to achieve reasonable delays. For example, the wordline driver transistor is critical for driving the wordline high in a short amount of time. The width of this transistor is scaled based on the amount of capacitance on the wordlines. Because of the length of the word and bitlines, the internal wiring capacitance of these structures is significant.

Our analysis is similar to Wilton and Jouppi's study of cache array structures [96]. That work analyzes the access and cycle times for on-chip caches. We modify the analysis to take into account multi-ported array structures such as the register alias table, register file, etc. In addition, Kamble and Ghose developed power models for cache arrays to study power optimizations within caches [50] and Zyuban and Kogge studied low-power circuit techniques for register file structures [100].

The physical implementation of some array structures may be very different from the logical structure. For example, caches may be banked in order to provide reasonable delays. In this work we estimate the physical implementations for cache structures using the help of the *Cacti* tool [96]. *Cacti* is a tool developed to determine delay-optimal cache hardware configurations given cache parameters such as size, block size, and associativity. We perform similar analysis on branch prediction structures to make them as square as possible in the physical implementation.

CAM Structures Our analysis of the CAM structures is very similar to that for array structures. However, in the CAM structure we model taglines and matchlines instead of bitlines and wordlines. Equations for the CAM tagline and matchline capacitance are shown in Table 2.1. Again we use a parameterized model which can be extended to the various CAM structures in the processor. We take into account the number of rows (number of tags), columns (number of bits per tag to match), and ports on the CAM. The analysis is similar to that for the array structures and follows the methodology taken in [69].

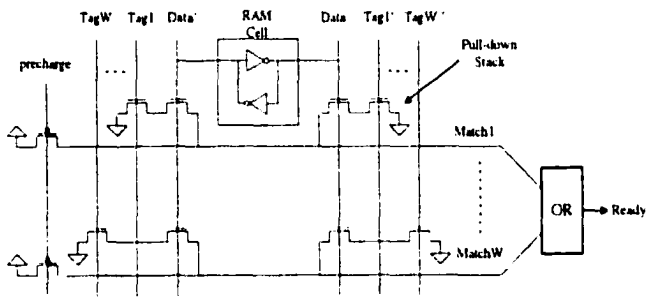

Figure 2.4: Core cell of wakeup logic modeled as a CAM.

As an example, Figure 2.4 depicts the core cell of the instruction wakeup logic which we model in our CPU as a form of the CAM structure. As described above, the key sizing parameters in this CAM are: (i) the issue/commit width of the machine (number of match or tag lines in each core cell, depicted by the parameter W in the figure), (ii) the instruction window size (which impacts the CAM's overall height) and (iii) the physical register tag size which equals logarithm base 2 of instruction window size (which impacts the CAM's width). Vertically, each core cell is replicated InstructionWindowSize times. Horizontally, the number of cells will equal the number of bits in the physical register tag; they share a common wide-OR for the final match which signals that the instruction is ready to issue. We also model the wordlines which are used to write new tag values into the CAM structure; for simplicity, these

lines are omitted from the figure.

Complex Logic Blocks Two of the larger complex logic blocks that we consider are the instruction selection logic (in the instruction window) and the dependency check logic (in the register renaming unit). We model circuit structures based on the selection logic described in [69] and the dependency check logic in [12].

We model the power consumption of result buses by estimating the length of the result buses using the same assumptions about functional unit height made in [70]. These lengths are multiplied by the metal capacitance per unit length. This equation is shown in Table 2.1.

Modeling the power consumption of the functional units (ALUs) at this high level would be difficult. Previous work has investigated the power consumption of various functional units [13; 98]. We scale the power numbers from these combinational structures for process and frequency in order to estimate the power consumption of the functional units.

Clocking The clocking network on high performance microprocessors can be the most significant source of power consumption. We consider three sources of clock power consumption:

- Global Clock Metal Lines: Long metal lines route the clock throughout the processor. We model a modified H-tree network in which the global clock signal is routed to all portions of the chip using equivalent length metal wires and buffers in order to reduce clock skew. This is similar to that used in the Alpha 21264 [36].
- Global Clock Buffers: Very large transistors are used to drive the clock throughout the processor in a timely manner. We estimate the size and number of these transistors from [15; 36] which describes the methodology used in the Alpha

21164 and 21264 for designing the global clock buffers.

- Clock Loading: We consider both explicit and implicit clock loading. Explicit clock loads are the values of the gate capacitances of pre-charge transistors and other nodes that are directly connected to the clock within the units that we model. Implicit clock loads include the load on the clock network due to pipeline registers. Here we use the number of pipeline stages in the machine and estimate the number of registers required per pipestage.

The models described above were implemented as a C program using the *cacti* tool [96] as a starting point. These models use SimpleScalar's hardware configuration parameters as inputs to compute the power consumption values for the various units in the processor. A summary of major hardware structures and the type of model used for each is given in Table 2.2.

Hardware Structure	Model Type
Instruction Cache	Cache Array (2x bitlines)
Wakeup Logic	CAM
Issue Selection Logic	Complex combinational
Instruction window	Array/CAM
Branch Predictor	Cache Array (2x bitlines)
Register File	Array (1x bitlines)
Translation Lookaside Buffer	Array/CAM
Load/Store Queue	Array/CAM
Data Cache	Cache Array (2x bitlines)
Integer Functional Units	Complex combinational
FP Functional Units	Complex combinational
Global Clock	Clock

Table 2.2: Common CPU hardware structures and the model type used by Wattch.

2.2.2 SimpleScalar Interface

The power models are interfaced with SimpleScalar, which keeps track of which units are accessed per cycle and records the total energy consumed for an application. We use a modified version of SimpleScalar's *sim-outorder* to collect results.

SimpleScalar provides a simulation environment for modern out-of-order processors with 5-stage pipelines: fetch, decode, issue, writeback, and commit. Speculative execution is also supported. The simulated processor contains a unified active instruction list, issue queue, and rename register file in one unit called the reservation update unit (RUU) [84]. Separate banks of 32 integer and floating point registers make up the architected register file and are only written on commit. We have extended SimpleScalar to provide for a variable number of additional pipestages between fetch and issue bringing the number of pipestages more in line with current microprocessors. In this study, we assume three additional pipestages between fetch and issue, and seven cycles of mispredict penalty.

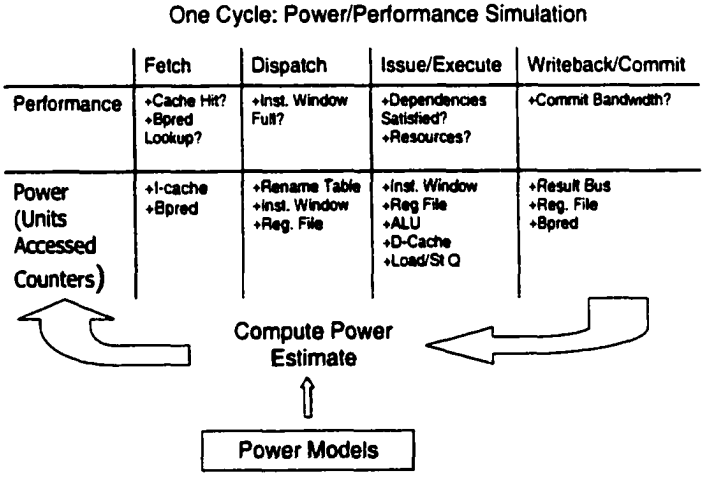

Figure 2.5: One Cycle in Wattch.

Our power-oriented modifications track which units are accessed on each cycle and how. For example, if a particular cycle involves reading the instruction cache,

selecting some ready instructions from the RUU, reading on two ports of the register file, and performing two integer additions, then counters in the simulator will record these events and a power estimate for each of these structures will be recorded on each cycle of the simulation. This is illustrated in Figure 2.5. Some of the power models vary the estimated power based on the number of ports used, as described in Section 2.2.2. As with most simulation frameworks, we hope that broader distribution of the framework will lead users to create an even richer variety of power modeling modules over time.

Section 2.3.1 describes further details of the baseline hardware parameters selected and the benchmarks we use.

Conditional Clocking Styles

One key issue that arises in estimating power concerns how to scale power consumption for multi-ported hardware units. Current CPU designs increasingly use conditional clocking to disable all or part of a hardware unit to reduce power consumption when it is not needed. In this work we consider three different options for clock gating to disable unused resources in multi-ported hardware. (More options can clearly be developed later; we give these as initial examples.)

The first and simplest clock gating style assumes the full modeled power will be consumed if any accesses occur in a given cycle, and zero power consumption otherwise. For example, a multi-ported register file would be modeled as drawing full power even if only one port is used. This assumption is realistic for many current CPUs that choose not to use aggressive conditional clocking. The second possibility assumes that if only a portion of a unit's ports are accessed, the power is scaled linearly. For example, if two ports of a 4-port register file are used in a given cycle, the power estimate returned will be one-half of the power compared to if four ports are used. Wattch tracks how many ports are used on each hardware structure per

cycle and scales power numbers accordingly. In practice, it may be impossible to totally shut off the power to a unit or port when it is not needed, so a small fraction of its total power may still be active. With this in mind, we also present a third option in which power is scaled linearly with port or unit usage, except that unused units dissipate 10% of their maximum power, rather than drawing zero power. This number was chosen as it represents a typical turnoff figure for industrial clock-gated circuits.

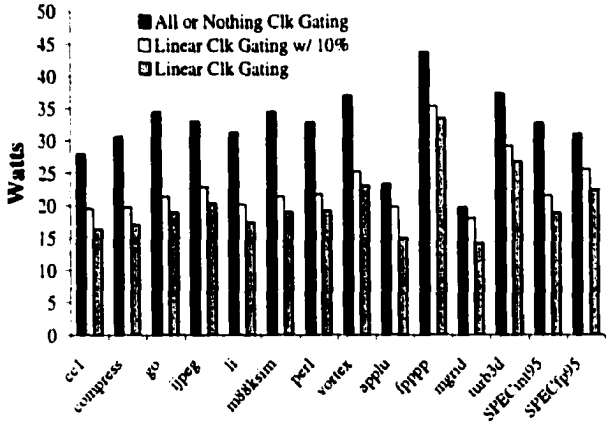

Figure 2.6: Power consumption of benchmarks with conditional clocking on multi-ported hardware. The first bar assumes simple clock gating where a unit is fully on if *any* of its ports are accessed on that cycle, or fully off otherwise. The second bar assumes clock gating where the power scales linearly with port usage, and disabled ports consume 10% of their maximum power. The third bar assumes ideal clock gating where the power scales linearly with port usage as in the second bar, but disabled units are entirely shut off.

Figure 2.6 shows the power dissipation for the eight SPECint95 and four of the SPECfp95 benchmarks for the three styles of conditional clocking. The maximum power for this configuration (similar to the 21264) was 58.4W. Future processors are likely to move towards the more aggressive style to reduce the average power dissipation. We expect that there will be more variability in the power consumption of the benchmarks when more clock gating is used. This assumption is supported by this

figure: for the simple clock gating style, the maximum variation of the benchmarks from the average is 36%. The variations for the more advanced clock gating techniques are 54% and 66%. The amount of clock gating in current processors falls somewhere between the styles that we consider.

Simulation Speed

Wattch is intended to run with overheads only moderately larger than other SimpleScalar simulators. It first computes the base power dissipation for each unit at program startup, which is a one-time cost. These base power costs are then scaled with per-unit access counts. For arithmetic units, we only charge power for the units that would be used each cycle; a cycle that performs two integer additions will not be charged for the multiply unit. In addition to the access counts, the simulator also scales power estimates for multi-ported hardware based on the style of clock gating chosen from the options given in Section 2.2.2.

Our simulation speed measurements are for our modified version of SimpleScalar/AXP's *sim-outorder* running on a Pentium-II 450MHz PC using RedHat Linux version 6.0. The simulation speed of sim-outorder without power modeling was approximately 105K instructions per second. With our current methodology which updates the power statistics every cycle according to access counts, we see roughly a 30% overhead on average compared to performance simulation alone. That is, our simulation speed drops to roughly 80K instructions per second. Given the ability to gauge power at a fairly high level, we feel this overhead is quite tolerable. It can be further reduced, however, by updating power statistics every few cycles. This would require loosening the accuracy of the port count statistics and the statistics on the usage of different functional units.

As a comparison to lower level tools, running PowerMill on a 64-bit adder for 100 test vectors takes approximately one hour. In the same amount of time, Wattch can

simulate a full CPU running roughly 280M SimpleScalar instructions and generate both power and performance estimates.

2.3 Case Study

In this section, we provide three case studies that demonstrate how Wattch can be used to perform architectural or compiler research. When performing power studies, a variety of metrics are important depending on the goals. Our simulator provides results for several of these metrics:

- Power: The average and maximum per-cycle power consumption of the processor are important because power translates directly into heat. With on-chip thermal sensors, techniques such as instruction cache throttling can be used to reduce the number of cycles in which the power consumption is significantly above the average [78]. Large cycle-by-cycle swings in the power dissipation (i.e., power glitches) are also important because they cause reliability problems. Our cycle-level power simulator is capable of analyzing these types of problems.
- Performance: The performance ramifications of an architectural proposal, whether positive or negative, are important for any architecture study. With Wattch, performance is measured in terms of number of cycles for program execution.
- Energy: The overall energy consumption of a program is equal to power dissipation multiplied by the execution time. Overall energy consumption is important for portable and embedded processors, where battery life is a key concern.
- Energy-Delay Product: The energy-delay product, proposed by Gonzalez and Horowitz [40], multiplies energy consumption and overall delay into a single metric. This produces a metric that does not give unwarranted preference to solutions that are either (1) very low-energy but very slow, or (2) very fast but very high power.

In the following subsections we present three case studies which demonstrate how the simulator infrastructure can be used for architecture and compiler research studies. The case studies illustrate the three possibilities shown in Figure 2.1. The main point of these case studies is to demonstrate the methodology for rapid exploration of these ideas, rather than to give details on each of the examples themselves. Before getting into the case studies, we explain our baseline hardware configuration and benchmarks.

2.3.1 Simulation Model Parameters

Unless stated otherwise, our results in this chapter, as well as for the rest of this thesis, model a processor with the configuration parameters shown in Table 2.3. These baseline configuration parameters roughly match those of the Alpha 21264 processor. The main difference is that the 21264 has a separate active list, issue queue, and rename register file while the SimpleScalar simulator uses a unified instruction window called an RUU. For technology parameters, we use the process parameters for a .35um process at 600MHz. In this section, we use Section 2.2.2's aggressive clock gating style (linear scaling with number of active ports) for all results.

Benchmark Applications

We chose to evaluate our ideas on programs from the SPECint95 and SPECfp95 benchmark suites. SPEC95 programs are representative of a wide mix of current integer and floating-point codes. We have compiled the benchmarks for the Alpha instruction set using the Compaq Alpha *cc* compiler with the following optimization options as specified by the SPEC Makefile: -migrate -std1 -O5 -ifo -non_shared.

For each program, we simulate 200M instructions. We select a 200M instruction window not at the beginning of the program by using warmup periods as discussed

Parameter	Value
Processor Core	
RUU size	80 instructions
LSQ size	40 instructions
Fetch Queue Size	8 instructions
Fetch width	4 instructions/cycle
Decode width	4 instructions/cycle
Issue width	4 instructions/cycle (out-of-order)
Commit width	4 instructions/cycle (in-order)
Functional Units	4 Integer ALUs
	1 integer multiply/divide
	1 FP add, 1 FP multiply
	1 FP divide/sqrt
Branch Prediction	
Branch Predictor	Combined, Bimodal 4K table
	2-Level 1K table, 10bit history
	4K chooser
BTB	1024-entry, 2-way
Return-address stack	32-entry
Mispredict penalty	7 cycles
Memory Hierarchy	
L1 data-cache	64K, 2-way (LRU)
	32B blocks, 1 cycle latency
L1 instruction-cache	64K, 2-way (LRU)
	32B blocks, 1 cycle latency
L2	Unified, 2M, 4-way (LRU)
	32B blocks, 12-cycle latency
Memory	100 cycles
TLBs	128 entry, fully associative
	30-cycle miss latency

Table 2.3: Baseline Configuration of Simulated Processor

in [81].

2.3.2 A Microarchitectural Exploration

One important application of Wattch is for microarchitectural tradeoff studies that account for both performance and power. For example, users may be interested in evaluating sizing tradeoffs between different hardware structures. Clearly, the architectural decisions made when power is considered may differ from those based solely on performance. One possible study which we consider in this section is to evaluate size tradeoffs between the RUU and data cache. This example demonstrates Scenario A from Figure 2.1. The baseline processor configuration is that from Table 2.3 and our simulations vary the sizes of the RUU and D-cache. For all simulations, the Load/Store Queue is set to half the size of the RUU. We have collected these results for the SPECint95 and several of the SPECfp95 benchmarks. As we will discuss below, the results typically fall into two main categories of behavior, and we present results for one representative benchmark from each category: *gcc* and *turb3d*.

Figures 2.7, 2.8 and 2.9 show the results for the *gcc* benchmark. The three graphs show performance (in instructions per cycle), average power dissipation, and energy-delay product for the benchmark. Similarly, Figures 2.10, 2.11 and 2.12 show the same results for the *turb3d* benchmark.

The IPC graphs show that *gcc* gets significant performance benefit from increasing the data cache size. It only begins to level off at roughly 64KB. In contrast, *turb3d* gets relatively little performance benefit from increasing the data cache size, but is highly sensitive to increases in the RUU size.

Although the performance contours are fairly different for these two benchmarks, the power contours shown in Figures 2.8 and 2.11 are quite similar. Both show steady increases in average power as the size of either unit is increased.

Despite the similarity in the average power graph, the two benchmarks do have

strikingly different energy characteristics, as shown in Figures 2.9 and 2.12. The energy-delay product combines performance and power consumption into a single metric in which lower values are considered better both from a power and performance standpoint. The energy-delay product curve for *gcc* reaches its optimal point for moderate (64KB) caches and small RUUs. This indicates that although large caches continue to offer *gcc* small performance improvements, their contribution to increased power begins to outweigh the performance increase. RUU size offers little benefit to gcc from either a performance or energy-delay standpoint.

For *turb3d*, energy-delay increases monotonically with cache size, reflecting the fact that larger caches draw more power and yet offer this benchmark little performance improvement in return. Moderate-sized RUU's offer the optimal energy-delay for *turb3d*, but the valley in the graph is not as pronounced as for *gcc*.

Overall, the point of this case study is to demonstrate how the power simulator and the resulting graphs shown can help explore tradeoff points taking into account both power and performance related metrics.

2.3.3 Power Analysis of Loop Unrolling

This section gives an example of how a high-level power simulation can be of use to compiler writers as well. We consider a simple case study which examines the effects of loop unrolling on processor power dissipation. Loop unrolling is a well-known compiler technique that extends the size of loop bodies by replicating the body n times, where n is the unrolling factor. The loop exit condition is adjusted accordingly. In this section, we consider a simple matrix multiply benchmark with 200x200 entry matrices. We have used the Compaq Alpha *cc* compiler to unroll the main loops in the benchmark, and we consider several unrolling factors.

Figures 2.13 shows the results for the execution time and power/energy results for loop unrolling. As one would hope, the execution time and the number of total

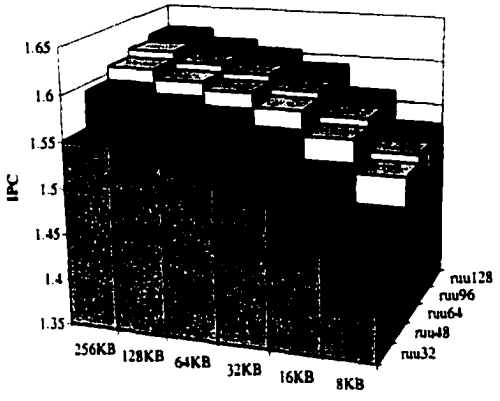

Figure 2.7: IPC for *gcc* when varying RUU and Data Cache size.

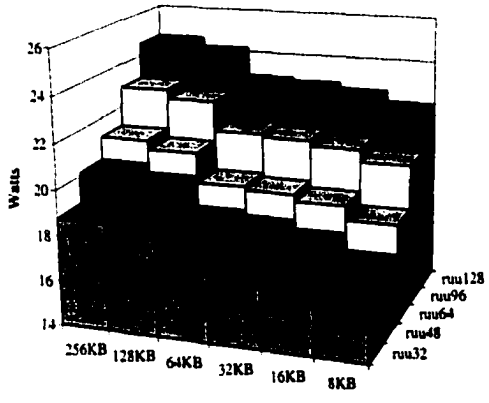

Figure 2.8: Power for *gcc* when varying RUU and Data Cache size.

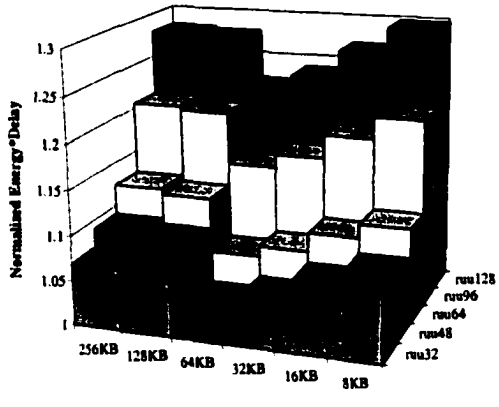

Figure 2.9: Energy-Delay Product for *gcc* when varying RUU and Data Cache size.

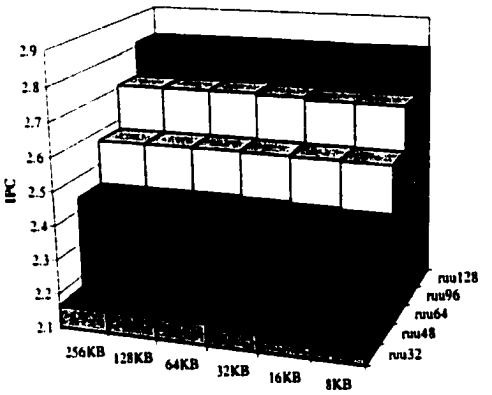

Figure 2.10: IPC for *turb3d* when varying RUU and Data Cache size.

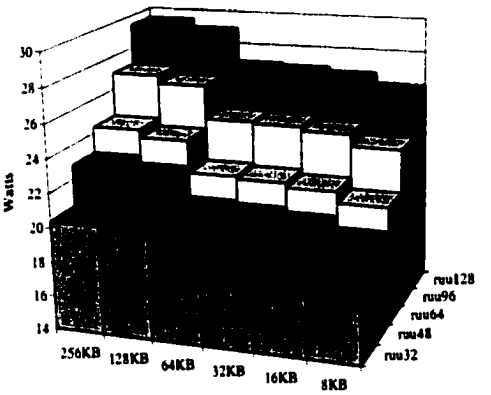

Figure 2.11: Power for *turb3d* when varying RUU and Data Cache size.

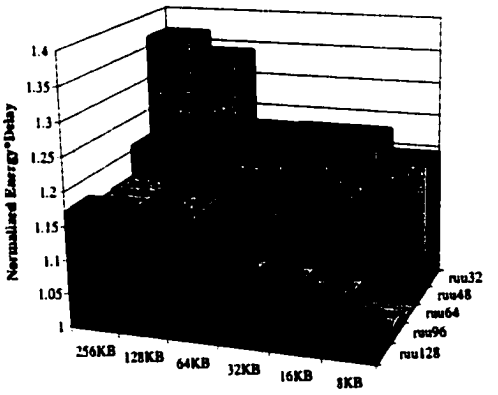

Figure 2.12: Energy-Delay Product for *turb3d* varying RUU and Data Cache size.

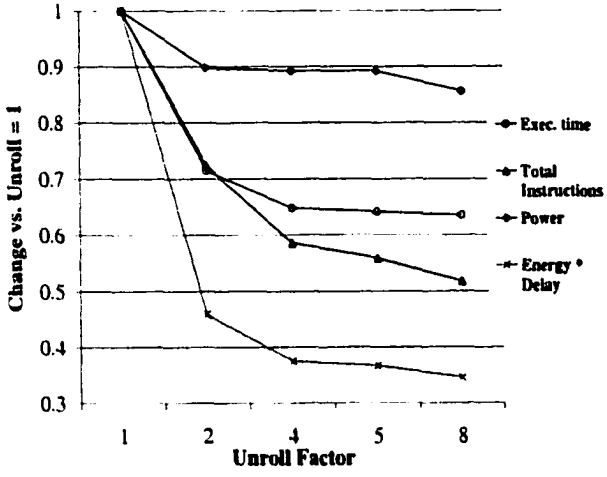

Figure 2.13: The effects of loop unrolling on performance and power. Note that the branch prediction direction accuracy decreases from 99.5% for no unrolling to 95.1% when unrolling 8 times.

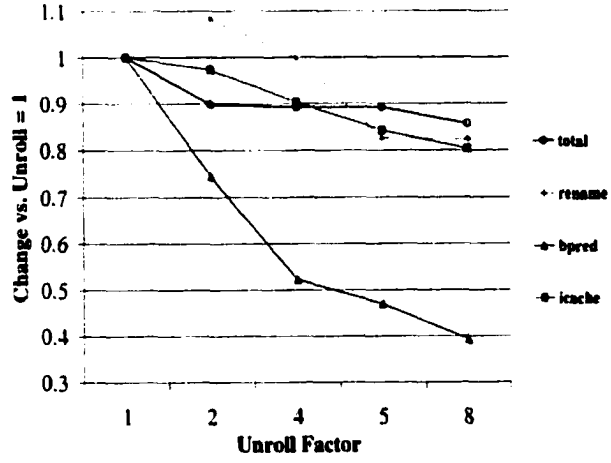

Figure 2.14: Detailed Breakdown of Power Dissipation.

instructions committed decreases. This is because loop unrolling reduces loop overhead and address calculation instructions. The power results are more complicated, however, which makes the tradeoffs interesting in a power-aware compiler.

Figure 2.14 shows a breakdown of the power dissipation of individual processor units normalized to the case with no unrolling. There are two important side-effects of loop unrolling. First, loop unrolling leads to decreased branch predictor accuracy, because the branch predictor has fewer branch accesses to "warm-up" the predictors and because mispredicting the final fall-through branch represents a larger fraction of total predictions.

Another side-effect of loop unrolling is that removing branches leads to a more efficient front-end. The fetch unit is able to fetch large basic blocks without being interrupted by taken branches. This, in turn, provides more work for the renaming unit and fills up the RUU faster. In fact, with this example the RUU becomes full for an average of 85% of the execution cycles after we move from an unrolling factor of 2 to 4. The fetch queue, which connects the fetch unit to the renaming hardware is also affected, and is full for an average of 73% of the cycles at an unrolling factor of 4.

Thus, the average fetch unit power dissipation decreases for two reasons. First, because the branch prediction accuracy has decreased, there are more misprediction stall cycles in which no instructions are fetched. The second reason is that at larger unrolling factors, the fetch unit is stalled during cycles when the instruction queue and RUU are full. The reduced number of branch instructions also significantly reduces the power dissipation of the branch prediction hardware. (Note that these stall cycles would increase the total energy required to run the full program, but this graph shows average power.)

The renaming hardware, on the other hand, shows a small increase in power dissipation at an unrolling factor of two. This is because the front-end is operating at

full-tilt, sending more instructions to the renamer per cycle. As the fetch unit starts to experience more stalls with unrolling factors of 4 and beyond, the renamer unit also begins to remain idle more frequently, leading to lower power dissipation.

While the varied power trends in each unit are somewhat complicated, the overall picture is best seen in Figure 2.13. The total instruction count for the program continues to decrease steadily for larger unrolling factors, even though the execution time tends to level out after unrolling by four. The combined effect of this is that energy-delay product continues to decrease slightly for larger unrolling factors, even though execution time does not. Thus, a power-aware compiler might unroll more aggressively than other compilers. This simple example is intended to highlight the fact that design choices are slightly different when power metrics are taken into account; Wattch is intended to help explore these tradeoffs.

2.3.4 Memoing To Save Power

Another important application for the Wattch infrastructure is in evaluating the potential hardware benefits of hardware optimizations. In this section, we consider result memoing, a technique that has been previously explored for performance benefits [30; 83]. Memoing is the idea of storing the inputs and outputs of long-latency operations and re-using the output if the same inputs are encountered again. The memo table is looked up in parallel with the first cycle of computation, and the computation halts if a hit is encountered. Thus memoing can reduce multi-cycle operations to one-cycle when there is a hit in the memo table.

We consider the power and performance benefits of this technique. Power consumption in the floating point units is reduced during memo table hits. On the other hand, the memo tables dissipate additional power. We base our analysis on [30], which showed that a small 32-entry, 4-way set associative table is capable of achieving reasonable hit rates.

Azam et al. have investigated a similar technique for saving power within integer multipliers [6]. Their work did not model the additional power dissipation of reads and writes to the cache structure (only the tag comparison logic) and concentrated on integer and multimedia benchmarks. The point of this section is to demonstrate the methodology for using Wattch to perform such a study.

We have inserted memo tables in parallel with the floating-point and integer multipliers (4 cycles), the floating point adder (4 cycles), and the floating point-divider (16-cycles, unpipelined). Citron's study examined the SPECfp95, Perfect, and a selection of multimedia and DSP applications finding that the multimedia applications have the lowest local entropy in result values and hence the highest hit rates. Since Citron's multimedia benchmarks were not readily available, we have examined a selection of benchmarks from the SPECfp95 suite. As in Citron's work, we do not enter "trivial" operations such as multiply/divide by 0/1 into the table, because we assume that simpler hardware could recognize and capitalize on these opportunities.

The modifications to the power simulator infrastructure for the new hardware were not complex. The behavior of the memo tables was implemented and *memo-table-lookup* and *memo-table-write* routines were inserted in the simulator pipeline. Both of these routines also serve as the access counters for the memo tables. The memo tables were modeled as simple cache array structures using the same power models that the other array structures use.

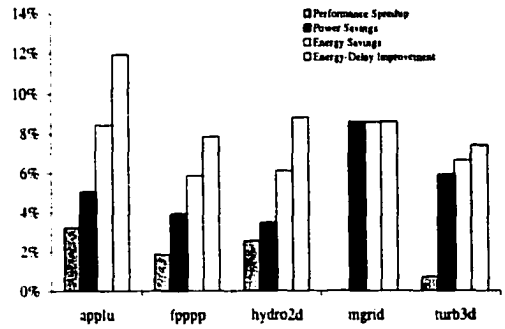

Figure 2.15: Performance and Power Effects of Memoing Technique.

Figure 2.15 shows the performance and power results for the memoing technique. The benchmarks showed an average speedup of 1.7% and an average power improvement of 5.4%. The larger power benefits of the memoing techniques are most likely due to the dynamically scheduled out-of-order execution core of the simulated processor. In an out-of-order processor, the delay of long-latency operations can often be hidden by finding other instructions to execute, thus the performance benefits of removing long-latency operations are not too large. However, stopping these operations after one cycle of execution can have a significant impact on power dissipation. This is most apparent in *mgrid*, which shows almost no performance benefit, but just over an 8% power benefit.

2.4 Chapter Summary

This chapter has described Wattch, a simulator framework that can be used to evaluate a wide range of architectural and compiler techniques. Wattch has the benefit of low-level validation against industry circuits, while opening up power modeling to researchers at abstraction levels above circuits and schematics. In addition, because of the fully parameterizable power models that have been developed, Wattch is ideally suited for exploring entirely brand new microarchitectures.

Wattch still has room for improvement and we hope that exposure and distribution to the architectural community will lead to the development of additional modules. Additional accuracy validations are important, and we plan to compare the models against lower-level tools on more designs. Speed-accuracy tradeoffs for signal activity factors are another area we will consider in the future.

Extensions to the simulator infrastructure and the creation of additional modules are topics of future research. The simulator infrastructure could be extended to consider different hardware organization styles. Additional power modules could

be developed with different circuit-implementation styles targeting different power-performance targets. Modeling off-chip communication is also an important module to be developed. Additional work can also focus on more automatic transistor sizing and the effects of future process technologies, including leakage power dissipation.

We see a wide range of power studies that can be performed with Wattch. First, many old techniques may take on a new light when power is considered as a metric. The memoing case study described in Section 2.3.4 is one example of this. Other interesting techniques to study with power as a metric would be value-prediction [37] and instruction pre-processing [48]. The effects of the compiler techniques and operating system control on power dissipation, including the use of power dissipation as feedback in a profiling compiler, are another possible research area. Finally, Wattch can be used in power studies which explore techniques that focus on micro-architectural solutions to lower-level power problems. One example of this is dynamic thermal management techniques to reduce power dissipation when thermal emergencies occur due to high-power sections of applications. An evaluation of this technique using Wattch will be discussed in Chapter 6. Another example would be the evaluation and development of solutions for large, short-term, current spikes due to clock gating, which can cause problems with chip reliability.

Exploring these classes of ideas in the power domain will open up new research possibilities for architects. The Wattch simulator infrastructure described in this chapter offers a starting point for such research efforts.

Chapter 3

PowerTimer

The Wattch framework presented in Chapter 2 was one of the first tools to link a traditional architectural performance simulator with energy models. This link is accomplished by sending both static information describing the simulated microarchitecture and dynamic information about the run-time characteristics of applications to the energy models. This chapter describes the PowerTimer infrastructure [22], an effort to apply the Wattch methodology to industrial performance simulators with energy models developed from circuits built for a high-performance commercial microprocessor.

PowerTimer is an ongoing project within IBM Research to develop a power-performance modeling toolkit, developed to aid in the evaluation and definition of future power-efficient, PowerPC™ processors. The power-performance modeling methodology described in the previous sections of this chapter is adapted for use within the modeling framework of a real, server-class processor development project. The key new contributions in this power-performance modeling tool are:

- Energy models that are derived from real, circuit-level power simulation data, but are then driven by microarchitecture-level parameters of interest. These higher-level abstractions are suitable for conducting power-performance tradeoff studies to define the follow-on design points within a given product family.

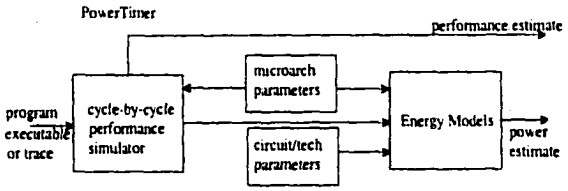

Figure 3.1: Block Diagram of PowerTimer.

Technology parameters and scaling equations are additional inputs to the model.

- A web-based graphical user interface, which allows one to quickly characterize the fundamental tradeoffs between performance growth and power-related cost, based on prior, one-time simulation data collected in a spreadsheet database.

Using this new modeling toolkit, we evaluate a current generation, high-end PowerPC processor design point from the viewpoint of power-performance efficiency. As part of this evaluation, we examine the sensitivity of such efficiency metrics with respect to individual (and combinations of) microarchitecture-level parameters: cache size and geometry parameters, queue/buffer sizes, number of ports to various storage resources, various other bandwidth parameters, etc.

3.1 *PowerTimer*: An Energy-Aware Performance Simulation Toolkit

Figure 3.1 shows the high-level block diagram of PowerTimer, our energy-model-enabled performance simulator. The basic methodology is similar to earlier models like Wattch [21].

The energy models are derived from circuit-level power simulation data, collected on a detailed, macro-by-macro basis. These models are controlled by two sets of parameters: (a) technology/circuit parameters, which allow appropriate scaling from

one CMOS generation to the next; and (b) microarchitecture-level parameters: various queue/buffer sizes, pipe latencies and bandwidth values. These latter parameters also drive the base performance simulator in the usual manner. The energy models can be used in two different modes. First, the performance simulator can be used standalone, to produce detailed CPI and resource utilization statistics. These can then be processed through the energy models to generate average, unit-wise power numbers. Second, the energy models can be embedded in the actual simulation code, so that they are "looked up" as needed on a cycle-by-cycle basis. This mode allows one to view the cycle-by-cycle energy characteristics in more detail; but the average statistics at the end of the run would obviously be the same as in the first mode.

3.1.1 Energy Model Construction

In the Wattch simulator [21], and in other similar toolkits [94; 99], analytical capacitance models were developed for various high-level block-types, such as RAMs, CAMs and other array structures, latches, buses, caches, and ALUs. While some of the characterizing parameters are gross length and width values which a logic-level designer or microarchitect can relate to, others are at a much lower (circuit or physical design) level. In the PowerTimer work, the goal is to form unit-specific energy models controlled by parameters familiar to a high-level designer or microarchitect. Thus, for example, once a characterizing equation has been formed for one of the issue queues, one is able to play "what-if" games in PowerTimer, by simply varying the queue size as normally done in microarchitectural performance simulation. The major difference between PowerTimer and Wattch is in the formation of energy models. PowerTimer's energy models are formed from empirical data collected from an existing, commercial microprocessor. In Wattch, low-level analytical capacitance equations are generated for major nodes within common hardware structures. Thus Wattch takes a top-down approach using analytically derived capacitance equations from known structures of

specific modules. PowerTimer is more of a bottom-up approach which uses existing, low-level circuit macros to generate higher-level energy models for microarchitectural units.

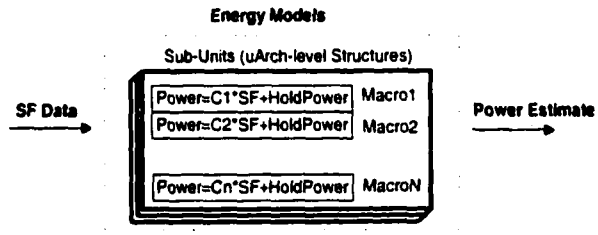

Figure 3.2: PowerTimer Energy Models.

Figure 3.2 depicts the derivation of the energy models in more detail. The energy models are based on circuit-level power analysis that has been performed on structures in a current, high performance PowerPC processor. The power analysis has been performed at the macro level; generally, multiple macros combine to form one micro-architectural level structure (super-macro). For example, the fixed-point issue queue (one super-macro) might contain separate macros for storage memory, comparison logic, and control. Power analysis has been performed on each macro to determine the macro's power as a function of the input switching factor. The *hold power*, or power when no switching is occurring, is also generated. These two pieces of data allow us to form simple linear equations for each macro's power. The energy model for a super-macro is determined by summing the linear equations for each macro within that structure. We have generated these power models for all microarchitecture-level structures modeled in our research simulator [64; 65].

In addition to the models that specify the power characteristics for particular super-macro (such as the fixed-point issue queue), we can derive power models for more generalized structures; for example, a generalized issue queue model. These

generalized models are useful for estimating the power cost of additions to the baseline microarchitecture. The generalized model is derived by analyzing the power characteristics of structures within the baseline microarchitecture. For example, the fixed-point, floating-point, logical-op, and branch-op queues have very similar functionality and power characteristics and the energy analysis for these queue structures has been used to derive a generalized issue-queue power model based on parameters such as the number of entries, storage bits, and comparison operations.

Since we are interested in determining power-performance tradeoff analysis for future microarchitectures within a particular product family, we must determine a method of scaling the power of microarchitectural structures as the size of these structures increases. The scaling factor depends on the particular structure; for example, the power of a cache array will scale differently than that of an issue queue. In addition, as resources increase in size, they necessarily cause other structures to become larger. For example, as the number of rename registers increases, the number of tag bits within each entry of the issue queues increases. Generally, as we increase the number of entries in a structure, there will be a proportional increase in the power. For this reason, we use linear scaling as a basis for many of the structures that we consider. In addition, we have performed detailed analysis on the scaling of queue and mapper structures. For these structures, we have determined the average power per storage bit and per comparison operation. As the queues and mappers increase in size, we compute the number of storage bits and comparisons that occur for the larger structures. We also use previously published work on power scaling within cache arrays which we discuss in Section 3.2.3.

3.1.2 Web-Based Interface and Power-Performance Metrics

In order to thoroughly explore the modeled design space, we selected 19 workloads (8 SPECint95, 10 SPECfp95, and TPC-C) each of which was evaluated for over

75 hardware configurations. Analyzing this amount of data is difficult and a GUI makes the results of our analysis more useful. We developed a web-based back-end analysis tool which allows the user to select the benchmarks of interest and the microarchitectural parameter(s) to vary as well as the technology parameters such as frequency, voltage, and feature size.

The tool also allows the selection of various power-savings features such as the style of conditional clocking within the microarchitecture. Finally, the tool provides the choice of five power-performance metrics: Average CPI, average power dissipation, $CPI*power$, $(CPI)^2*power$, and $(CPI)^3*power$. The latter three metrics correspond to energy, energy-delay product [31; 40], and $energy * delay^2$ [16]. In the remainder of this section we will present our power-performance results as $(CPI)^3 * power$.

3.2 Power-Performance Evaluation Examples

In this section, we first provide a high-level description of the processor model assumed in our simulation toolkit. Then, we present some example experimental results with analysis and discussion. The results were obtained using our current version of PowerTimer, which works with pre-silicon performance models used in defining future PowerPC structures.

3.2.1 Base Microarchitecture Model

We assume a generic, parameterized, out-of-order superscalar processor model adopted in a research simulator called Turandot [64; 65]. The overall pipeline structure (as reported in [64]), is repeated here in Figure 3.3. The modeled microarchitecture is similar in complexity to a current generation microprocessor (e.g. [34; 63]). As described in [64], this research simulator was calibrated against a pre-RTL, detailed, latch-accurate processor model (referred to as R-model in [64]). The R-model is a

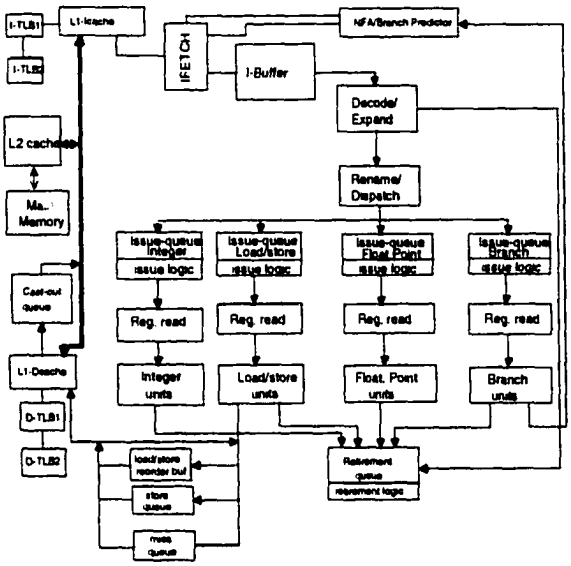

Figure 3.3: Processor Organization Modeled by the Turandot Simulator.

custom simulator, written in C++ (with mixed VHDL "interconnect code"). There is a 1-to-1 correspondence of signal names between the R-model and the actual VHDL (RTL) model. However, the R-model is about two orders of magnitude faster than the RTL model and is considerably more flexible. Many microarchitecture parameters can be varied, albeit within restricted ranges. Turandot, on the other hand is a classical trace/execution-driven simulator, written in C, which is 1-2 orders of magnitude faster than R-model. It supports a much greater number and range of parameter values.

We report power-performance results using the same version of R-model that was used in [64]. That is, we first used our energy models in conjunction with the R-model: this ensured accurate measurement of the resource utilization statistics within the machine. To circumvent the simulator speed limitations, we used a parallel workstation cluster; also, we post-processed the performance simulation output and fed the average resource utilization statistics to the energy models to get the average power numbers. This is faster than the alternative of looking up the energy models on every

cycle. While it would have been possible to get instantaneous, cycle-by-cycle energy consumption profiles through such a method, it would not have changed the average power numbers for entire program runs. Having used the detailed, latch-accurate reference model for our initial energy characterization, we were able to look at the unit- and queue-level power numbers in detail in order to understand, test and refine the various energy models. Currently, we have reverted to using an energy-model-enabled Turandot model, for fast CPI vs. Power tradeoff studies with full benchmark traces. Turandot allows us to experiment with a wider range and combination of machine parameters. In future publications and talks based on PowerTimer, we plan to report these results in detail.

3.2.2 Workloads Used in the Study

In this section, we report experimental results based on the SPEC95 benchmark suite and a commercial TPC-C trace. All workload traces are PowerPC-based. The SPEC95 traces were generated using the tracing facility called *Aria* within the MET toolkit [65]. The particular SPEC trace repository used in this study was created by using the full reference input set. However, sampling was used to reduce the total trace length to 100 million instructions per benchmark program. A systematic validation study to compare the sampled traces against the full traces was done, in finalizing the choice of exact sampling parameters. The TPC-C trace used is a contiguous (i.e. unsampled) trace collected and validated by the processor performance team at IBM Austin. It is about 180 million instructions long.

In the following three sections we present examples of the use of the PowerTimer simulation infrastructure. The results show the average CPI and average $(CPI)^3 * power$ for the traces described above. Each SPEC data point was obtained by averaging across the benchmark suite. Note, however, that we have excluded *apsi* from the SPECfp results due to a problem with these simulation runs.

3.2.3 Data Cache Size and the Effect of Scaling Techniques

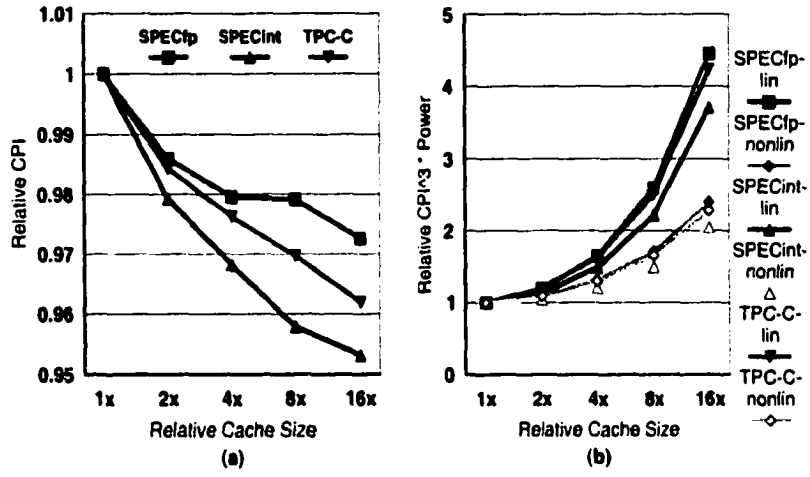

Figure 3.4: Variation of Performance and Power-Performance with Cache Size.

In this section we evaluate the relationship between performance, power, and L1 data cache size. We vary the cache size by increasing the number of cache lines per set while leaving the linesize and cache associativity constant. Figure 3.4a and 3.4b show the results of increasing the cache size from the baseline architecture (points labeled 1x on the x-axes). Figure 3.4a illustrates the relation between the cache size in the first-level data cache and the *relative* CPI for the workloads that we studied. The CPI value for each cache size is computed as a ratio, relative to the base 1x CPI for that workload. Figure 3.4b shows the relation when we consider the metric $(CPI)^3 * power$. From Figure 3.4a, it is clear that the small CPI benefits (note the small range on the relative CPI plot) of increasing the data cache are outweighed by the increases in power dissipation due to larger caches.

In Figure 3.4b, we show the results with two different scaling techniques. The first technique assumes that power scales linearly with the cache size. As the number of lines is doubled, the power of the cache is also doubled. The second scaling technique is based on data from [53] which studied energy optimizations within multi-level cache

architectures. In [53], data is presented for cache power dissipation for conventional caches with sizes ranging from 1KB to 64KB.

In the second scaling technique, which we call "non-lin" in Figure 3.4b, the cache power is scaled with the data showing power/performance results for many cache sizes presented in [53]. The increase in cache power by doubling cache size using this technique is roughly 1.46x, as opposed to the 2x with the simple linear scaling method. Obviously the choice of scaling technique can greatly impact the results. It is clear, however, that with either scaling choice, conventional performance-focused cache organizations will not scale in a power-efficient manner. (Note that the curves shown in Figure 3.4b assume a fixed circuit/technology generation; they are intended to show the effect of adding more cache to the current design.)

3.2.4 Number of Completion Buffers

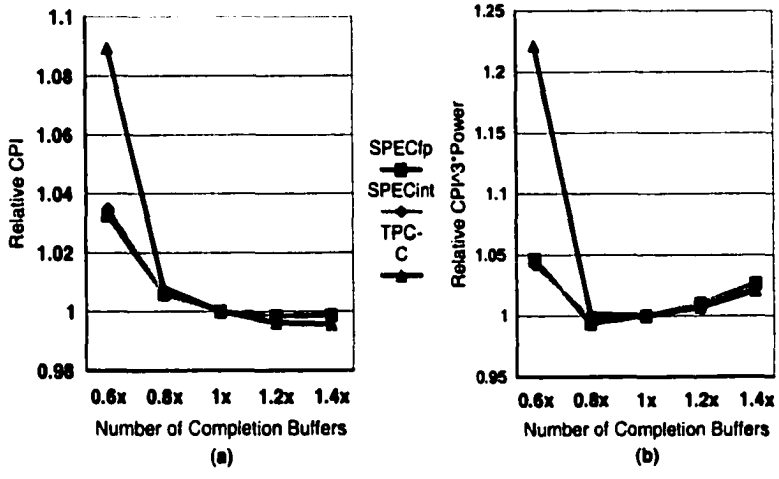

Figure 3.5: Variation of Performance and Power-Performance with Number of Completion Buffers.

In the target microarchitecture, the number of completion buffers determines the total number of instructions that can be active within the machine. The completion

table is very similar to a re-order buffer in that it tracks instructions as they dispatch, issue, execute, wait for exceptions, and complete. Figures 3.5a and 3.5b show the effects of varying the number of completion buffers on performance and the power-performance metric. From Figure 3.5a, it is evident that little additional performance is gained by increasing the number of buffers past the current design point (1x). When considering $(CPI)^3 * power$ in Figure 3.5b, we see that power-efficiency is slightly degraded by increasing the number of entries due to a roughly 3% increase in the core's power dissipation.

3.2.5 Ganged Sizing

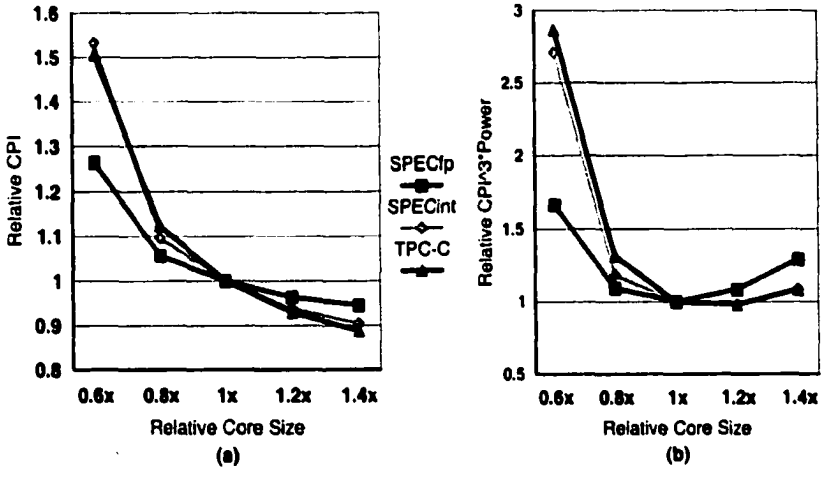

Figure 3.6: Variation of Performance and Power-Performance with Core Size (ganged parms).

The out-of-order superscalar processors we consider rely on queues and buffers to efficiently decouple instruction execution to increase performance. The depth of the pipeline and the sizes of the resources required to support decoupled execution (queues, rename registers, completion table) combine to determine the performance of the machine. Because of this decoupled execution style, increasing the size of one

resource without regard to the other resources in the machine may quickly create a performance bottleneck. Thus, in this section we consider the effects of varying multiple parameters rather than just a single parameter.

Figure 3.6a and 3.6b show the effects of varying all of the resource sizes within the processor core. This includes issue queues, rename registers, branch predictor tables, memory disambiguation hardware, and the completion table. For the buffers and queues, the number of entries in each resource is scaled by the values specified in the charts (0.6x, 0.8x, 1.2x, and 1.4x). For the instruction cache, data cache, and branch prediction tables, the size of the structures are doubled or halved at each data point. From Figure 3.6a, we can see that performance is increased by 5.5% for SPECfp, 9.6% for SPECint, and 11.2% for TPC-C as the size of the resources within the core is increased by 40% (except for the caches which are 4x larger). The configuration had a power dissipation of 52%-55% higher than the baseline core. Figure 3.6b, shows that the most power efficient core microarchitecture is somewhere between the 1x and 1.2x cores.

3.3 Chapter Summary

We have described PowerTimer: a research power-performance simulator designed to help with the definition and evaluation of follow-on products within the high-end PowerPC microprocessor family. Based on this model, we have evaluated power and performance tradeoffs using SPEC95 workloads and a TPC-C trace. We have presented a few selected experimental results from our analysis repository to illustrate the kinds of tradeoffs that one may be able to study using this toolkit. A web-based interface allows users to view specific power-performance tradeoff curves of their choice. This allows users to evaluate the worth and wisdom of making specific microarchitecture-level enhancements to an existing design point. The tool allows

one to evaluate whether a certain aspect of the design is inherently power-efficient or not. For example, in an initial, voltage-invariant "technology remap" scenario, we may like to know whether simply increasing the cache sizes, without perturbing the core engine would buy us enough performance to counterbalance any power increase.

Compared to Wattch, PowerTimer uses a very similar methodology for power estimation, although its energy models are based on existing circuits for an industrial microprocessor. PowerTimer's models are best suited for exploring microarchitectural tradeoff decisions building off of this core microarchitecture.

PowerTimer allows one to experiment with a large number of design parameters and there are multiple choices available in terms of selecting a power-performance efficiency metric. We have presented just a few examples in this section. For example, one can study the effectiveness of various flavors of conditional clocking to see how the sensitivity curves are affected. Also, the use of technology scaling parameters, allows the user to explore the future design space in a realistic manner.

Chapter 4

Power Model Validation

Validation is a critical part in the process of developing a model or simulator for any type of system. This phase is important to give the users of the model information about how reliable the model is under different operating conditions. This is especially important when abstractions have been used within the model to provide superior simulation speed, improved design space flexibility, or faster model construction.

Validation of architectural performance simulators is a challenging problem. Validating early-stage architectural power-performance simulators is even more difficult. In Section 4.1 we discuss the types of modeling error that we would like to quantify. In Section 4.2, we will describe three forms of validation of the relative and absolute accuracy of the Wattch infrastructure. In Section 4.3, we focus our validation efforts on quantifying the robustness of the *relative* accuracy of our power-performance simulators. We discuss why relative accuracy is sufficient for many interesting experiments, and we present results showing the effects of inserting artificial errors into our power models to demonstrate the relative accuracy for both Wattch and PowerTimer.

4.1 Types of Modeling Error

The two types of accuracy that we would like from an architectural-level power-performance simulator are *relative* and *absolute*. With relative accuracy, the simulator can estimate the proper ratio of power dissipation among all of the major components of the modeled architecture. Essentially, the simulator provides an accurate estimate of the fraction of the total power that each component uses. Absolute accuracy, on the other hand, requires relative accuracy but also requires that the magnitude of each component (or the total chip power) be estimated accurately.

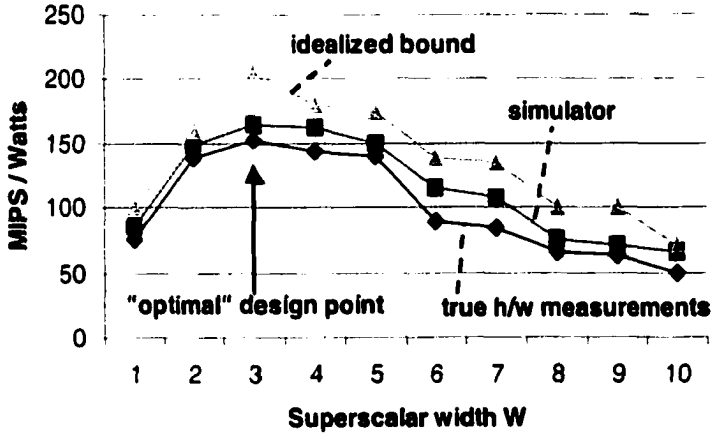

Figure 4.1: Relative accuracy in a design tradeoff study.

Achieving relative accuracy is much easier than achieving absolute accuracy, especially during the early-stages of the design process. This is because relative accuracy can be maintained despite errors in low-level technology parameters, incorrect assumptions about circuit-design styles, clocking network design methodologies, etc. Absolute accuracy will be degraded due to all of these conditions.

A simulator that ensures good relative accuracy still provides quite a bit of useful information to an architect. For example, design tradeoff studies with the goal

of choosing architectural parameters to achieve an optimal power-performance efficiency can easily be performed. Figure 4.1 demonstrates this with an example using contrived data. In this figure, a metric of power efficiency is plotted against the superscalar width of a processor allowing the CPU designer to choose the optimal superscalar width. The bottom-most curve in this chart is the MIPS/Watt of the processor as measured from a true-hardware design. The middle curve shows the values that the power-performance simulator gave during the architectural design stage. The top curve shows an upper-bound on power which could be generated through architected test cases or simple technology scaling equations. The point of this example is to show that while in some cases the absolute error can be quite significant, because the simulator maintains good relative accuracy, the chip architect could choose the correct design point, in this case a processor with a superscalar width of three.

This example demonstrates how relative accuracy is extremely useful for a chip architect's design decisions. This is not to say that absolute accuracy is not important at all. Absolute accuracy is primarily needed to estimate full chip-power for planning package and system power budgets. However, good relative accuracy, combined with useful upper-bounding techniques, could also help CPU designers with this problem.

4.2 Model Validation

Validating the power models is crucial because fast power simulation is only useful if it is reasonably accurate. In this section we provide three methods of validation. The first is a low-level check to compare the capacitance values generated from our model with those of real circuits. The second validation level aims at quantifying the *relative* accuracy of our model. Namely, we compare the relative power weights that our model generates with experimentally-measured results from published works on industry chips. The final validation technique seeks to quantify the absolute magnitude

accuracy of our models. With this method we compare the maximum processor power reported in published works with the power results of similar processor organizations generated from our models.

4.2.1 Validation 1: Model Capacitance vs. Physical Schematics

The parameterized power models presented in Section 2.2 obtain power dissipation estimates by calculating the capacitance values on critical nodes within common circuits. Thus, a low-level method for validating the models is to compare the capacitance value computed by the model, against circuit design tool calculations of capacitance values for industry schematics.

In this section, we describe this type of validation for a 128-entry, 64-bit wide register file structure with 8 read ports and 6 write ports. The physical register file schematic was selected from the actual design for one of Intel's IA-64 products. This type of large array structure is common in modern microprocessors and hence provides a good sample for our study.

% Change in Capacitance				
	Gate	Diffusion	InterConn.	Total
Wordline (r)	1.11	0.79	15.06	8.02
Wordline (w)	-6.37	0.79	-10.68	-7.99
Bitline (r)	2.82	-10.58	-19.59	-10.91
Bitline (w)	-10.96	-10.60	7.98	-5.96

Table 4.1: Percentage difference between lower-level tool capacitance values and the values estimated by our model.

Table 4.1 presents the results for validating the register file. We studied both the read and write nodes for the bitlines and wordlines in the register file. The table breaks down the capacitance for each of these into gate capacitance, diffusion

capacitance, and interconnect capacitance. For each entry, we present the percentage difference between the capacitance value estimated from a circuit-level capacitance extraction tool and the one calculated by our model. Most of the capacitance error rates are within +/-10%. The largest sources of error were within the interconnect capacitance. There are two reasons for this. First, the capacitance of polysilicon wires is difficult to model, because the lengths of these wires vary with the physical layout. Second, it is difficult to match the exact lengths of the interconnects in the physical schematic with the modeled nodes. For example, the wire in the bitline node in the physical schematic may extend beyond the length of the edge of the array structure, whereas our model assumes that the wire ends directly on the array boundary. Still, the total capacitance values are within 6-11% for the four nodes that were studied.

Array structures comprise roughly 50% of the total modeled chip power dissipation. Similar low-level validation could be performed on other hardware structures such as CAM arrays. We expect that the results will be similar, since the methodology for modeling these units is identical.

4.2.2 Validation 2: Relative power consumption by structure

Comparing low-level capacitance values is the most precise means of validating a power simulator. This method of validation has shown the models to be accurate within 10%, which is similar to what has been reported by the CACTI authors for analytical delay models [96] and later for analytical power models [76]. Amrutur and Horowitz have also studied analytical power and delay models for SRAMs [4].

To validate our models at a slightly higher level, we present a second set of validation data. This data compares relative power of different hardware structures predicted by our model against *published* power breakdown numbers available for several high-end microprocessors. The downside to this comparison is that we have

no way of knowing whether the design style we model for each unit matches the design style that they actually use. In spite of this downside, it is reassuring to see that these power breakdowns track quite well. As shown in Tables 4.2 and 4.3, the relative power breakdown numbers for our models are within 10-13% on average of reported data.

Hardware Structure	Intel Data	Model
Instruction Fetch	22.2%	21.0%
Register Alias Table	6.3%	4.9%
Reservation Stations	7.9%	8.9%
Reorder Buffer	11.1%	11.9%
Integer Exec. Unit	14.3%	14.6%
Data Cache Unit	11.1%	11.5%
Memory Order Buffer	6.3%	4.7%
Floating Point Exec. Unit	7.9%	8.0%
Global Clock	7.9%	10.5%
Branch Target Buffer	4.7%	3.8%

Table 4.2: Comparison between Modeled and Reported Power Breakdowns for the Pentium Pro®.

Hardware Structure	Alpha 21264	Model
Caches	16.1%	15.3%
Out-of-Order Issue Logic	19.3%	20.6%
Memory Management Unit	8.6%	11.7%
Floating Point Exec. Unit	10.8%	11.0%
Integer Exec. Unit	10.8%	11.0%
Total Clock Power	34.4%	30.4%

Table 4.3: Comparison between Modeled and Reported Power Breakdowns for the Alpha 21264.

Tables 4.2 and 4.3 compare breakdowns of Wattch's power consumption for different hardware structures, with those from published data for the Intel Pentium Pro® and (then Compaq) Alpha 21264 CPUs [41; 59]. This power consumption is

shown for "maximum power" operation, when all of the units are fully active. This mode of operation represents our worst-case power estimates; we assume all of the ports on all of the units are fully active, with maximum switching activity. We did not actually modify SimpleScalar's internal structure to resemble these processors. Instead we used the worst-case power estimates from our models for the hardware configurations of the processors. The parameter configurations for our models are set based on published Intel and Alpha 21264 parameters [41; 43].

The power breakdowns track fairly well. For example, the Intel data in Table 4.2 is an exact or near match for several units. These include the data caches, instruction fetch and out-of-order control logic. The average difference between the power consumption of our modeled structures and the reported data was 13.3% for the Intel processor.

The relative power proportions for the Alpha 21264 are again similar to the reported data, with an average difference of 10.7%.

One unit which shows some inaccuracy in our current model is the global clock power for the Intel processor; our model predicts it to be 10% of total chip power, while the published data suggests it is less: 8%. This difference could be because the clock power model we use is based on an aggressive H-tree style that was used in Alpha 21264 [36], but not in the Intel processor.

The Alpha 21264 has a significantly higher percentage of total clock power than Intel: 34% for the Alpha compared to 8% for the Intel processor. The main reason for this large difference is simply the method of accounting that is used for clock power by the two groups. The clock power category for 21264 includes all clock capacitance including the clock nodes within individual units. On the other hand, the Intel method for clock power accounting only counts clock power as the global clock network. Clock nodes that are internal to various hardware structures are counted towards the power dissipation of those units. When we model these two

different chips, we adjust our clock power accounting method to match that of the respective company's data reporting.

Finally, note that the power proportions we discuss here are normalized to the hardware structures that we consider. For example, since we do not model Intel's complex x86 to micro-op decoding, we do not report the instruction decode unit power consumption, which consumes 14% of the chip power.

4.2.3 Validation 3: Max power consumption for three CPUs

In this section we perform a third form of validation in which we compare the published maximum power numbers for three commercial microprocessors with the values produced by our models for similar configurations. This allows us to evaluate both the relative and absolute accuracy of our power models. While such a comparison is difficult without exact process parameter information, general power trends can be seen based on the hardware organizations of these machines. Table 4.4 describes the details of the three processors that we consider.

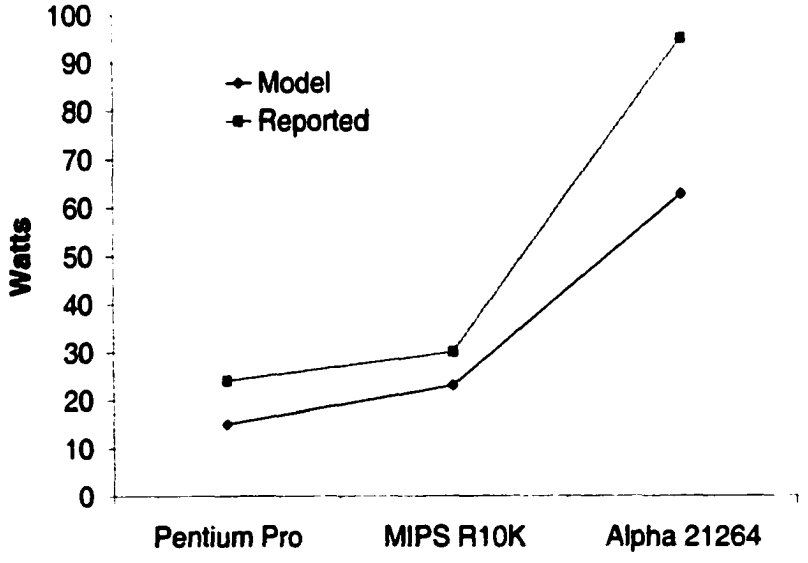

Figure 4.2: Maximum power numbers for three processors: Model and Reported.

Processor	Alpha 21264	Pentium Pro	MIPS R10000
Processor Core			
Instr. Window(s)	20 INT 15 FP	20 UOPs	16 INT 16 MEM 16 FP
Physical Registers	2x80-INT 72-FP	40 UOPs	64-INT 64-FP
Memory Order Queue	32	20	8
Fetch width per cycle	4	3	4
Decode width per cycle	4	6	4
Issue width per cycle	6	3	4
Commit width per cycle	4	3	4
Functional Units	4 Int 2 FP	4 Int 1 FP	3 Int 3 FP
Branch Prediction			
Local History Table	1024x10	N/A	N/A
Local Predict	1024x3	512x4	512x2
Global History Register	12	N/A	N/A
Global Predict	4096x2	N/A	N/A
Choice Predict	4096x2	N/A	N/A
BTB	1K entry 2-way	512 entry 4-way	32 entry
Return-address stack	32 entry	N/A	N/A
Memory Hierarchy			
L1 Dcache Size	64K	8K	32K
L1 Dcache Assoc.	2-way	2-way	2-way
L1 Icache Size	64K	8K	32K
L1 Icache Assoc.	2-way	4-way	2-way
DTLB Size (full assoc)	128	64	64
ITLB Size (full assoc)	128	32	64
Process Specifications			
Feature Size	.35um	.35um	.35um
Vdd	2.2V	3.3V	3.3V
MHz	600	200	200

Table 4.4: Configuration of Processors

Figure 4.2 shows the results for the maximum power dissipation for the three processors that we considered. In all three cases, Wattch's modeled power consumption was less than the reported power consumptions, on average 30% lower. There are a few reasons for this systematic underestimation. First, we have concentrated on the units that are most immediately important for architects to consider, neglecting I/O circuitry, fuse and test circuits, and other miscellaneous logic. Second, circuit implementation techniques, transistor sizings, and process parameters will vary from company to company. On the other hand, the models are general enough that they could be tuned to a particular processor's implementation details. It is reassuring to see that the trends already track published data for several high-end commercial processors.

4.3 Robustness of Relative Accuracy

In the remainder of this chapter we will provide quantitative results for both Wattch and PowerTimer to demonstrate how design tradeoff studies can be performed despite the presence of different types of error in the low-level power models. These results give some insight into the robustness of the relative accuracy of the power models and demonstrate the extent to which a design tradeoff study can withstand error in the low level power models.

4.3.1 Design Criteria

When performing a design tradeoff study, a methodology must first be established for deciding when to choose a particular design point over another design point. When viewing design tradeoff curves visually, we would like to choose the "knee" of the curve so as to pick the point that is close to optimal without reaching the point of diminishing returns. To quantify this tradeoff selection, we propose the

acceptable range window as a method to quantify the selection of design points from raw power/performance data.

Acceptable Range Window

The experiments in this chapter quantify the amount of acceptable error within a power/performance simulator tolerated before different design points are chosen. The acceptable range window forms a group of points which meet the criteria for selection. Generally, we choose the lowest cost point within the acceptable range window for implementation.

Two different definitions of the acceptable range window are considered:

- +/-x% of absolute at optimal choice (*range1*)
- +/-y% of (worst_choice - optimal_choice) for this design study (*range2*)

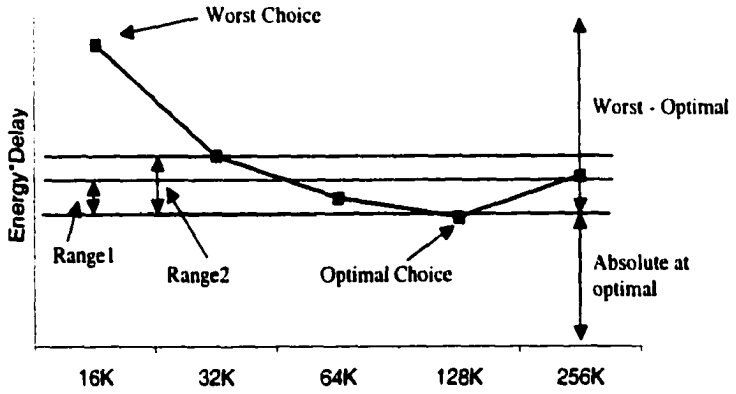

Figure 4.3: Example of acceptable range windows.

The derivation of the acceptable range window is shown graphically in Figure 4.3. In this figure, the absolute at the optimal choice and the range of the worst_choice - optimal_choice are shown. The *range1* and *range2* windows are also shown.

For each design tradeoff experiment, two checks are performed. First, we check for overlap between the acceptable range windows of the baseline simulator and the

modified simulator. Second, we check if the window still suggests the same lowest cost point for the modified simulator.

In Figure 4.3, the 64K and 128K design choices are optimal for both criteria, but the 256K design choice is only optimal under the *range2* criteria. However, both criteria choose the same least cost design point at 64K.

4.3.2 Wattch

We use Wattch as the baseline simulator to perform the first set of studies. Four distinct types of error are considered that could affect the power models. While all of these types of error disturb the absolute accuracy of the simulator, this study quantifies the effect on the relative accuracy of the simulator by investigating several design tradeoff ppstudy scenarios.

The four types of error considered are as follows:

- Error within a unit that is independent of the design tradeoff experiment.
- Error within a power model that is used in the structure under study as well as in independent structures.
- Error solely within the unit under study.
- Error in the amount/type of clock gating style used in the simulator.

Three typical design tradeoff studies were considered for each of these four error conditions. These design tradeoff studies investigate energy-delay product for the number of RUU-entries, the size of the L1 Data Cache, and the size of the L1 Instruction Cache.

The results are shown for both the *range1* and *range2* windows with x=y=5%.

Example 1: Error in an Independent Unit

We will first consider a simple experiment which has error in the power estimate for a unit that is totally independent of the unit under investigation in the design study. For example, error in the ALU power model or the global clock power, is mostly independent of the power model for the RUU or the L1 caches. While the absolute accuracy of the model suffer quite a bit under these conditions, the relative accuracy of the model for this particular design study will be less severely affected.

Figures 4.4, 4.5, and 4.6 show two graphs each for the *vortex* application while varying the number of RUU entries, D-Cache size, and I-Cache size. In each of the graphs there are five curves showing the power and energy-delay product trends while varying the microarchitectural parameters. The five lines labeled -.2x through .2x refer to the amount of additional power dissipation inserted into the model. The amount of power added or subtracted is equal to the ratio given multiplied by the total chip power of the baseline case with an 80-entry RUU, and 64KB D- and I-Caches. For example, if the baseline power dissipation estimate was 30W, the .2x point adds 6W additional power and the -.2x point subtracts 6W of power from the total chip power.

The first graph in each figure shows the power dissipation while varying both conditions. Since the additional power dissipation added in this experiment is independent of the RUU or cache power models, it does not affect the relative accuracy of this curve and only shifts the curves up and down by the corresponding amounts.

The second graph in each figure shows the energy-delay product while varying the microarchitectural parameters and the amount of error. The energy-delay product factors in the IPC, performance, for the various microarchitectural choices. Because of this, the energy-delay product curves are skewed by the IPCs of the various design points.

Although the relative accuracy of the power dissipation curves is not disturbed,

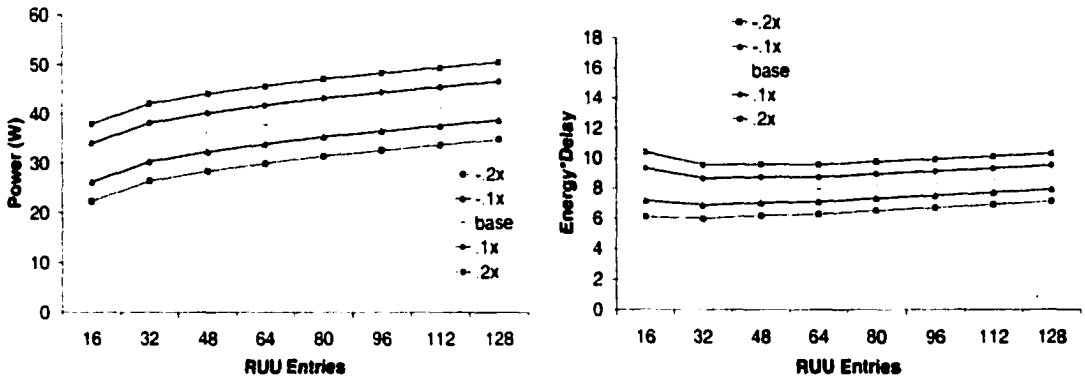

Figure 4.4: Power and EDP for *vortex* varying indep. unit and RUU entries.

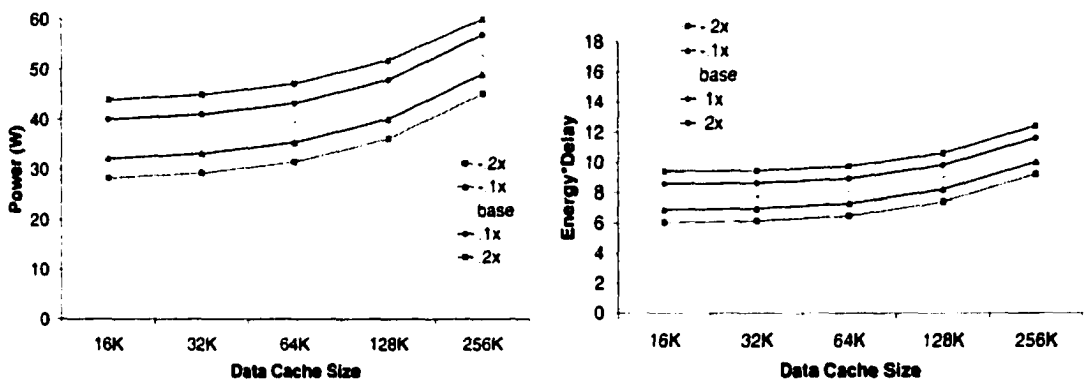

Figure 4.5: Power and EDP for *vortex* varying indep. unit and D-Cache size.

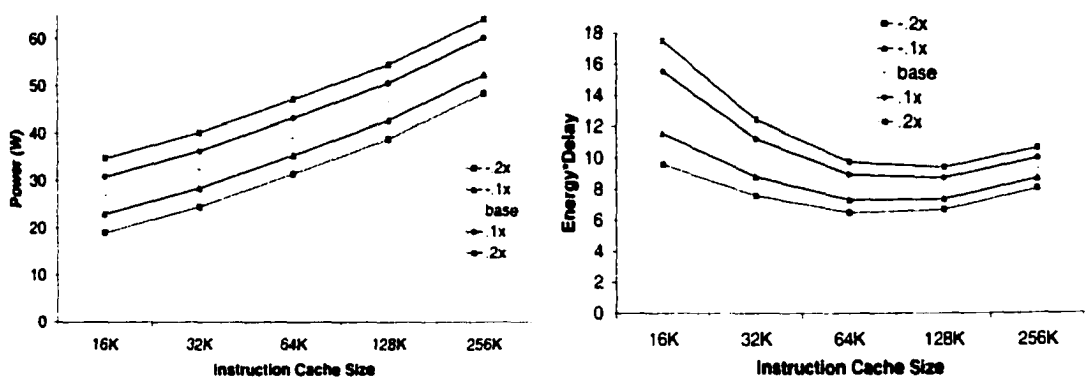

Figure 4.6: Power and EDP for *vortex* varying indep. unit and I-Cache size.

Acceptable range windows under independent scaling error ratios
Range1 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r1	16-32	16-32	16-32 entries	**32-48**	**32-48**
gcc-r1	**16**	16-32	16-32 entries	16-32	16-32
go-r1	16	16	16 entries	16	16-**32**
ijpeg-r1	**16-48**	32-**48**	32-80 entries	32-**112**	32-**128**
m88ksim-r1	16	16	16 entries	16-**32**	16-**32**
vortex-r1	**16-64**	**16-64**	32-80 entries	32-80	32-**96**

(a) RUU-entries

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r1	32K-128K	32K-128K	32K-128K	32K-128K	32K-128K
gcc-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
go-r1	16K-64K	16K-64K	16K-64K	16K-64K	**32K**-64K
ijpeg-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
m88ksim-r1	16K-32K	16K-32K	16K-32K	16K-**64K**	16K-**64K**
vortex-r1	16K-32K	16K-32K	16K-32K	16K-**64K**	16K-**64K**

(b) L1 Data Cache

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r1	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
gcc-r1	64K	64K	64K	64K-**128K**	64K-**128K**
go-r1	32K-64K	32K-64K	32K-64K	32K-64K	32K-64K
ijpeg-r1	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
m88ksim-r1	32K-64K	32K-64K	32K-64K	32K-64K	32K-64K
vortex-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.5: Optimal design choice decisions under indepent scaling ratios

Acceptable range windows under independent scaling error ratios
Range2 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r2	**16**-32	32	32 entries	32	32
gcc-r2	16	16	16 entries	16-**32**	16-**32**
go-r2	16	16	16 entries	16	16
ijpeg-r2	32	32	32 entries	32	32
m88ksim-r2	16	16	16 entries	16	16
vortex-r2	32	32	32 entries	32	32,**64**

(a) RUU-entries

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r2	64K	64K	64K	64K	64K
gcc-r2	**16K**-32K	32K	32K	32K	32K-**64K**
go-r2	32K-64K	32K-64K	32K-64K	32K-64K	32K-64K
ijpeg-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
m88ksim-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
vortex-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	-.2x	-.1x	No Error (0x)	.1x	.2x
compress-r2	16K	16K	16K	16K	16K
gcc-r2	64K	64K	64K	64K	64K-**128K**
go-r2	**32K**	**32K**	32K-64K	32K-64K	32K-64K
ijpeg-r2	16K	16K	16K	16K	16K
m88ksim-r2	32K	32K	32K	32K	32K-**64K**
vortex-r2	**64K**	64K-128K	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.6: Optimal design choice decisions under indepent scaling ratios

the variation in IPC among design points, combined with the additional independent unit power dissipation, can lead to differences in the energy-delay product design tradeoff choices. Tables 4.5 and 4.6 show the acceptable range windows for optimal EDP designs for the RUU, L1 D-Cache, and L1 I-Cache under the independent error scaling ratios described above with the *range1* and *range2* design criteria.

Each table shows the acceptable range windows for the baseline case with no error (0x scaling) and the cases for -.2x through .2x. Bold entries in the table indicate acceptable range windows with a deviation from the baseline case. If a deviation occurs on the left-side of the acceptable range window, a different lowest-cost design point would be chosen. For example in Table 4.5, the lowest cost RUU size for *ijpeg* within the acceptable range windows is 32 entries with the baseline case. However, at -.2x scaling the design choice would be 16-entries. In general, *range2* has fewer deviations in design choices, because of the tighter design criteria.

Figures 4.7, 4.8, and 4.9 show the percent difference from the optimal design choice for the various microarchitectural choices and the amount of error inserted. The y-axis of these figures plot the percent difference between the design point and the optimal (lowest energy-delay) design point. The x-axis shows the amount of additional independent unit error that is inserted, and the lines plotted show the different design points under consideration. For example, in Figure 4.7 the 32-entry curve is always equal to zero, because this is the optimal energy-delay design point for this experiment under all of the error conditions.

These figures show that for microarchitectural design points that are smaller than optimal, the difference from the optimal point increases with additional independent unit error. As the microarchitectural design points become larger than the optimal design choice, the difference from optimal decreases with additional independent unit error. This occurs because as the total power dissipation becomes larger (when the

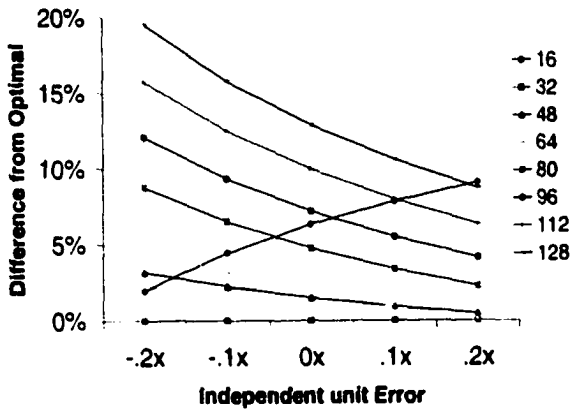

Figure 4.7: Design optimality for *vortex* varying independent error and RUU entries.

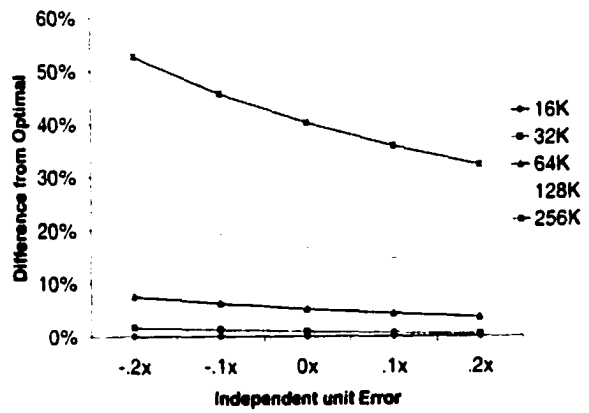

Figure 4.8: Design optimality for *vortex* varying independent error and D-Cache size.

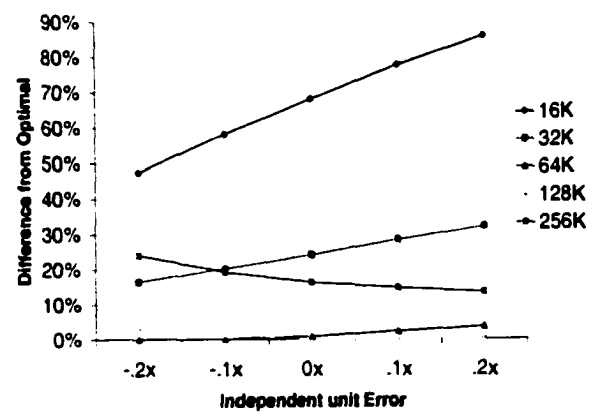

Figure 4.9: Design optimality for *vortex* varying independent error and I-Cache size.

error increases) the additional performance benefit achieved by increasing the microarchitectural structure is magnified. Thus, the smaller design points are farther from optimal as the error increases, and the larger design points will become closer to optimal as the error increases.

In these curves, any point that is less than 5% corresponds to an entry in the acceptable range window with the *range1* criteria. Discrepancies in the acceptable range window occur when a curve has some points above and some points below the 5% threshold. Curves with a steeper slope are more affected by the independent unit error, because they are more likely to have some points that fall above and below the threshold.

Example 2: Error in Bitline Capacitance

A second major class of experimental inaccuracy in power models is error that occurs in a model that is used within many microarchitectural structures. For example, the cache power models is used in the L1 instruction and data caches, the L2 cache, and the branch predictor tables. Error in the cache power model would affect the power estimates for many of these units.

In this example, we consider bitline capacitance, a component that will have an effect on many microarchitectural structures in our processor model. Bitline capacitance estimates are used within the array structure models for caches and register files. The errors in bitline capacitance affect all three of the microarchitectural parameters under study, as well as several independent structures.

Figures 4.10, 4.11, and 4.12 show the power and energy-delay product for the *vortex* application while varying the number of RUU entries, D-Cache size, and I-Cache size. The five curves shown are similar to the ones in the previous section, but each of these curves shows a different ratio for the bitline capacitance scaling that was used. Again, significant deviations are difficult to see from these curves even with

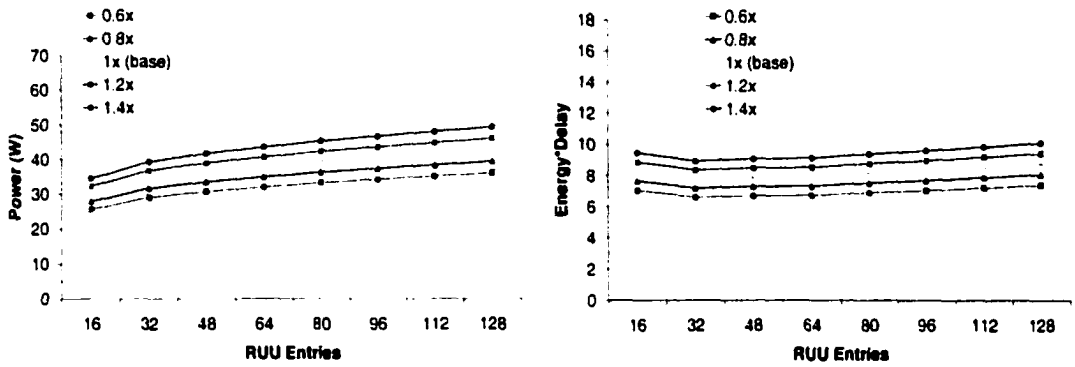

Figure 4.10: Power and EDP for *vortex* varying bitline error and RUU entries.

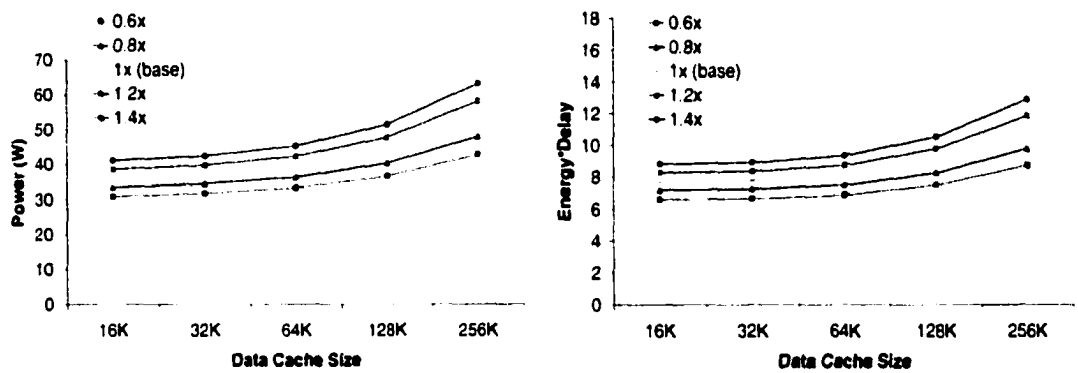

Figure 4.11: Power and EDP for *vortex* varying bitline error and D-Cache size.

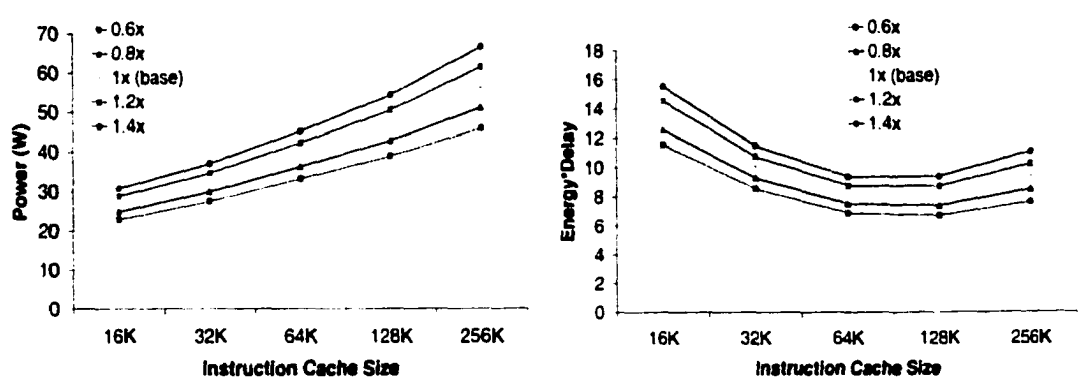

Figure 4.12: Power and EDP for *vortex* varying bitline error and I-Cache size.

0.6x and 1.4x scaling of the bitline capacitance estimates.

Tables 4.7 and 4.8 show the acceptable range windows for the design choices for the RUU, L1-DCache and L1-ICache under the same bitline scaling. These tables show the benchmark with the *range1* (r1) and *range2* (r2) acceptable range window calculations. For many of the benchmarks with these three design tradeoff studies, the error in bitline capacitance had no effect. However, in a few cases the acceptable range window did change. In only one case, *compress*'s L1-Dcache design choice under *range1*, did the least-cost design choice change. In this case, with 0.6x scaling, a 32K cache is chosen, while with the 1x scaling a 16K cache is chosen as the least-cost design choice.

Figures 4.13, 4.14, and 4.15 show the difference from design optimality for the design tradeoff selections. Again, any point on these curves that is less than 5% corresponds to an entry in the acceptable range window with the *range1* criteria. The relatively small slopes of most of the curves demonstrates that the bitline error will have a small effect on the design tradeoff choices.

Example 3: Error in Dependent Unit Scaling Factors

We now consider the effect of error solely associated with the unit under consideration in the design study. These experiments explore this source of error by explicitly scaling the power estimate for the individual structures (RUU and L1 Caches) by 1x through 2x. This type of error could exist if the wrong subbanking scheme was assumed, if a different circuit-design style was chosen for that particular structure, etc.

Figures 4.16, 4.17, and 4.18 show the power and energy-delay product for the *vortex* application while varying the number of RUU entries, D-Cache size, and I-Cache size. Each of the five curves again shows the energy-delay product as that particular unit's power estimate scales by 1x through 2x. This type of error clearly affects the design tradeoff study. As the amount of scaling increases, instead of just

Acceptable range windows under bitline error ratios
Range1 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r1	16-48	16-48	16-48 entries	16-48	16-48
gcc-r1	16-64	16-64	16-64 entries	16-**48**	16-**48**
go-r1	16-**64**	16-**64**	16-48 entries	16-48	16-48
ijpeg-r1	32-80	32-80	32-80 entries	32-80	32-**64**
m88ksim-r1	16-64	16-64	16-64 entries	16-64	16-64
vortex-r1	32-80	32-80	32-80 entries	32-80	32-80

(a) RUU-entries

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r1	**32K**-128K	16K-128K	16K-128K	16K-128K	16K-128K
gcc-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
go-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
ijpeg-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
m88ksim-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-**32K**
vortex-r1	16K-**64K**	16K-**64K**	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r1	64K-**128K**	64K-**128K**	64K	64K	64K
gcc-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K
go-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K
ijpeg-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K
m88ksim-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K
vortex-r1	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.7: Optimal design choice decisions under bitline error ratios

Acceptable range windows under bitline error ratios
Range2 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r2	32	32	32 entries	32	32
gcc-r2	32	32	32 entries	32	32
go-r2	16-32	16-32	16-32 entries	16-32	16-32
ijpeg-r2	32	32	32 entries	32	32
m88ksim-r2	32	32	32 entries	32	32
vortex-r1	32	32	32 entries	32	32

(a) RUU-entries

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r2	64K	64K	64K	64K	64K
gcc-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
go-r2	32K-64K	32K-64K	32K-64K	**32K**	**32K**
ijpeg-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
m88ksim-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
vortex-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	.6x	.8x	No Error (1x)	1.2x	1.4x
compress-r2	64K	64K	64K	64K	64K
gcc-r2	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K
go-r2	64K-128K	64K-128K	64K	64K	64K
ijpeg-r2	64K-128K	64K-128K	64K	64K	64K
m88ksim-r2	64K-128K	64K-128K	64K-128K	64K-128K	**64K**
vortex-r2	64K-128K	64K-128K	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.8: Optimal design choice decisions under bitline error ratios

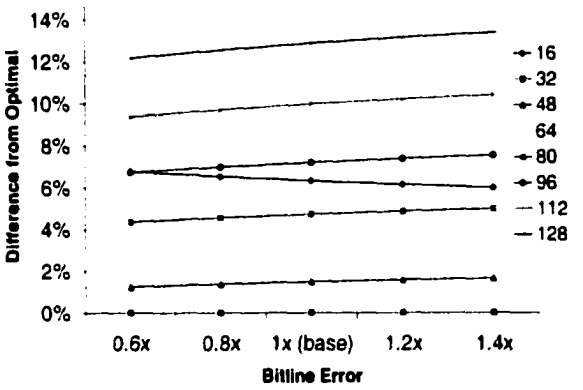

Figure 4.13: Design optimality for *vortex* varying bitline error and RUU entries.

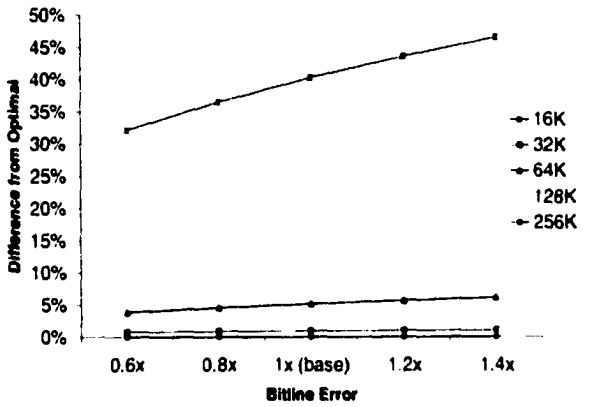

Figure 4.14: Design optimality for *vortex* varying bitline error and D-Cache size.

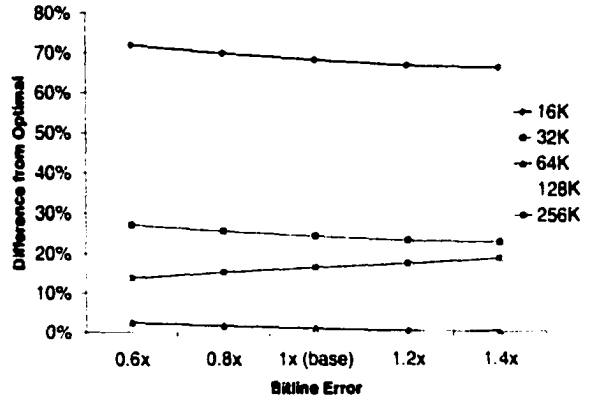

Figure 4.15: Design optimality for *vortex* varying bitline error and I-Cache size.

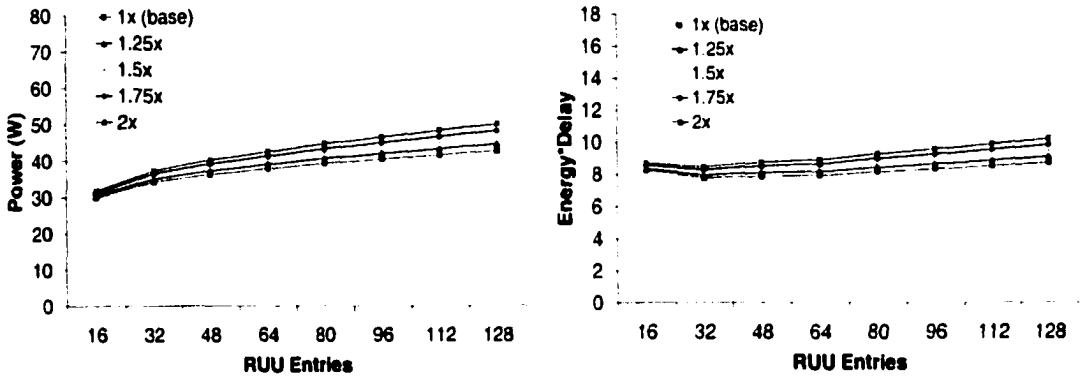

Figure 4.16: Power and EDP for *vortex* varying RUU-scale-factor and RUU entries.

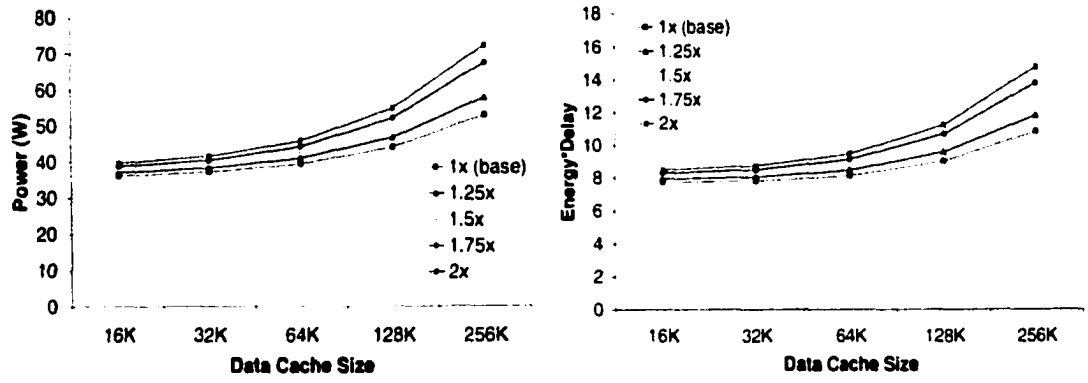

Figure 4.17: Power and EDP for *vortex* varying DCache-sf and D-cache size.

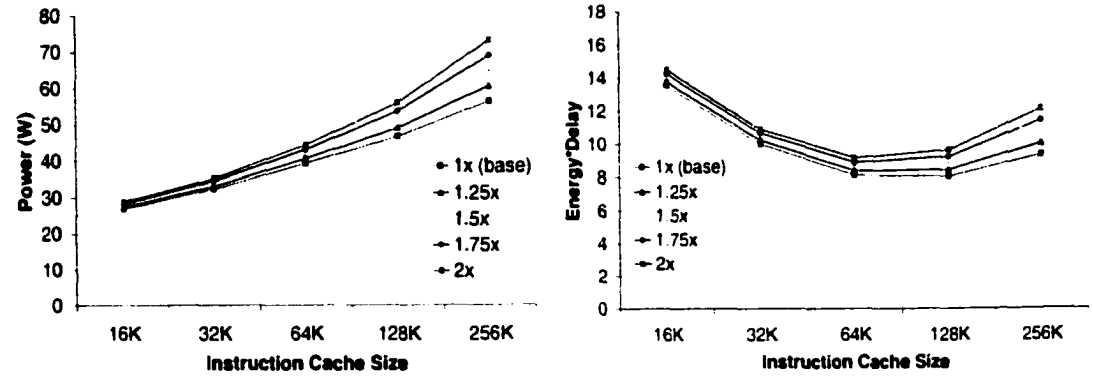

Figure 4.18: Power and EDP for *vortex* varying ICache-sf and I-cache size.

shifting the results, the curves begin to separate as more scaling is applied.

The acceptable range windows quantify the disturbance to the design tradeoff study. With a 1.25x scaling factor, there is very little change in the acceptable range windows for the three design tradeoff studies with these applications. Only *compress* with the *range2* window resulted in a different design decision; with the baseline simulator a 64K I-Cache is chosen and with the 1.25x scaling, a 32K I-Cache is chosen.

Moving to the 1.5x and larger scaling factors, more design decisions change. *Vortex* under the *range1* and *compress* under the *range2* window need a smaller number of RUU-entries.

Errors that only involves the unit under study in a design tradeoff experiment will be more likely to cause a different design choice to be made. This occurs because the additional scaling on the microarchitectural structure, in the absence of the scaling in other independent units, causes the structure in the tradeoff experiment to become a larger share of the overall chip pie.

Figures 4.19, 4.20, and 4.21 show the difference from design optimality for the various scaling ratios. The curves with steep slopes that are near the 5% range correspond to the entries highlighted in the previous tables. For example, in Figure 4.19 the 16-entry curve falls below 5% only under the 1.5x - 2x scaling factor conditions.

Clock Gating

This section considers the effect of Wattch's three base clock gating modes on the design tradeoff study. The first mode is simple clock gating, where a unit consumes 100% power if any port is active on a cycle, and can only be gated off if no ports are in use. The next mode is ideal clock gating where power is linearly proportional to the number of ports in use. The last mode is aggressive clock gating which is similar to ideal, but disabled ports consume some additional power.

Acceptable range windows under scaling ratios
Range1 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r1	16-32 entries	16-32	16-32	16-32	16-32
gcc-r1	16-32 entries	16-32	16-32	**16**	**16**
go-r1	16 entries	16	16	16	16
ijpeg-r1	32-80 entries	32-**64**	32-**64**	32-**48**	32-**48**
m88ksim-r1	16 entries	16	16	16	16
vortex-r1	32-80 entries	32-**64**	**16-64**	**16-64**	**16-48**

(a) RUU-entries

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r1	32K-128K	32K-128K	**16K-64K**	**16K-64K**	**16K-64K**
gcc-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-**32K**
go-r1	16K-64K	16K-64K	16K-64K	16K-64K	16K-64K
ijpeg-r1	16K-64K	16K-64K	16K-**32K**	16K-**32K**	16K-**32K**
m88ksim-r1	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
vortex-r1	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r1	16K-32K	16K-32K	16K-32K	16K-32K	**16K**
gcc-r1	64K	64K	64K	64K	64K
go-r1	32K-64K	32K-64K	32K-64K	32K-64K	**32K**
ijpeg-r1	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
m88ksim-r1	32K-64K	32K-64K	32K-64K	**32K**	**32K**
vortex-r1	64K-128K	64K-128K	64K-128K	64K-128K	**64K**

(c) L1 Instruction Cache

Table 4.9: Optimal design choice decisions under scaling ratios

Acceptable range windows under scaling ratios
Range2 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r2	32 entries	32	32	**16**-32	**16**-32
gcc-r2	16 entries	16	16	16	16
go-r2	16 entries	16	16	16	16
ijpeg-r2	32 entries	32	32	32	32
m88ksim-r2	16 entries	16	16	16	16
vortex-r2	32 entries	32	32	32	32

(a) RUU-entries

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r2	64K	**32K**-64K	**32K**-64K	**32K**-64K	**32K**-64K
gcc-r2	16K-64K	16K-64K	16K-64K	16K-64K	16K-**32K**
go-r2	32K-**64K**	32K-**64K**	32K	32K	32K
ijpeg-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
m88ksim-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K
vortex-r2	16K-32K	16K-32K	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	No Error (1x)	1.25x	1.5x	1.75x	2x
compress-r2	16K	16K	16K	16K	16K
gcc-r2	64K	64K	64K	64K	64K
go-r2	32K-64K	**32K**	**32K**	**32K**	**32K**
ijpeg-r2	16K	16K	16K	16K	16K
m88ksim-r2	32K	32K	32K	32K	32K
vortex-r2	64K-128K	64K-128K	64K-128K	**64K**	**64K**

(c) L1 Instruction Cache

Table 4.10: Optimal design choice decisions under scaling ratios

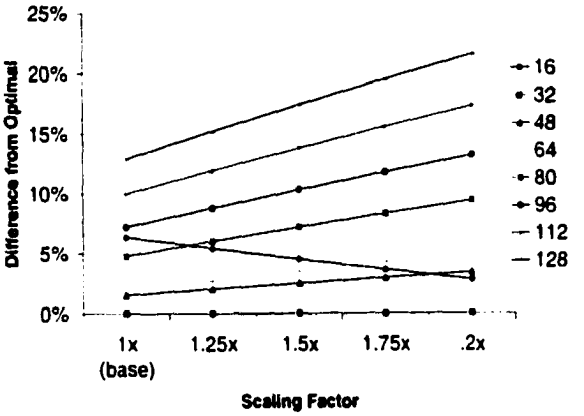

Figure 4.19: Design optimality for *vortex* varying RUU-scale factor and RUU entries.

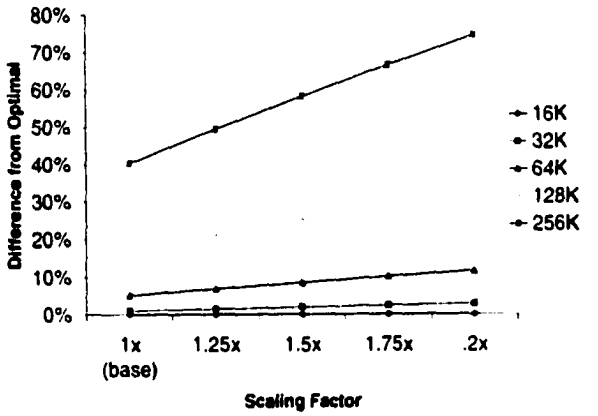

Figure 4.20: Design optimality for *vortex* varying D-Cache scaling and D-Cache size.

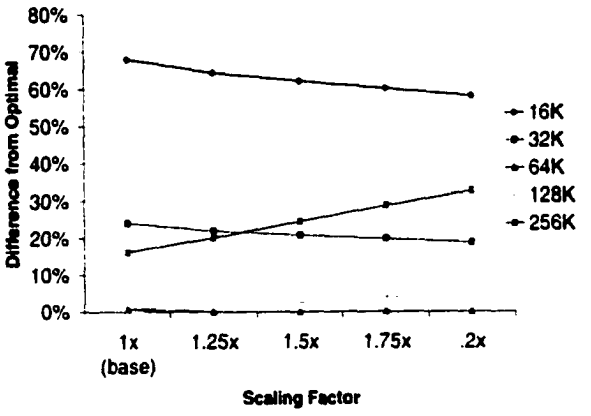

Figure 4.21: Design optimality for *vortex* varying I-Cache scaling and I-Cache size.

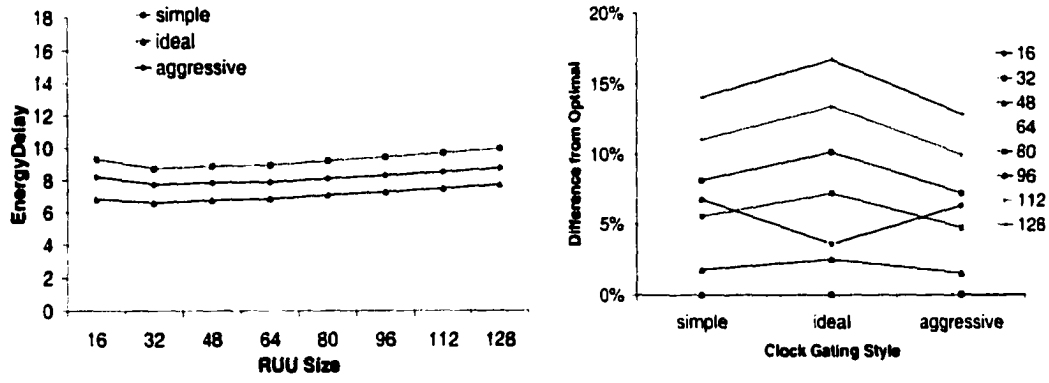

Figure 4.22: EDP and optimality for *vortex* varying clock gating and RUU entries.

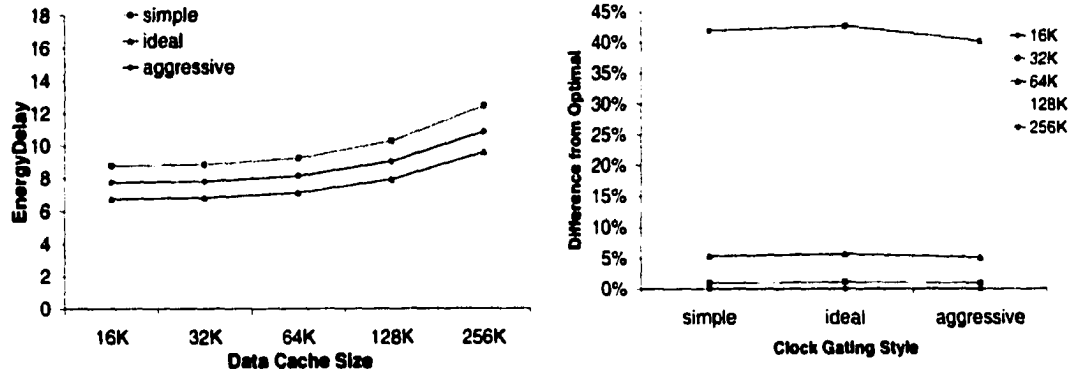

Figure 4.23: EDP and optimality for *vortex* varying clock gating and D-Cache size.

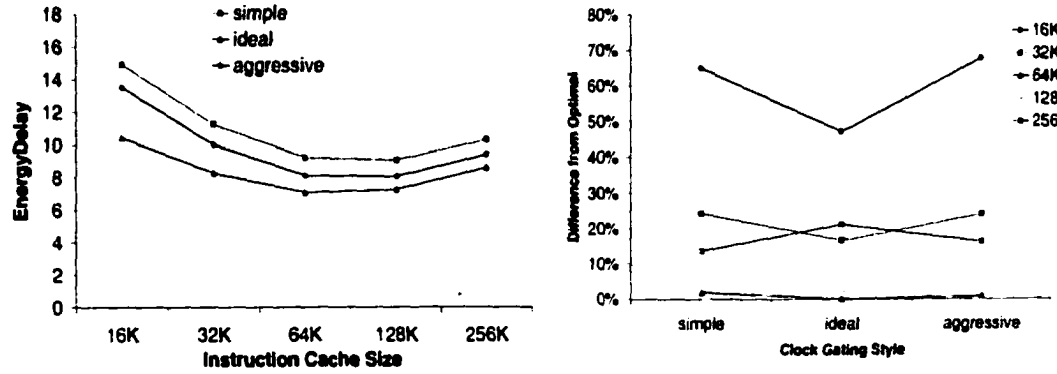

Figure 4.24: EDP and optimality for *vortex* varying clock gating and I-Cache size.

Figures 4.22, 4.23, and 4.24 show the energy-delay and the optimality analysis for *vortex* while varying the clock gating style and the RUU, D-Cache, and I-Cache sizes. There are some changes in the optimal design points under different clock gating conditions. These changes are particularly noticeable with the I-Cache experiment at the 16K size. *Vortex* incurs a large number of I-Cache misses with small I-Caches, resulting in many opportunities for it to be clock gated in the simulator. This causes a noticeable difference with the three clock gating strategies. The RUU figure also has noticeable changes particularly at the 16-entry and 32-entry design choices.

Tables 4.11 and 4.12 show the acceptable range windows for the design choices for the RUU, L1-DCache and L1-ICache while varying the clock gating strategy. These tables show that the choice of clock gating style can have an effect on the design choices for these microarchitectural structures. The lowest-cost RUU sizing within the acceptable range windows varies considerably depending on the simple, ideal, or aggressive clock gating settings. For the I-Cache and D-Cache tables, there are fewer differences primarily because since there are fewer ports on these structures, the different clock gating strategies have smaller effects.

4.3.3 PowerTimer

In the next study, we use PowerTimer to perform similar experiments. One of the potential sources of inaccuracy in PowerTimer is the scaling ratio factors used as the size of the microarchitectural structures increases or decreases. Chapter 2 discusses these scaling factors. For most structures, power increases proportionally to the number of entries in a structure. For example, if the number of entries in an issue queue doubles, the power consumption doubles. For cache structures, the power increases by 1.46x for every doubling of the cache size as suggested by the circuit experiments in [53].

Acceptable range windows under clock gating schemes
Range1 Criteria

Benchmark	simple	ideal	aggressive
compress-r1	16-32 entries	16-32	16-48
gcc-r1	16-48 entries	16-48	16-64
go-r1	16-48 entries	16-32	16-48
ijpeg-r1	32-64 entries	16-64	32-80
m88ksim-r1	16-64 entries	16-48	16-64
vortex-r1	32-64 entries	16-64	32-80

(a) RUU-entries

Benchmark	simple	ideal	aggressive
compress-r1	32K-128K	32K-256K	16K-128K
gcc-r1	16K-64K	16K-64K	16K-64K
go-r1	16K-64K	16K-64K	16K-64K
ijpeg-r1	16K-64K	16K-64K	16K-64K
m88ksim-r1	16K-32K	16K-64K	16K-64K
vortex-r1	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	simple	ideal	aggressive
compress-r1	64K-128K	64K	64K
gcc-r1	64K-128K	64K-128K	64K-128K
go-r1	64K-128K	64K	64K-128K
ijpeg-r1	64K-128K	64K-128K	64K-128K
m88ksim-r1	64K-128K	64K-128K	64K-128K
vortex-r1	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.11: Optimal design choice decisions under different clock gating schemes

Acceptable range windows under clock gating schemes
Range2 Criteria

Benchmark	simple	ideal	aggressive
compress-r2	32 entries	16	32
gcc-r2	32 entries	16-32	32
go-r2	32 entries	16	16-32
ijpeg-r2	32 entries	32	32
m88ksim-r2	32 entries	16-32	32
vortex-r2	32 entries	32	32

(a) RUU-entries

Benchmark	simple	ideal	aggressive
compress-r2	64K	64K-128K	64K
gcc-r2	16K-32K	16K-32K	16K-32K
go-r2	32K-64K	32K-64K	32K-64K
ijpeg-r2	16K-32K	16K-32K	16K-32K
m88ksim-r2	16K-32K	16K-32K	16K-32K
vortex-r2	16K-32K	16K-32K	16K-32K

(b) L1 Data Cache

Benchmark	simple	ideal	aggressive
compress-r2	64K	64K	64K
gcc-r2	64K-128K	64K	64K-128K
go-r2	64K-128K	64K	64K
ijpeg-r2	64K-128K	64K	64K
m88ksim-r2	64K-128K	64K	64K-128K
vortex-r2	64K-128K	64K-128K	64K-128K

(c) L1 Instruction Cache

Table 4.12: Optimal design choice decisions under different clock gating schemes

Scaling Factors

Figures 4.25, 4.26, and 4.27 show the energy-delay for the aggregate of *SPECint95* while varying the scaling ratios and the core size, L1 D-Cache size, and the size of the global completion table (GCT). Figures 4.26 and 4.27 show the scaling factors of 1.2x-2.8x (with a baseline of 2x), and Figure 4.25 shows scaling factors from 0.876x through 2.044x with a baseline of 1.46x.

Figure 4.27 shows results as we vary the size of the core by scaling all of the issue queues, renamers, and other major microarchitectural structures in the core while leaving the caches constant. Core3 is the baseline core; core4 scales the size of every structure by 1.2x and core5 by 1.4x. Similarly core2 and core1 reduce the size of the core by scaling by 0.8x and 0.6x.

Tables 4.13 and 4.14 show the acceptable range windows for the 8 individual SPECint95 applications. For the L1 D-Cache experiment, although the acceptable range windows do change in some cases, all of the design choices remain the same with 1.168x through 2.044x scaling ratios. At 0.876x scaling, *go* chooses a larger cache size under both range definitions.

The completion buffer experiment demonstrates several cases where the design tradeoff choice changes. At 1.6x and 2.4x scaling, *go*, *perl*, and *vortex* choose different numbers of GCT entries under the *range2* criteria. At the 2.8x scaling there are even more differences. For example, *go* would choose a 0.6x size GCT with 2x scaling, but would choose a 1x size GCT with 2.8x scaling. Under the *range1* criteria, the design tradeoff choices remain the same. This is because the GCT is a relatively small structure in the overall chip's power dissipation so the differences in this design tradeoff only show up with the *range2* criteria.

The core size experiment changes the size of many structures, so it is likely that this experiment will be especially susceptible to varying scaling ratios. With the 1.6x and 2.4x scaling, *compress*, *go*, *m88ksim*, and *perl* had different acceptable range

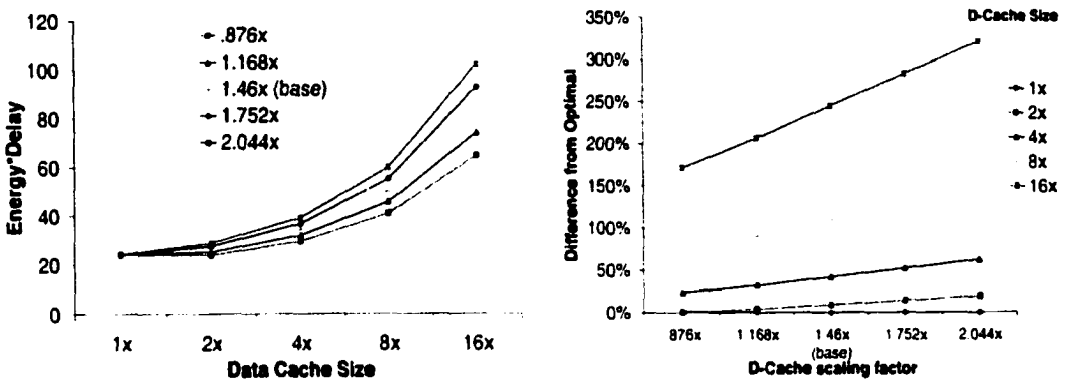

Figure 4.25: EDP and optimality for *SPECint95* varying DCache-SF and DCache size.

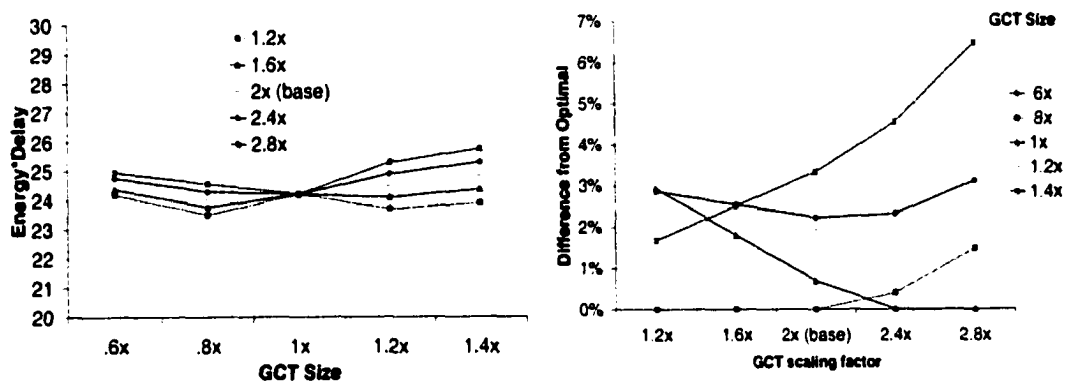

Figure 4.26: EDP and optimality for *SPECint95* varying GCT-SF and GCT size.

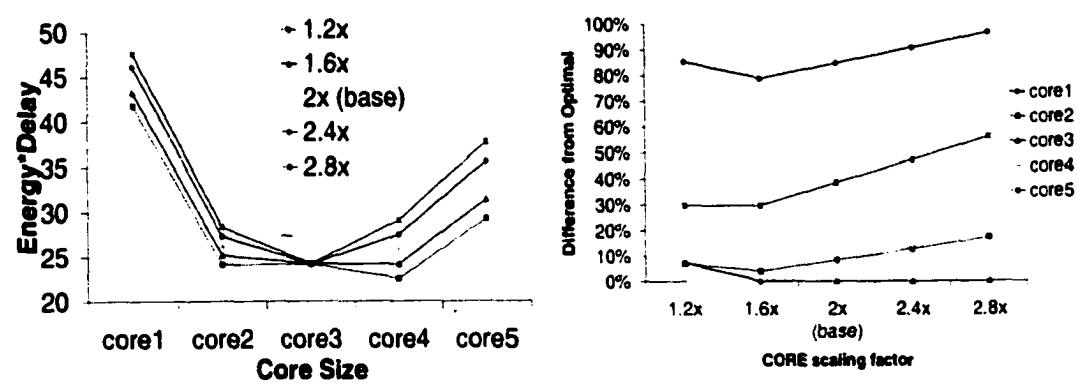

Figure 4.27: EDP and optimality for *SPECint95* varying CORE-SF and CORE size.

Acceptable range windows under scaling ratios
Range1 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	.876x	1.168x	Baseline (1.46x)	1.752x	2.044x
compress-r1	1x-**2x**	1x-**2x**	1x	1x	1x
gcc-r1	1x-**2x**	1x	1x	1x	1x
go-r1	**2x**	1x-2x	1x-2x	**1x**	**1x**
ijpeg-r1	1x-**2x**	1x	1x	1x	1x
li-r1	1x-**2x**	1x	1x	1x	1x
m88ksim-r1	1x-**2x**	1x	1x	1x	1x
perl-r1	1x-**2x**	1x	1x	1x	1x
vortex-r1	1x-**2x**	1x-**2x**	1x	1x	1x

(a) Data Cache Size

Benchmark	1.2x	1.6x	Baseline (2x)	2.4x	2.8x
compress-r1	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-**1.2x**
gcc-r1	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-**1.2x**
go-r1	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-**1.2x**
ijpeg-r1	.6x-**1.4x**	.6x-**1.4x**	.6x-1.2x	.6x-1.2x	.6x-**1**
li-r1	.8x-1.4x	.8x-1.4x	.8x-1.4x	.8x-1.4x	.8x-**1.2x**
m88ksim-r1	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-**1.2x**
perl-r1	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-1.4x	.6x-**1.2x**
vortex-r1	.8x-1.4x	.8x-1.4x	.8x-1.4x	.8x-1.4x	.8x-**1.2x**

(b) Completion Table Size (GCT)

Benchmark	1.2x	1.6x	Baseline (2x)	2.4x	2.8x
compress-r1	**c2,c4**	c2-**c4**	c2-c3	**c3**	**c3**
gcc-r1	**c4**	c3-**c4**	c3	c3	c3
go-r1	**c4**	**c4**	c2-c4	**c3**	**c3**
ijpeg-r1	c3-**c4**	c3-**c4**	c3	c3	c3
li-r1	c3-**c4**	c3-**c4**	c3	c3	c3
m88ksim-r1	c2-**c4**	c2-**c4**	c2-c3	**c3**	**c3**
perl-r1	**c2,c4**	**c2-c4**	c3	c3	c3
vortex-r1	**c4**	c3-**c4**	c3	c3	c3

(c) CORE Size

Table 4.13: Optimal design choice decisions under scaling ratios

Acceptable range windows under scaling ratios
Range2 Criteria
Bold entries indicate a deviation from the baseline

Benchmark	.876x	1.168x	Baseline (1.46x)	1.752x	2.044x
compress-r2	1x-2x	1x-2x	1x-2x	**1x**	**1x**
gcc-r2	1x-2x	1x-2x	1x-2x	**1x**	**1x**
go-r2	**2x**	1x-2x	1x-2x	1x-2x	1x-2x
ijpeg-r2	1x-2x	1x-2x	1x-2x	**1x**	**1x**
li-r2	1x-2x	1x-2x	1x-2x	**1x**	**1x**
m88ksim-r2	1x-2x	1x-2x	1x-2x	**1x**	**1x**
perl-r2	1x-**2x**	1x-**2x**	1x	1x	1x
vortex-r2	1x-2x	1x-2x	1x-2x	1x-2x	**1x**

(a) Data Cache Size

Benchmark	1.2x	1.6x	Baseline (2x)	2.4x	2.8x
compress-r2	.8x	.8x	.8x	.8x-**1x**	**1x**
gcc-r2	.8x	.8x	.8x	**1x**	**1x**
go-r2	**.8x**	**.8x**	.6x-.8x	**.6x**	**1x**
ijpeg-r2	.6x	.6x	.6x	.6x	.6x
li-r2	.8x	.8x	.8x	.8x	**1x**
m88ksim-r2	.8x	.8x	.8x	.8x-**1x**	**1x**
perl-r2	**.8x,1.2x**	**.8x**	.8x-1x	**1x**	**1x**
vortex-r2	**1.2x**	**1.2x**	1x	1x	1x

(b) Completion Table Size (GCT)

Benchmark	1.2x	1.6x	Baseline (2x)	2.4x	2.8x
compress-r2	**c2,c4**	c2-c3	c2-c3	**c3**	**c3**
gcc-r2	**c4**	c3-**c4**	c3	c3	c3
go-r2	**c4**	**c4**	c3-c4	**c3**	**c3**
ijpeg-r2	c3-**c4**	c3-**c4**	c3	c3	c3
li-r2	c3-**c4**	c3	c3	c3	c3
m88ksim-r2	**c2-c4**	**c2**-c3	c3	c3	c3
perl-r2	**c4**	**c2-c4**	c3	c3	c3
vortex-r1	c3-c4	c3-c4	c3-c4	**c3**	**c3**

(c) CORE Size

Table 4.14: Optimal design choice decisions under scaling ratios

windows and design choices. At the 1.2x ratio, *gcc* and *vortex* also had different design choices.

It is also interesting to consider the design choices that are made when taking SPECint95 as an aggregate. For the L1 D-Cache and GCT size experiments, although the acceptable range windows changed slightly, the same minimum cost design points are chosen for all scaling ratios. When considering the choice of the core size, however, the change is more drastic. At 1.2x scaling, the core size of 4 would be chosen. At 1.6x scaling, the acceptable range window was between core2 and core4, so core2 is the minimum cost choice. With 2x-2.8x scaling ratios, core3 is chosen.

4.4 Chapter Summary

We have presented details on the power models and simulator infrastructure required to perform architectural-level power analysis. We have verified these power models against industrial circuits and found our results to be generally within 10% for low-level capacitance estimates. We have also shown the relative accuracy of the models, which is especially important for architectural and compiler research on tradeoffs between different structures, is within 10-13% on average.

One limitation of the power models within Wattch is that they do not necessarily model all of the miscellaneous logic present in real microprocessors. Furthermore, different circuit design styles can lead to different results. Hence, the power models will not necessarily predict maximum power dissipation of custom microprocessors. The methodology for modeling this extra logic or other circuit design styles is the same as what we have done thus far; there is no inherent limitation to the models that prevents this additional hardware from being considered. Another limitation of the models is that the most up-to-date industrial fabrication data is not available in the public-domain, which can lead to variations in the results. The models will

be most accurate when comparing CPUs of similar fabrication technology. This is reasonable for architects considering tradeoffs on a particular design problem, where the fab technology is likely to be a fixed factor.

In this chapter, we have also considered the robustness of the relative accuracy of Wattch and PowerTimer. We have investigated the primary potential sources of error within these tools and demonstrated how design tradeoff studies can tolerate some error while still leading to the choice of the same design point.

When performing a design tradeoff study, it is most important to provide accurate power models for the unit under consideration in the study. Error in independent units will not affect the study, and errors that can affect multiple units could also have small disturbances because relative accuracy is maintained. However, errors that affect only the unit under study can lead to errors in the relative accuracy of the power model and incorrect design choices in some cases.

Chapter 5

Value Based Clock Gating

The first half of this dissertation has discussed a framework for estimating power dissipation at the architectural level, several schemes to validate the accuracy of the models, and some simple case studies demonstrating its usage. The rest of this thesis will discuss the application of these models to exploring techniques for high-performance, power/thermal-efficient design. In this chapter, we will discuss one such technique which can reduce the average power of a microprocessor, value-based clock gating.

In recent years there has been a shift towards 64-bit instruction sets in major commercial microprocessors. The increased word widths of these processors were largely motivated because addresses were getting larger; however, the size of the actual data has not increased as quickly. As high-end processor word widths have made the shift from 32 to 64 bits. there has been an accompanying trend towards efficiently supporting subword operations. Subword parallelism, in which multiple 8- or 16-bit operations are performed in parallel by a 64-bit ALU, is supported in current processors via instruction set and organizational extensions. These include the Intel MMX [71], HP MAX-2 [57], and Sun VIS [93] multimedia instruction sets, as well as vector microprocessor proposals such as the T0 project [5].

All of these ideas provide a form of SIMD (single instruction-multiple data) parallel

processing at the word level. These instruction set extensions are focused primarily on enhancing performance for multimedia applications. Such applications perform large amounts of arithmetic processing on audio, speech, or image samples which typically only require 16-bits or less per datum. The caveat to this type of processing is that thus far these new instructions are mainly used only when programmers hand-code kernels of their applications in assembler. Little compiler support exists to generate them automatically, and the compiler analysis is limited to cases where programmers have explicitly defined operands of smaller (i.e., char or short) sizes.

This chapter proposes hardware mechanisms for dynamically exploiting narrow width operations without programmer intervention or compiler support. By detecting "narrow bitwidth" operations dynamically, we can exploit them more often than with a purely-static approach. Thus, our approach will remain useful even as compiler support improves.

We have explored two optimizations that take advantage of the core "narrow width operand" detection that we propose. For both techniques, we explore both a basic and extended version of the optimization. The basic approach only operates in cases that it is guaranteed to succeed. In the extended version of the proposals, we demonstrate speculative techniques that can improve the efficiency of the optimizations.

The first optimization that we propose watches for small operand values and exploits them to reduce the amount of power consumed by the integer unit. This is accomplished by an aggressive form of clock gating. Clock gating has previously been shown to significantly reduce power consumption by disabling certain functional units if instruction decode indicates that they will not be used [40]. The key difference of our work is to apply clock gating based on operand values. When the full width of a functional unit is not required, we can save power by disabling the upper bits. With this method we show that the amount of power consumed by the integer execution unit can be reduced for the SPECint95 suite with little additional hardware.

The second proposed optimization improves performance by dynamically recognizing, at issue time, opportunities for packing multiple narrow operations into a single ALU. With this method the SPECint95 benchmark suite shows an average speedup of 4.3%-6.2% depending on the processor configuration. The MediaBench suite showed an average speedup of 8.0%-10.4%. Since this optimization falls outside the scope of this dissertation, I refer readers to previously published work which describes this optimization in detail [17; 19].

The primary contributions of this work are a detailed study of the bitwidth requirements for a wide range of benchmarks and two proposals for methods to exploit narrow width data to improve processor power consumption and performance. In Section 5.1 we further discuss the motivations for our work and place it in the context of prior work in multimedia instruction sets, power savings, and other methods of using dynamic data. Section 5.2 describes the experimental methodology used to investigate our optimizations. Section 5.3 details the power optimization technique based on clock gating for operand size and presents results on its promise. In Section 5.4, we describe speculative techniques to improve the benefits of value based clock gating. Finally, Section 5.6 concludes and discusses other opportunities to utilize dynamic operand size data in processors.

5.1 Motivation

5.1.1 Application Bitwidths

In this study we show that a wide range of applications frequently calculate using small operand values. Figure 5.1 illustrates this by showing the cumulative percentage of integer instructions in SPECint95 in which both operands have values that can be expressed using less than or equal to the specified bitwidth. (Section 5.2 will

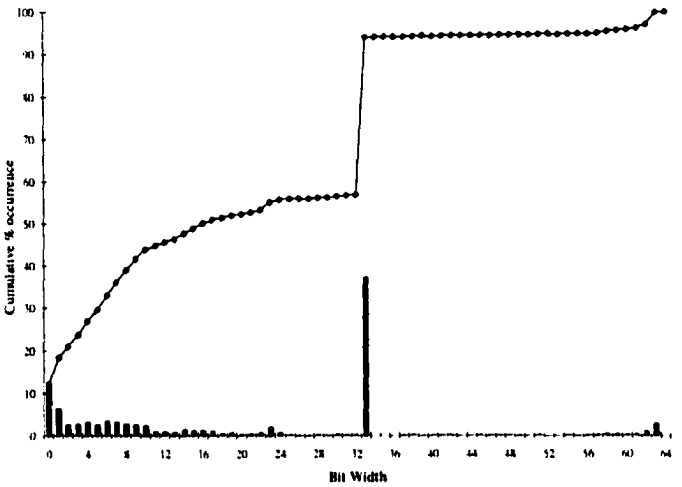

Figure 5.1: Bitwidths for SPECint95 on 64-bit Alpha.

discuss the Alpha compiler and SimpleScalar simulator used to collect these results.) Roughly 50% of the instructions had both operands less than or equal to 16-bits. We will refer to these operands as narrow width; an instruction execution in which both operands are narrow width is said to be a "narrow-width operation". Since this chart includes address calculations, there is a large jump at 33-bits. This corresponds to heap and stack references. (Larger programs than SPEC might have this peak at a larger bitwidth.) The data demonstrate the potential for a wide range of applications, not just multimedia applications, to be optimized based on narrow-width operands. While other such work, e.g., narrow bitwidth transformations to a protein-matching application [3], required algorithm or compiler changes, we focus here on hardware-only approaches.

5.1.2 Observing and Optimizing Narrow Bitwidth Operands

The basic tenet behind the optimizations proposed here is that when operations are performed with narrow-width operands, the upper bits of the operation are unneeded. To decrease power dissipation, clock gating can disable the latch for these unneeded upper bits. Alternatively, to improve performance, we propose "operation packing",

in which we issue and execute several of these narrow operations in parallel within the same ALU. In either case, the crux in exploiting narrow-width operands lies in recognizing them and modifying execution. Section 5.3 will discuss hardware approaches for tagging result operands as "narrow-width" as they are produced, and for storing these tags along with source operands as we stage subsequent instructions waiting for issue.

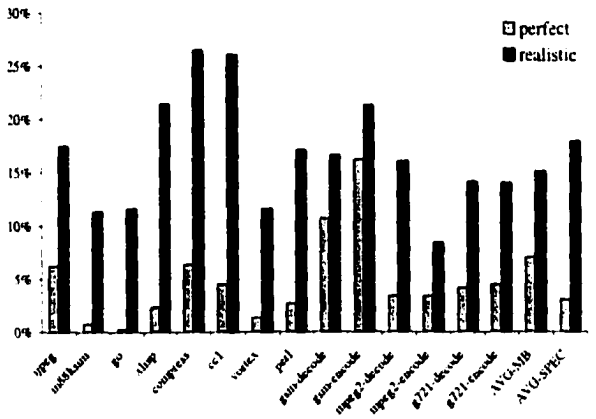

Figure 5.2: Percentage of instructions whose operand precision changes from less than 16-bit to greater than 16-bit over a single program run. Data is presented for both perfect and realistic branch prediction.

5.1.3 Disadvantages of Static Compiler Analysis

Part of the motivation for this work was the fact that *static* analysis of input operand sizes has several disadvantages. First, RISC instruction sets, such as the Alpha instruction set that we consider in this study, typically do not include instructions that specify the operand size information for each operation. For example, the Alpha ISA does not include add instructions that operate on 8-bit or 16-bit quantities. Thus the compiler could not embed operand size information without instruction set extensions. More importantly there are many cases where it is impossible to know what the true operand bitwidths (as opposed to the declared operand sizes) will be

until run-time. Actual operand sizes depend very much on the input data presented. Operand sizes for particular instructions can also vary over the program run even with the same input data, which makes the task of the compiler even more difficult.

Figure 5.2 shows the percentage of PC values where operand width changes as the instruction is executed repeatedly within a single run. In particular, the figure shows how often an instruction fluctuates from having less than 16-bit operands to greater than 16-bit operands as it executes repeatedly within a single program run. Figure 5.2 thus demonstrates some of the difficulty that a compiler would encounter in determining the operand-widths of operations statically. In particular, it is interesting to note that with perfect branch prediction, the instruction operand sizes are far more predictable than with realistic branch prediction. This is because with perfect branch prediction only the true execution path is seen. With imperfect branch prediction, uncommon paths, like error conditions, may be executed (but not committed) if the branch predictor points that way. Along these paths, operand statistics may be markedly different. Compile time analysis must conservatively analyze all potential paths to ensure that operations can truly be packed. This may include uncommon error conditions and other extreme cases. As a result, the compiler runs into much of the same diverse operand values as seen by imperfect branch prediction.

Overall, compiler dataflow analysis for operand sizes must be conservative about possible operand values. Programmer hints about operand sizes can aid the compiler. It is unrealistic, however, to assume that programmers will provide these hints on codes other than small multimedia kernels.

From Figure 5.1 it is clear that many opportunities exist to exploit narrow-width data for subword parallelism and aggressive clock gating. Searching for subword parallelism in applications is somewhat analogous to the search for instruction-level parallelism (ILP) in applications. In the late 80s and early 90s, most general purpose

superscalar microprocessors were statically scheduled, and the compiler was responsible for uncovering ILP in programs. Current microprocessors implement aggressive dynamic scheduling techniques to uncover more ILP. This evolution was necessary to feed the wider-issue capabilities of these processors. In a similar manner, more subword parallelism can be uncovered with the dynamic approaches we propose than if one relies solely on compiler techniques.

5.1.4 Related Work

The notion of disabling the clocks to unused units to reduce power dissipation in high performance microprocessors has been discussed in [41; 89]. In the CAD community, similar techniques have been demonstrated at the logic level of design. Guarded evaluation seeks to dynamically detect which parts of a logic circuit are being used and which are not [90]. Logic pre-computation seeks to derive a pre-computation circuit that under special conditions does the computation for the remainder of the circuit [2]. Both of these techniques are analogous to conditional clocking, which can be used at the architectural level to reduce power by disabling unused units.

There has been other work in specializing for particular operand values at runtime. The PowerPC 603 includes hardware to count the number of leading zeros of input operands to provide an "early out" for multicycle integer multiply operations. This can reduce the number of cycles required for a multiply from five for 32-bit multiplication to two for an 8-bit multiplication [38]. At a higher level, value prediction seeks to predict result values for certain operations and speculatively execute additional instructions based on these predicted operand values [58]. Memoing is another high-level technique that exploits data redundancy to eliminate power dissipation of long-latency integer and floating point operations [6]. Memoing is the idea of storing the inputs and outputs of long-latency operations and re-using the output if the same inputs are encountered again.

Finally, there has also been other work in detecting and exploiting narrow bitwidth operations. Razdan and Smith propose a hardware-programmable functional unit which augments the base processor's instruction set with additional instructions that are synthesized in configurable hardware at compile time [74]. Since all synthesized instructions must complete in a single cycle, bitwidth analysis is performed at compile time to highlight sequences of narrow-width operations that are the best candidates for implementation. Stephenson et al. have developed a compiler framework that detects bitwidth requirements for integers and memory addresses by statically propagating information back and forth in the dataflow graph [86]. Stefanovic use a run-time profiling tool to analyze the bitwidth requirements of applications under different accounting models for measuring bitwidth requirements [85].

Tong et al. have proposed sacrificing computational accuracy for reduced power consumption [91]. Their analysis shows that certain floating point programs suffer very little loss of accuracy with a significant reduction in bit-width. They propose minimizing the bit-width representation of floating-point data to reduce power consumption in the floating point unit. Our work differs from this technique, because we include hardware structures to dynamically detect opportunities to capitalize on narrow bitwidth operations ensuring that program will produce the same results as without the optimization.

5.2 Methodology

In Sections 5.3 and 5.4 of this chapter we present the results for the low power optimizations that we propose for dynamically exploiting small operand values. This section lays the groundwork by detailing the experimental methodology used for obtaining those results.

5.2.1 Simulator

We have used a modified version of SimpleScalar's *sim-outorder* to collect our results. In Section 2.3.1 the simulator and parameters are discussed.

Most of the changes made to the simulator for this study are localized to the issue and decode stages. In the decode stage, bitwidths are calculated for dynamic data and stored in the reservation station entry to be used during the issue stage. In the issue stage, this data is used to decide if instructions can be issued and executed in parallel based on the data from the decode stage. While these changes reflect the simulator implementation, subsequent sections discuss how our ideas would be implemented in an actual processor.

5.2.2 Benchmark Applications

A goal of this study is to demonstrate and exploit the prevalence of operations with narrow bitwidths even in applications outside the multimedia domain. For this reason we evaluate the SPECint95 suite of benchmarks as well as several benchmarks from the MediaBench suite [56]. For the power optimization we also consider eight of the SPECfp95 benchmarks.

We have compiled the benchmarks using the DEC/Compaq *cc* compiler with the following optimization options as specified by the SPEC Makefile: -migrate -std1 -O5 -ifo -non_shared. In particular, the -O5 setting, along with numerous other optimizations, provides vectorization of some loops on 8-bit and 16-bit data (char and short).

For this study we used the reference inputs for the SPEC95 suite. We did not want to use the test or training inputs because our data-specific optimizations might be unfairly helped by smaller data sets. Using the reference inputs, the SPEC95 benchmarks run for billions of instructions, which, if simulated fully, would lead to

Benchmark	Family	# of Warmup Insns. or Description	Input Data
cc1	SPECint	221M	cccp.i
perl	SPECint	601M	scrabble game
ijpeg	SPECint	824M	vigo.ppm
compress	SPECint	2576M	bigtest.in
m88ksim	SPECint	26M	dhrystone
li	SPECint	271M	All inputs
vortex	SPECint	2451M	persons.1k
go	SPECint	926M	9stone21
applu	SPECfp	1410M	applu.in
apsi	SPECfp	1400M	apsi.in
fpppp	SPECfp	1000M	natoms.in
hydro2d	SPECfp	375M	hydro2d.in
mgrid	SPECfp	1410M	mgrid.in
su2cor	SPECfp	2500M	su2cor.in
turb3d	SPECfp	1000M	turb3d.in
wave5	SPECfp	1410M	wave5.in
adpcm	Media	16 bit PCM < − > 4-bit ADPCM coder	clinton.pcm
mpeg2	Media	MPEG digital compressed format encoding	rec%d
gsm	Media	Audio and speech encoding with GSM std.	clinton.pcm
g721	Media	Voice compression using the G.721 standard	clinton.pcm

Table 5.1: Characteristics of the SPEC95 and Mediabench benchmarks studied.

excessively long execution times. Thus we have adopted a methodology similar to that described in [81]. We warm up the architectural state using a fast-mode cycle-level simulation that updates only the caches and branch predictors during each cycle. The warmup period also avoids the effects of smaller operand sizes that are prevalent within program initialization. Using the results of [81] to identify representative sections of the program run based on cache and branch prediction statistics, we then simulate a 100 million instruction window using the detailed simulator. Table 5.1 lists the reference input that we have chosen for the SPEC95 benchmarks, and the number of instructions for which we warm up the caches and branch predictor. Table 5.1 also describes the applications chosen from the MediaBench suite. For the MediaBench suite, *gsm*, *g721*, and *mpeg2-decode* were run to completion while *mpeg2-encode* was simulated for 100 million instructions after a 500M instruction warmup period.

5.3 Proposal: Value Based Clock Gating

5.3.1 Clock Gating

Dynamic power dissipation is the primary source of power consumption in CMOS circuits. In CMOS circuits, dynamic power dissipation occurs when changing input values cause their corresponding output values to change. Only small leakage currents exist as long as inputs are held constant. Clock gating has been used to reduce power by disabling the clock and thereby disabling value changes on unneeded functional units. In static CMOS circuits, disabling the clock on the latch that feeds the input operands to functional units essentially eliminates dynamic power dissipation. Power consumption on the critical clock lines is also saved because the latch itself is disabled. In dynamic or domino CMOS circuits, the same effect can be obtained by disabling the clocks that control the pre-charge and evaluate phases of the circuit. The use of

clock gating may introduce additional clock skew and can complicate timing analysis which provide challenges for circuit designers performing the implementation. Despite these difficulties, conditional clocking is commonly used in current microprocessors [41].

Currently most work on clock gating has focused on using the decoded opcode to decide which units can be disabled for a particular instruction. For example, *nop*'s allow most of the units to be disabled since no result is being computed. As another example of opcode-based clock gating, consider an "add byte" instruction. Since the opcode *guarantees* that only the lower portion of the adder is needed, the top part of the functional unit can be disabled.

Proposed Architecture

Our approach proposes a more aggressive clock gating approach and quantifies its benefits. At run-time, it determines instances when, based on the input operands, the upper bits of an operation are not needed; in those cases, it disables the upper portion of the functional unit. The key differences from prior approaches are that (1) our approach is operand-based, not opcode-based, and (2) our approach is dynamic, not static. (One could, of course, use our method *in addition to* prior opcode-based approaches.) Different runs of the program, or even different executions of the same instruction, can dissipate different amounts of power depending on the operands seen.

There are several different possible hardware implementations for this technique. Figure 5.3 is a diagram of one possible implementation. This unit recognizes that the upper bits of both input operands are zeros. For example, in an addition operation, if both input operands have all zeros in their top 48 bits, these bits do not have to be latched and sent to the functional units. We already know that the result of this part of the addition will be zero, and thus zeros can be multiplexed onto the top 48 bits of the result bus, rather than computed via the adder. In this architecture the low

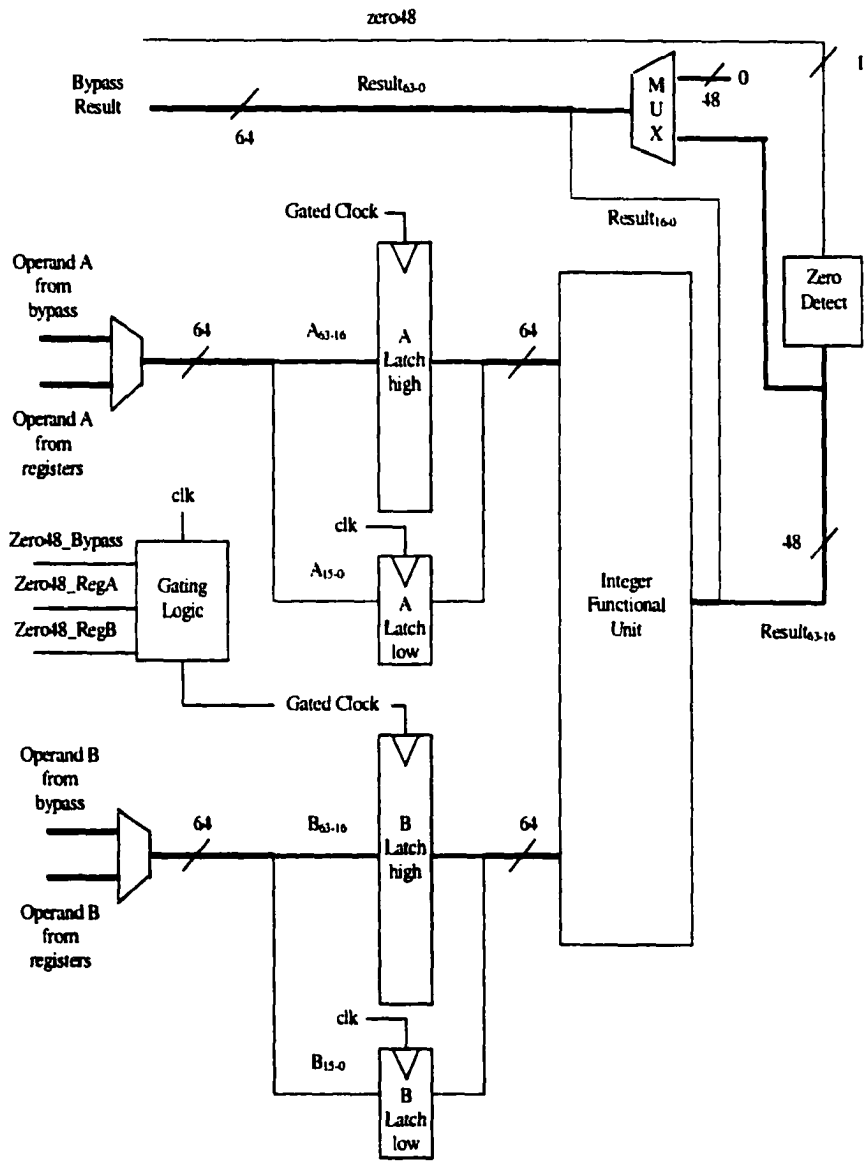

Figure 5.3: Clock gating architecture.

16 bits are always latched normally. The high 48 bits are selectively latched based on a signal that accompanies the input operand from the reservation stations or the bypass network. This signal, called *zero48* in Figure 5.3, denotes that the upper 48 bits are all zeros and is created by zero detection logic when the result was computed. Since some operands come directly from the cache, there must also be a zero-check during load instructions. We believe such zero-detect hardware and corresponding flags within the reservation stations are already present in some processors; it is used, for example, to recognize divide-by-zero exceptions early. However, in some processors it may not be possible to perform zero-detects on incoming loads, and in these cases the hardware will not recognize an opportunity to gate the clock. For the SPECint95 suite, 13.1% of power saving instructions have one or more operands that come directly from a load instruction; these are the instructions that would be missed if zero-detect were omitted on loads. The percentages for the media benchmarks are much lower at 1.5%.

The gated clock signal used to disable the upper 48 bits of the functional unit is generated based on the *zero48* signals of the respective operands and is combined with an AND gate in parallel with data bypass multiplexing. In the case of functional units designed with static logic this signal can be used to disable the upper 48 bits of the preceding latches thus effectively reducing the switching activity to zero. For functional units design with dynamic logic, the *zero48* signal would be placed into the latches and used in the next cycle to disable the clock on the upper 48 bits of the functional unit.

In Figure 5.3 the *zero48* signal is generated after the functional unit completes the specified operation. In processors with architecturally visible zero-flags such as the Intel x86, Motorola 68K, and IBM/Motorola PowerPC architectures, this approach would be feasible because there would be no additional serial delay introduced. However, in other architectures in which adding a zero-detect in the execute stage would

affect cycle time, another implementation is possible. This implementation relies on the fact that if we know that the two source operands of an operation are 16 bits or less, then it is relatively easy to determine whether or not the result will be 16 bits or less. For example, with an arithmetic operation, if the carry-out signal of the 16th bit is zero and the two source operands are 16 bits or less, then we know that the result will be 16 bits or less. Thus, the *zero48* signal can be computed after the carry-out of the 16th bit is generated, well before the final adder result is finished. Finally, in some cases a designer might not want to insert the *zero48* signal into the register file or reservation stations. In this case, the 48-bit zero-detects could be inserted after register fetch while waiting for the bypass results to be returned. This relies on the fact that register read generally takes place in the first half the cycle and writeback occurs in the second half of the cycle.

In order for any power saving technique to be useful, it must save more power than it consumes. In our technique, the new power dissipated is mainly in the zero-detection logic and in widening the mux onto the result bus. The primary power savings stems from selectively clock-gating the functional units based on the results of the zero-detection logic. In the following subsections we evaluate these costs and benefits in more detail.

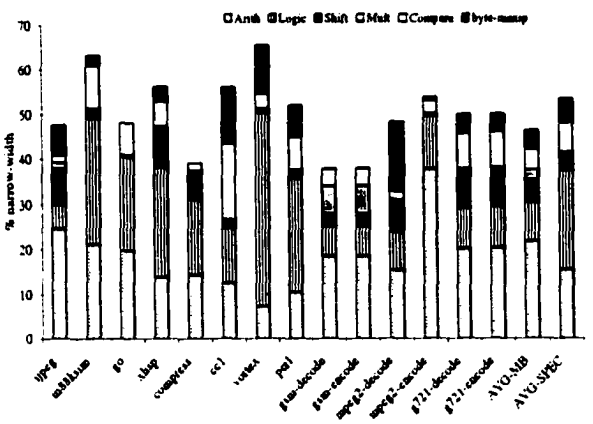

Figure 5.4: Operations with both operands 16 bits or less.

Bitwidth Analysis of Benchmarks

The success of our approach relies on the frequent occurrence of narrow bitwidth operands. Figure 5.4 shows, for each benchmark, the percentage and type of operations whose input operands are both less than or equal to 16 bits. (Both operands must be small in order for the clock gating to be allowed.) The breakdown by operation type is another important metric. Intuitively, disabling the upper bits on an adder or multiplier will save more power than turning off the upper bits on the less power-hungry logical functions. Figure 5.4 shows that for most benchmarks, arithmetic and logical operations dominate the number of narrow-width operations. In most of the benchmarks multiplies are rather infrequent although they do account for 6% of the narrow-width operations in *gsm*.

Recall that Figure 5.1 illustrated how address calculations result in many operations with bitwidths of 33. Roughly 94% of SPECint95 compute operations had bitwidth requirements of 33 or less with 37% occurring at the 33-bit mark. From this data it makes sense to include a second control signal for clock gating of operands that are 33 bits or less. The zero detect logic can be shared so that the extra hardware requirements are minimal. This modification is also useful for optimizing the multiplication of two 16-bit numbers. In these cases a 32-bit result can occur, so the 33-bit mux onto the result bus would be used as shown in Figure 5.5. Figure 5.5 also shows the expanded clock gating architecture with clock gating at the 16-bit and 33-bit boundaries. The operand latches have been further partitioned and an additional clock gating signal is generated. In sections 5.3.3 and 5.3.4 we discuss the choice of the bitwidths to clock gate in more detail.

Negative numbers provide another source of narrow-width data for operand-based clock gating to exploit. In the Alpha architecture that we considered in this study, the fundamental datum is the 64-bit quadword. Quadword integers are represented with a sign bit occupying the most significant bit [11]. Numbers are expressed in two's

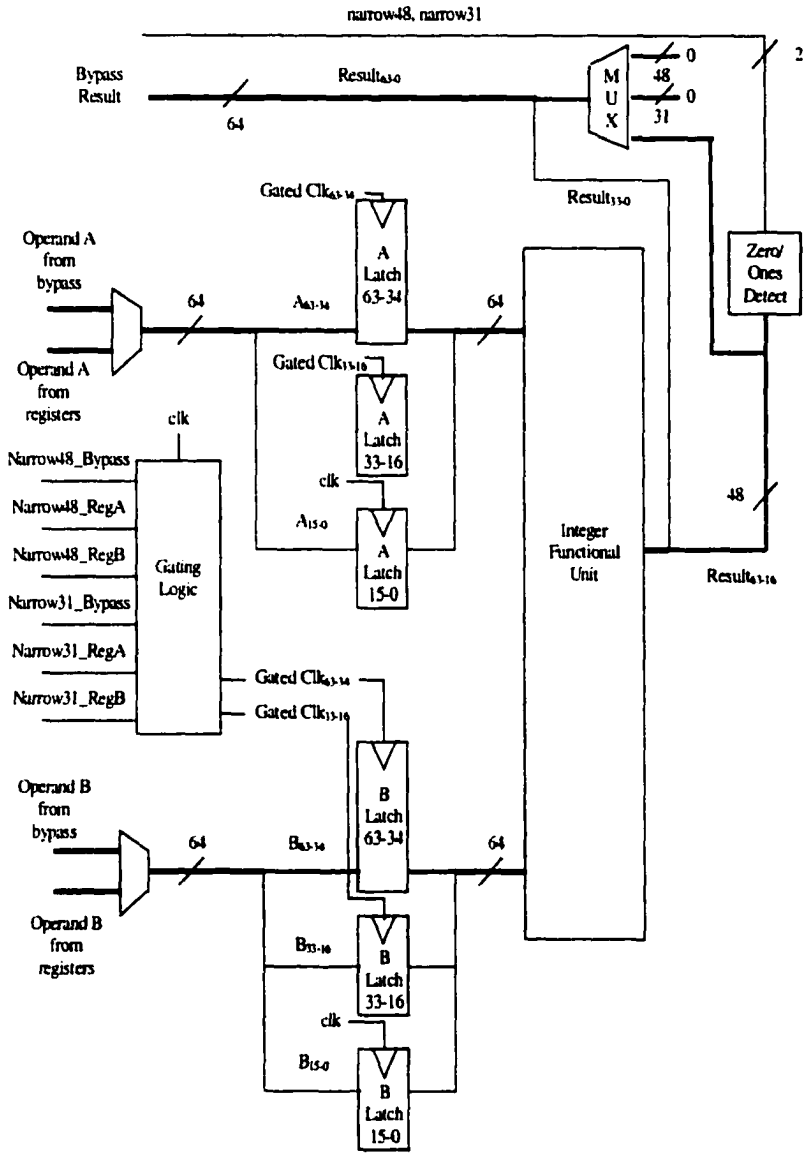

Figure 5.5: Expanded clock gating architecture with 33-bit gating.

complement form which simplifies arithmetic operations. The techniques presented in this work rely on determining when data requires less than the full word width of the machine. For positive numbers, this can be accomplished by performing a zero detect on the high order bits. For negative numbers in the two's complement representation, leading 1's signify the same thing that leading 0's do for positive number - essentially unneeded data. Thus a ones detect computation (simply an AND of the high-order bits) must be performed in parallel with the zero detect computation to detect narrow bitwidth negative numbers. An additional signal does not need to be stored in the register file because this information can be derived by sampling one of the higher order bits. Figure 5.5 shows the zero and ones-detect logic which creates the signals *narrow31* and *narrow48* (analogous to the *zero48* from Figure 5.3).

5.3.2 Power Results: Overview

Device	32-bit	48-bit	64-bit
Adder (CLA)	105	158	210
Booth Multiplier	1050	1580	2100
Bit-wise Logic	5.8	8.7	11.7
Shifter	4.4	6.6	8.8
Zero-Detect	-	4.2	-
Additional Muxes	-	3.2	-

Table 5.2: Estimated power consumption of functional units at 3.3V and 500Mhz (mW).

The amount of power that is saved by our approach depends on both the type and frequency of narrow-width operations. In order to quantify the amount of power saved, we use previously-reported research to estimate the amount of power that various functional units use [13; 28; 67; 98]. From these sources we obtain power estimates assuming dynamic logic and relatively fast carry look-ahead adders. We assume that the power scales linearly with the number of bits of the units based

[66]. We assume that the multiplier is pipelined with its power usage scaling linearly with the operand size. Again, the zero-detect for 33 bits can be computed within the 48-bit zero-detect so no additional power is consumed. Table 5.2 summarizes the values that we have assumed for different size devices. The functional units in current high-end microprocessors are likely to use even more power. For this analysis though, the important factor is the ratio of the respective functional units to each other.

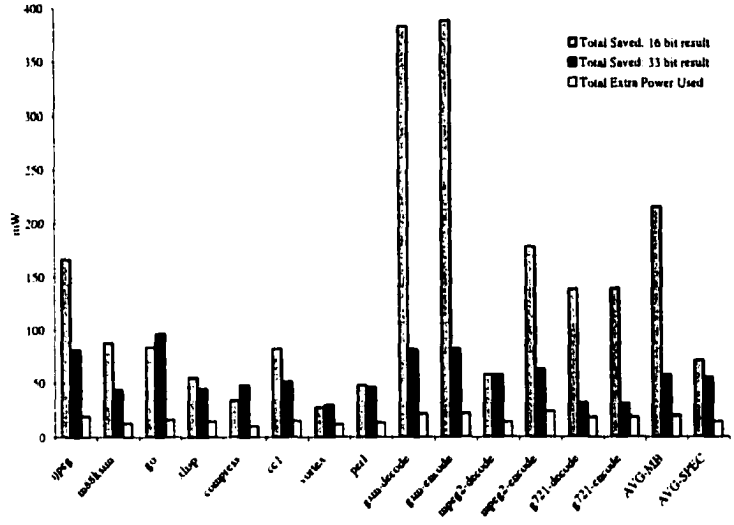

Figure 5.6: Net power saved by clock gating at 16 and 33 bits. Total extra used is the amount used by zero detection and multiplexing. Net savings is equal to the amount saved at 16 bits plus the amount saved at 33 bits minus the amount used.

Figure 5.6 summarizes the amount of power saved and expended by the integer execution units. We arrived at these numbers by determining the amount of power saved and expended per operation executed and multiplying by the average issue rate. These results include all loads, stores, branches, and other integer execution unit instructions that are not part of the set of instructions that our optimization applies to. Among the SPECint95 benchmarks, our technique saves the most power for *ijpeg* and *go*. *Ijpeg* has a large number of narrow-width arithmetic operations. *Go* includes a large number of address calculations and is helped the most by adding the extra signal to detect 33-bit operations. The media benchmarks tend to save

even more power than the SPECint95 benchmarks. This is primarily because of the larger number of arithmetic operations. *GSM*, in particular, has a relatively large number of narrow bitwidth multiply operations. The amount of power used by the zero detection circuitry is small and nearly constant for all benchmarks. In no case does the amount of power used for zero detection exceed the amount of power saved.

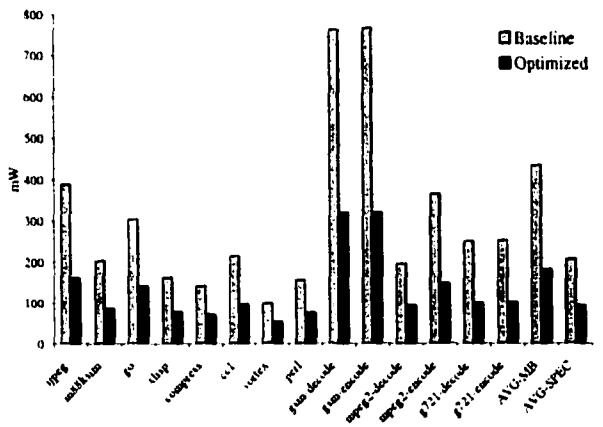

Figure 5.7: Power usage of integer unit.

Figure 5.7 shows the total amount of power that is saved by the integer unit with our optimization. For the baseline system, we assume that all operations use the amount of power that a 64-bit device would use. (We assume basic clock gating in which, for example, multipliers are turned off for add instructions and vice versa.) For the SPECint95 benchmark suite, the average power consumption of the integer unit was reduced by 54.1%. For the media benchmarks, the reduction was 57.9%.

While a 50-60% power reduction seems exceptional, it is important to note that the integer unit's contribution to total power varies depending on the CPU. In some high-end CPUs much of the power is spent on clock distribution and control logic, and thus the integer unit represents only about 10% of the power dissipation [41]. In such a processor, our optimizations will lead to 5-6% power reductions on average. As control is streamlined, either in DSPs or via explicitly-parallel instruction computing (EPIC) as in future Intel processors [35], the integer unit is a larger factor in the processor's

total power dissipation, as much as 20-40% [54]. In these cases, the total power savings from our technique will approach 20%. In all processors, our approach promises a relatively easy way to prune power from the integer unit where this is important. We also note that our power savings estimates are somewhat conservative. The clock gating technique also reduces the switch capacitance seen by the clock distribution network, and this can lead to a further power reduction. Although this effect can be significant, it cannot be quantified without a chip floorplan.

5.3.3 Selecting Gating Boundaries

In the previous subsections, data has been presented for clock gating at 16-bit and 33-bit boundaries. The choice of the 33-bit mark was motivated because the empirical data demonstrated that a large number of operations exist with both source operands 33 bit or less, primarily due to address calculations. The reason for choosing the 16-bit mark is more arbitrary and reflects the need to balance two tradeoffs in the selection of the boundary at which to clock gate. First, if the boundary is chosen to be too large, the amount of power saved will not be as significant as possible. On the other hand, if the boundary is selected to be too small, not enough operations will be eligible for clock gating at that boundary.

In this subsection, we systematically investigate the selection of the clock gating boundary. In this analysis, we limit the number of boundaries that are clock gated to one or two. We also assume that the power dissipation of the functional units scales linearly at the bit-level. In the next section, we investigate the potential for clock gating at more than two points with finer granularities.

Figure 5.8 shows the integer unit power reduction by having one clock gating boundary at the specified bitwidths. The data is shown as a percentage power reduction relative to the original integer unit power. Clearly, if we are only allowed to clock gate at one point, we should clock gate at 33 bits. Clock gating at points

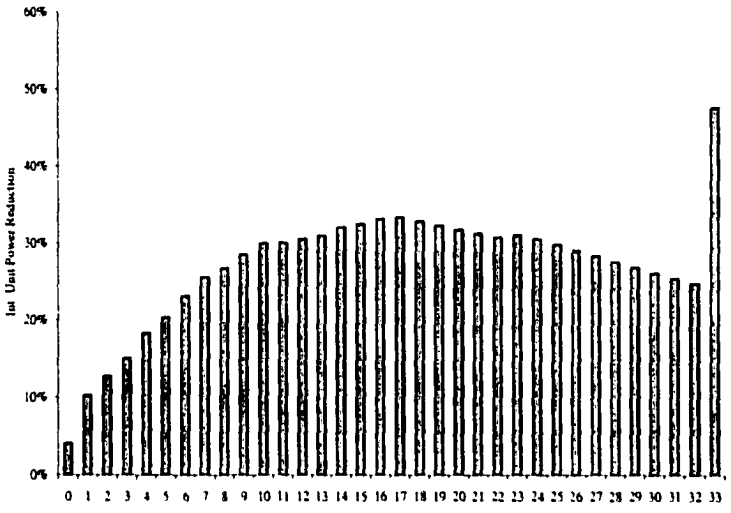

Figure 5.8: Integer unit power reduction by selecting to gate at one bitwidth.

beyond 33 bits does not make sense for this set of benchmarks, because they rarely utilize the upper portion of the functional units. Section 5.3.5 will discuss floating point benchmarks in more detail. Future applications written for 64-bit CPUs may use larger values more frequently, but we typically expect this usage to grow slowly from the 33-bit mark as addressing needs grow.

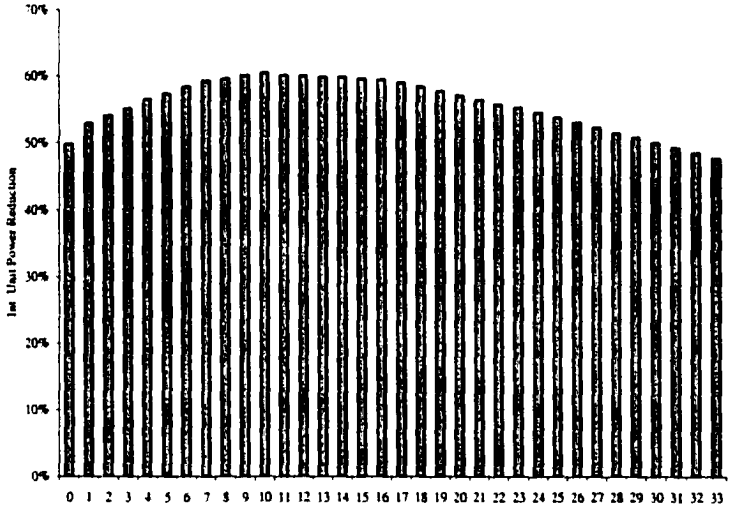

Figure 5.9: Integer unit power reduction by clock gating at 33 bits as well as at the specified bitwidth.

Figure 5.9 shows the power savings assuming that we now are able to clock gate at two points. One of the two points is always chosen to be 33 bits, capturing the large number of address calculations. The second point varies, with each bar measuring the total amount of power saved by clock gating at that bitwidth as well as at 33 bits. Figure 5.9 demonstrates that choosing the clock gating point to be anywhere from ten to seventeen results in very little difference in the total amount of power saved. Thus our original choice of clock gating at 16 bits was reasonable. On the other hand, certain benchmarks display a preference for clock gating at a particular bitwidth. This can affect the total amount of power saved significantly. For example, the optimal selection of clock gating boundary for *m88ksim* is 5 bits. Clock gating at this bitwidth would save approximately 10% more power than our default selection of 16 bits.

5.3.4 Selecting the Number of Clock Gate Boundaries

In the previous subsection we investigate the optimal selection of clock gating boundaries for one and two points. In this subsection, we investigate the potential for clock gating at multiple points at finer granularities. For example, instead of clock gating just at 16 bits and 33 bits, as in our original proposal, another choice might be to clock gate four bitwidths: the 8-bit, 16-bit, 24-bit, and 33-bit boundaries.

Figures 5.10 and 5.11 show the percent of the integer unit power saved by clock gating at the specified granularities for the SPECint95 and multimedia benchmarks. In these figures, the last bar assumes clock gating at only the 33-bit boundary. The second to last bar is similar to our original proposal, in which we clock gate at 16-bits and 33-bits. The remaining three bars show the improvement by clock gating at additional, finer granularities. These figures show the diminishing marginal returns for clock gating as we approach 1 bit of granularity. The data suggests that our original proposal with two boundaries at 16 and 33 is close to optimal. If additional

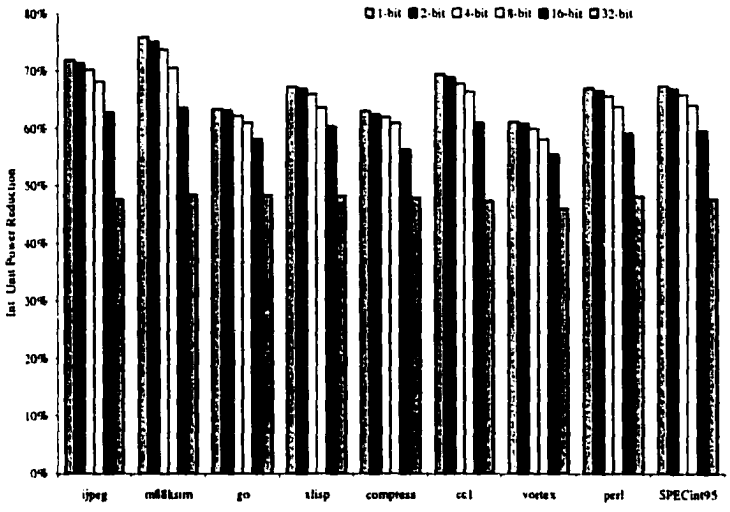

Figure 5.10: Percent of integer unit power saved with varying clock gating granularities.

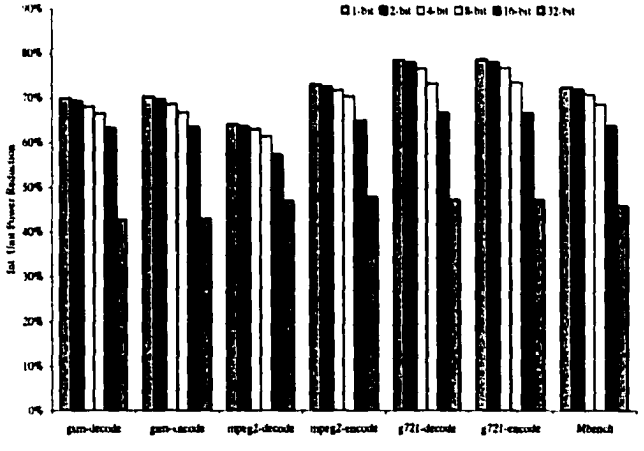

Figure 5.11: Percent of integer unit power saved with varying clock gating granularities.

boundaries are desired, then 8-bit boundaries provide slightly better power savings.

5.3.5 Value-based Clock Gating in Floating Point Benchmarks

In this section we discuss value-based clock gating within floating point benchmarks. Here we will consider clock gating on both integer data, as in the previous sections, and within certain types of floating point operations.

Clock Gating Integer Code in Floating Point Benchmarks

Floating point benchmarks often contain a significant percentage of integer code in addition to floating point operations. Integer code in floating point benchmarks is often used for loop index variables and address computations. In the integer benchmarks that we studied, roughly 50% of the instructions are integer computations that are available for clock gating. In the floating point benchmarks approximately 25% of the instructions are integer computations. The integer computations within these benchmarks tend to have a larger percentage of arithmetic operations which consume more power than the other classes of instructions. Thus the power consumption within the integer unit is significant within these benchmarks.

Figure 5.12 presents the data for functional width analysis on the integer code within SPECfp95. This graph is similar to Figure 5.1 in which we present the data for SPECint95. The main difference between the two graphs is that the spike at 33 bits is larger, corresponding to the fact that address calculations will be a larger percentage of the integer code than within floating point programs. Still, about 37% of the operations require 16 bits or less to perform their computation.

We next present data on the power saved by clock gating the floating point benchmarks. We assume that we will definitely want to clock gate at the 33 bits. Figure

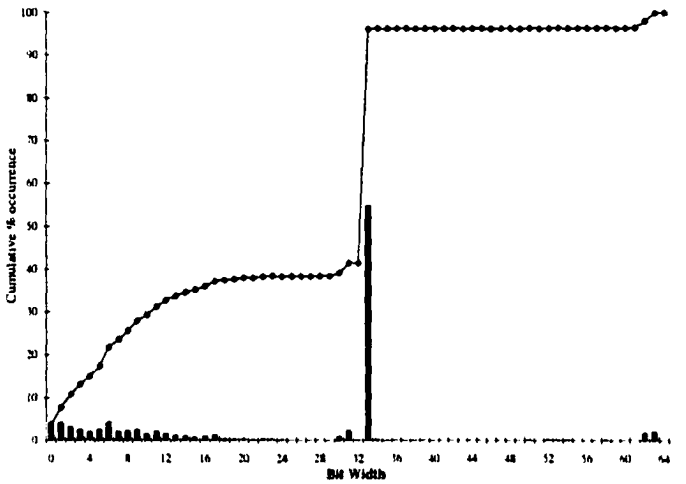

Figure 5.12: Bitwidths for integer computation in SPECfp95 on 64-bit Alpha.

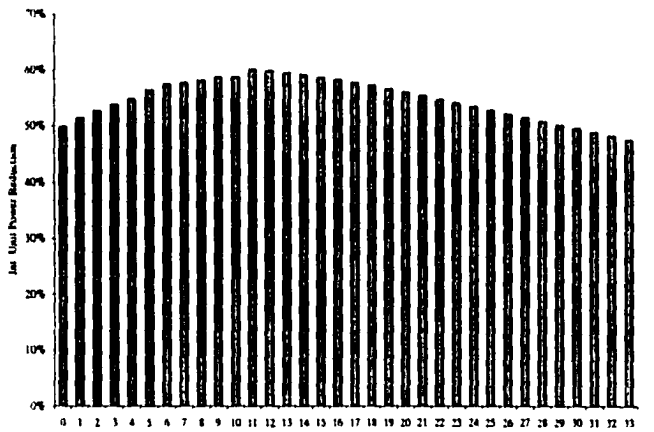

Figure 5.13: SPECfp95 integer unit power reduction by clock gating at 33 bits as well as at the specified bitwidth.

5.13 shows that the optimal mark for placing the second clock gating mark is at the 11-bit mark. However, the difference between choosing the 11-bit mark and the 16-bit mark that we chose before is only 2%, so we can use 16-bits to keep the same hardware structure as the original proposal for the integer benchmarks.

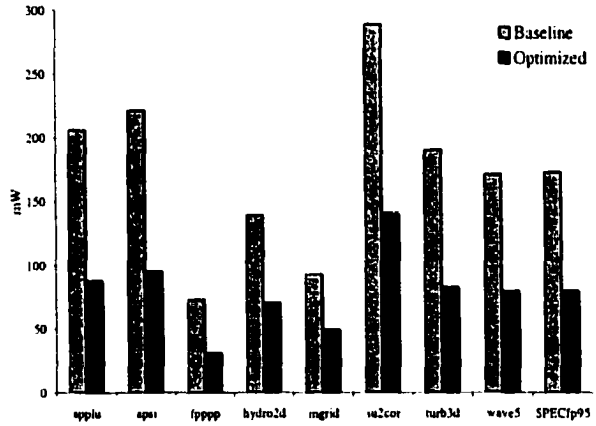

Figure 5.14: SPECfp95 power usage of integer unit.

Figure 5.14 shows the total power used by the integer assuming the baseline and clock gated configurations. The percentage savings of the clock gated configuration is still over 50%. However, as expected the total power used and saved within the integer unit is less than before. Hence the optimization would have less of an effect on the overall power dissipation of the processor for these floating point programs.

Clock Gating Floating Point Operations

Applications with floating point code tend to have higher overall power dissipation. This is because floating point operations are much more complex and hence use more power. For example, floating point programs tend to have a larger number of power hungry multiplication operations. We will focus on these multiplication operations in this section for two reasons. First, in floating point arithmetic, multiplication is simpler than addition and subtractions in that it does not require shifting an operand to align them before performing the computation. Essentially, the mantissas of the

input operands are multiplied together and the exponents of the input operands are added together. Second, since multiplication is more expensive in terms of power dissipation there is more potential for power savings.

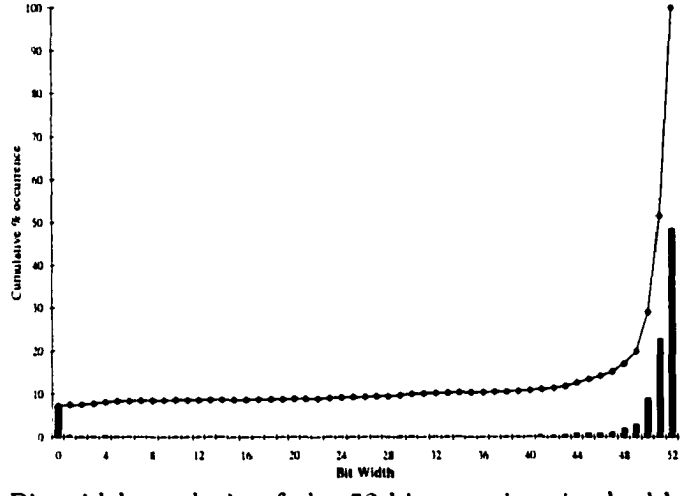

Figure 5.15: Bit width analysis of the 52-bit mantissa in double precision floating point multiplication

According to IEEE Standard 754, 64-bit double-precision, floating point arithmetic uses a mantissa of 52 bits, an exponent of 11 bits, and one sign bit [47]. We consider clock gating on input operands of the 52-bit integer multiplication operation that occurs in double precision multiplication. In single precision operations, the lower 29 bits are all zeros. Single precision multiplication uses the same functional units as double precision multiplication and would present many additional opportunities for clock gating. However, we do not consider them here because traditional opcode-based clock gating techniques would be sufficient to capture these situations.

Figure 5.15 presents the bit width analysis for the 52-bit mantissa in double precision floating point multiplication. Most often the operations require nearly the full 52 bits of precision. However, roughly 10% of the operations require less than 4 bits of precision. Despite the small number of instructions that are amenable to clock gating, being able to clock gate nearly the full width of the multiplication saves an

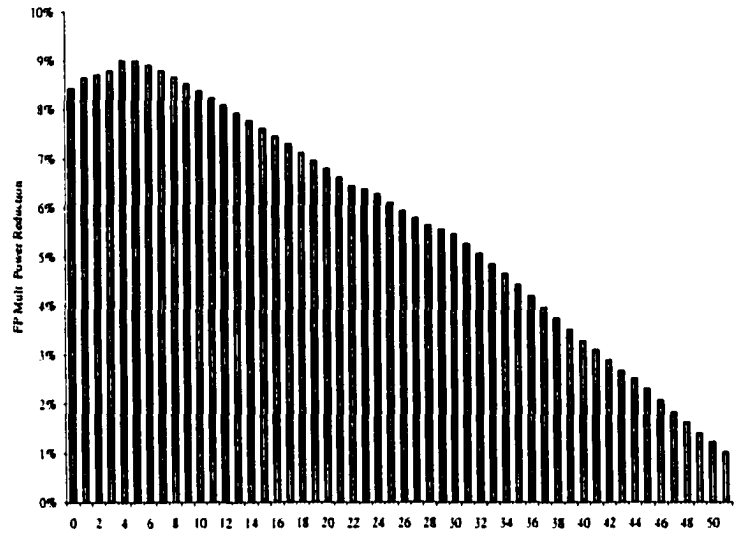

Figure 5.16: FP multiplier power reduction by selecting one clock gating point in the 52-bit mantissa.

appreciable amount of power. Figure 5.16 shows the power saved by selecting one clock gating point within the 52-bit mantissa. By selecting gating at the 4-bit boundary, approximately 9% (18mW) can be saved. This compares to about 125mW saved by clock gating operations in the integer benchmarks, and about 100mW saved by clock gating integer operations in the floating point benchmarks.

5.4 Speculative Approaches for Exploiting Narrow-Width Operands

The power optimization discussed in Section 5.3 requires that both input source operands be less than 16-bits to operate most efficiently. For the power optimization, if the first input operand is less than 16-bits and the second operand is greater than 16-bits, yet still less than 33-bits, it will be clock gated at the 33-bit mark rather than the more optimal 16-bit mark.

The requirement that both input operands be less than 16-bits excludes a large

number of arithmetic operations used for memory addressing, loop incrementing, etc. In many of these cases, one of the input operands may be very large, while the other is quite small. When this is true, it is possible that adding them will result in a carry that ripples into the highest bits, but in practice, such large ripple carries occur infrequently. Based on this observation, we present extensions here to value based clock gating that allows the optimization to proceed speculatively assuming that there will be no overflow from the 16-bit operation; the high 48 bits of the larger source operand can be muxed onto the result bus to proceed into the destination RUU stations. However, in the rare cases that there is overflow from the 16-bit addition, the instruction can be squashed and subsequently re-executed as a full-width instruction. Such a situation could be handled in a similar manner to "replay traps", which are already available for other reasons in the Alpha 21164 and other CPUs [15].

5.4.1 Replay Clock Gating for Arithmetic Operations with Varying Operand Sizes

In this section we investigate the benefits of speculatively clock gating operations at the 16-bit mark when one source operand is less than 16-bits and the other source operand is greater than 16-bits. We will call this technique *replay clock gating.*

When the replay clock gate operation succeeds, the power savings are similar to those previously presented. We must also, however, account for the cases when the 16-bit addition has carry-out and the instruction must be re-executed. These *replay overflows* incur both a performance and a power penalty. Because of this, we would like the percentage of instructions that overflow the 16-bit boundary to be low. Figure 5.17 demonstrates that for most of the benchmarks this is true. This figure shows the percentage of replay clock-gated operations that overflowed the 16-bit boundary. For the SPECint95 benchmark suite, about 9% of the speculatively

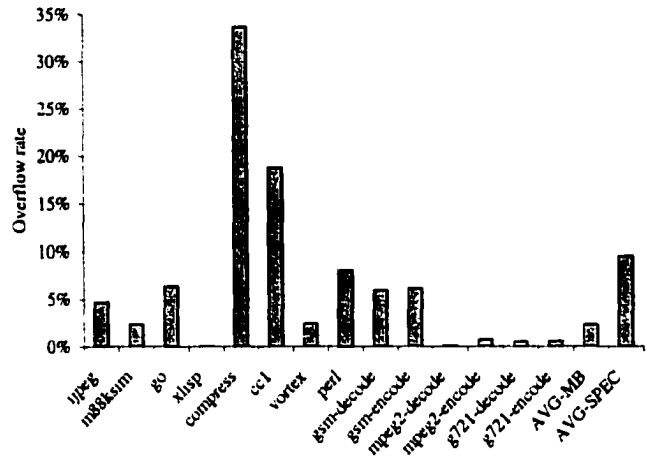

Figure 5.17: Percentage of replay clock gated instructions that overflow the 16-bit boundary.

clock gated instructions did have overflow. The multimedia benchmarks, having more regular data types and ranges, had a overflow rates of only 2%. *Compress* (33%) and *cc1* (18%) had the highest overflow rates.

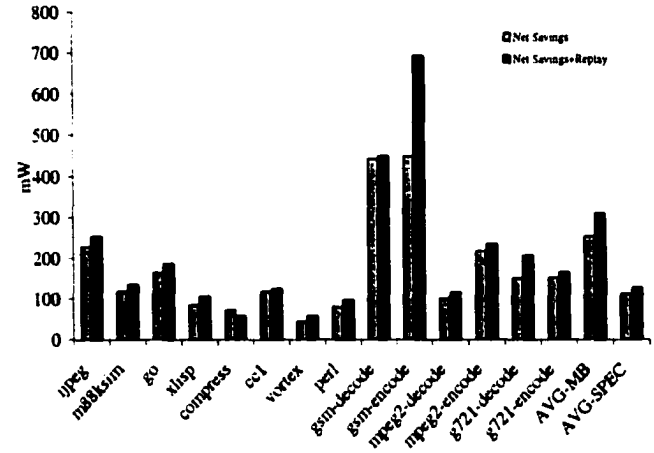

Figure 5.18: Net savings with and without replay clock gating.

In computing the net power saved via replay clock gating, we attempted to charge operations with a power cost when they overflow and need to be re-executed. We also took into account the power cost of re-issuing instructions in the previous pipeline stage that were dependent on the squashed instruction. Computing the amount of

power used when re-executing is fairly straightforward; we simply charge the instruction with the cost of a second add (usually 33-bits, assuming 33-bit clock gating was valid). Estimating the amount of additional power consumed to re-issue the dependent instruction is more difficult and depends heavily on the actual implementation details of the processor. We used the Wattch infrastructure discussed in Chapter 2 to provide these estimates.

Figure 5.18 shows the net savings with and without replay clock gating. The net savings with replay includes the amount of additional power saved on replay clock gated instructions as well as an estimate for the amount of extra power dissipated due to replay overflows. The amount of additional power saved was approximately 12% for SPECint95 and 21% for the multimedia benchmarks. However, as expected the benchmarks did not perform uniformly. In fact, the net savings for *compress* was 20% lower when using replay clock gating; its unusually large number of replay overflows incur additional power consumption. Figure 5.19 shows the two components of the additional power used when replay overflow occurs. The power used to re-execute instruction is about 2-3X higher.

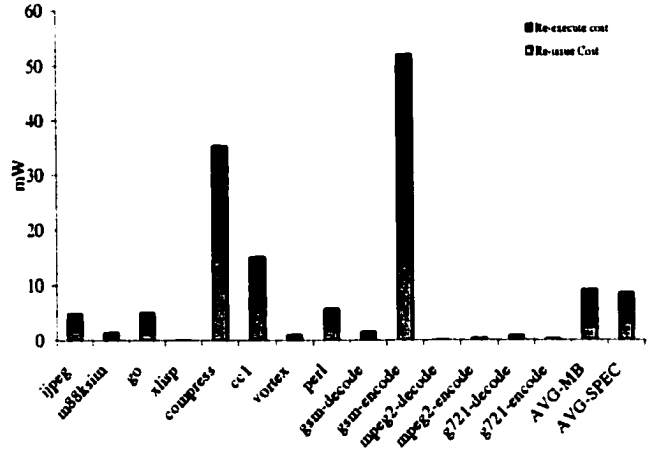

Figure 5.19: Additional power used when replay overflow occurs.

In addition to consuming additional power, re-issuing and executing instructions

can lead to performance degradation. All of the benchmarks we considered, except *compress*, performed within 0.5% of the baseline system when using replay clock gating. *Compress* suffered a 4% performance degradation due to the large number of replay overflows.

Overall, replay clock gating has mixed results. For most of the applications in the benchmark suite, the additional power savings benefits are attractive. However, for *compress* the performance degradation is noticeable.

While speculatively applying clock gating has mixed results, we found that speculative approaches for operation packing are quite successful [19]. We call this technique *replay packing*. Replay packing achieved speedups of 4.3%-6.2% for SPECint95 and 8.0%-10.4% for the multimedia benchmarks. This is a significant improvement over the non-speculative version of the operation packing optimization.

5.4.2 Summary of Results

Sections 5.3 and 5.4 have explored value based clock gating in order to exploit the detection of narrow-width operations at run-time. We discussed both speculative and non-speculative versions of the optimization.

For this optimization, the non-speculative version of the clock gating optimization seems like the best choice. While the speculative optimization saved approximately 20% more power, performance may be sacrificed for some applications, since instructions must be re-issued after a misspeculation.

The speculative technique is most successful when the 16-bit overflow rate is low as shown in Figure 5.17. Overflow confidence predictors could be used to decrease the overflow rates by recording the 16-bit overflow history of arithmetic operations to determine whether it is expected to be useful to perform the replay gating/packing. This would decrease the replay overflow rate and hence the benefits of replay clock gating.

5.5 Value-Based Clock Gating in an Industry Context

This chapter discusses the potential benefits and possible implementation of value-based clock gating and operation packing. This study was performed within an academic environment and while many details were considered, industrial circuit design styles and pipeline designs could have an impact on this technique. In this section we discuss the results of a study on the analysis and one potential implementation of value-based clock gating within a high-performance Itanium^{TM}(IA64) processor family microprocessor design. This study was conducted as part of a summer internship project at Intel Corporation.

5.5.1 IA64 Bitwidth Analysis

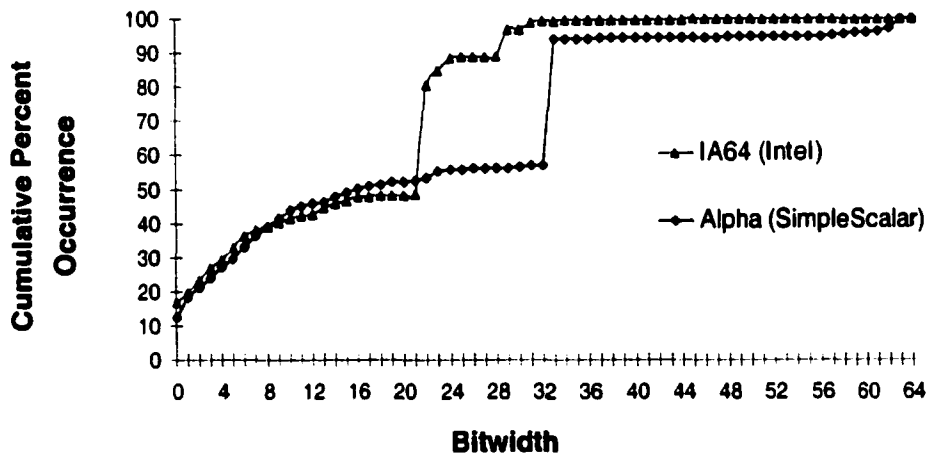

Figure 5.20: Bitwidths for SPECint95 for IA64 and 64-bit Alpha.

In Figure 5.20 we compare the bitwidth analysis that was performed on the Alpha ISA during the academic study with a similar analysis that was performed at Intel on the IA64 ISA using an internal research simulator. The results of the analysis are

very similar up to the 20-bit mark. Around this region, there are two small spikes which show the percent of integer compute operations which required 20 and 28-bits to perform their computation. These points most likely correspond to the location where the IA64 compiler used in this study performed address calculations. This compares to the 33-bit mark with the Alpha compiler and ISA. Note that the results depend on whether or not the binaries are generated to take advantage of the entire 64-bit address space. For this study, we used SPECint95 binaries generated from the IA64 research compiler.

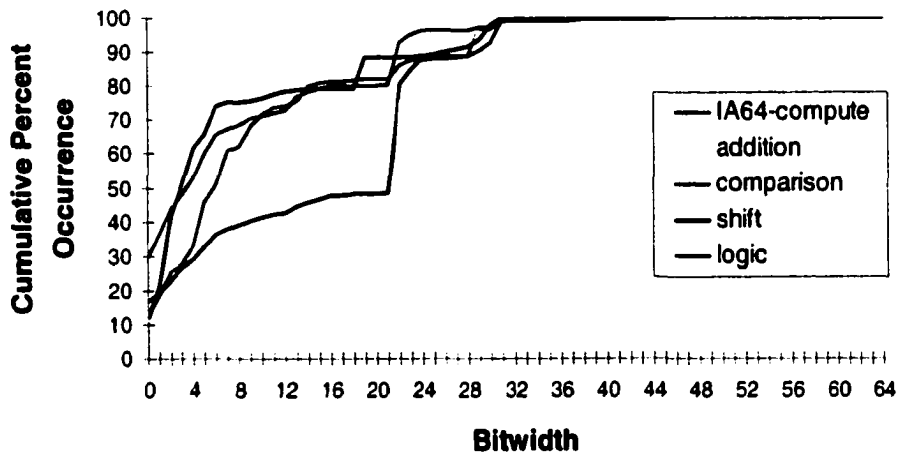

Figure 5.21: Breakdown of Bitwidths for SPECint95 for IA64 and 64-bit Alpha.

Figure 5.21 shows the bitwidth and operation type breakdown for SPECint95 with the IA64 ISA. About 80% of the comparison, shift, and logic operations occur at the 16-bit mark or below, while roughly 35% of the pure addition operations occur at the 16-bit mark or below.

Overall, the bitwidth analysis on the IA64 ISA was very similar to the Alpha ISA. This suggests that a similar value-based clock gating implementation can be useful for IA64 processors, although the exact location of the value-gating mark may change. For example, it may be sufficient to perform value-gating at the 28-bit mark instead

64-bit Adder	Randomized	Zeros	Ijpeg Stream
Upper 32 Disabled	42.6%	34.7%	37.5%
Upper 48 Disabled	62.4%	53.8%	58.8%
Selective Gating	N/A	N/A	46.3%

Table 5.3: Reduction in power dissipation for a 64-bit adder under various gating schemes

of the 33-bit mark.

5.5.2 Value Based Clock Gating Implementation

Circuit-level analysis was performed to determine how much power could potentially be saved with value-based clock gating under a variety of operating conditions. These experiments were performed with PowerMill, a commercial circuit-level power-estimation tool. The circuit and layout design for the 64-bit arithmetic units within the high-performance IA64 processor design was used for this study. The average power dissipation for 100 test vectors as reported by PowerMill.

Three different input streams were applied to the adders under three different gating conditions. The three different input streams are randomized inputs, a stream where all of the inputs are zeros, and a stream which was captured from arithmetic computations within the *ijpeg* application (and had a representative mix of arithmetic computation values). The three gating conditions that we consider are where the upper 32-bits are disabled for every operation, where the upper 48-bits are disabled for every operation, and where the upper bits are disabled selectively (either 32- or 48-bits disabled) based on the inputs (for the *ijpeg* stream).

Table 5.3 shows the results from this circuit-level analysis. These results show the decrease in power dissipation relative to the unmodified 64-bit adder. With randomized inputs, disabling half of the adder reduced the power dissipation by slightly less than half. Disabling three-fourths of the adder reduced the power by 62%. This

reflects the fact that there is some additional logic that is not disabled so the power dissipation does not decrease directly proportionally to the amount of bits being disabled.

The second column of results in Table 5.3 demonstrates the reduction in power dissipation when the input stream of zeros is sent to this adder. Because of the dynamic logic in this adder, there is significant power dissipation under this input stream.

The last column in the table shows the power dissipation under the stream of arithmetic operations captured from the *ijpeg* application. The row labeled selective gating shows the results assuming that a value gating signal exists which disables either the upper 32-bits or upper 48-bits depending on the actual values in the stream. With this scheme, selective gating saves roughly half of the power of this arithmetic unit.

Finally, we performed a power and delay analysis of zero detection circuitry which would be required to generate the value based clock gating signals. The power dissipation of a 64-bit zero-detect operation was less than 1% of the power dissipation of the 64-bit adder, suggesting that the power overhead of generating the gating signals is small.

Figure 5.22 shows the major pipestages in the microprocessor which would be affected by value based clock gating. We consider a possible implementation for the value-based clock gating signals. In this implementation, 48-bit zero detection logic is inserted after the values are fetched from the register file. This location was selected because there is some slack time available while waiting for the bypass values to feed into the value-select mux that writes into the execute stage pipeline registers. This allows value-gating signals to be generated for values that are fetched directly from the register file.

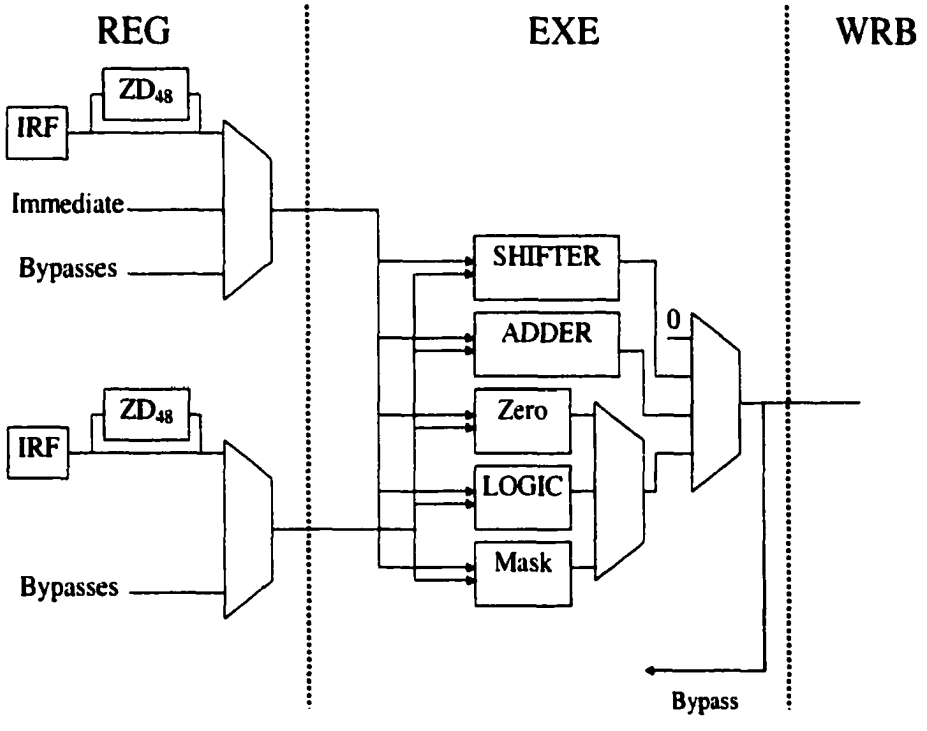

Figure 5.22: Pipeline diagram showing proposed changes

For values that come from the time critical bypass path, two solutions were considered. First, the gating signal could be generated within the flag generation circuitry which is already performed with the ALU after the value is computed. Second, a small optimization can be used to simplify the generation of the gating signal for these bypass values. We recognize that if the two source inputs to the adder are 16-bits or less, the results of the arithmetic operation will be 16-bits or less if there is no carry-out from the 16-bit stage of the adder. Using this optimization, a gating signal can be generated for bypass values with 1 gate delay following the 16-bit carry-out signal of the adder which can be performed before the final 64-bit value is computed.

We have proposed an implementation of value-based clock gating which does not affect any critical timing paths in a high-performance commercial microprocessor. This implementation can save roughly 50% of the power within the arithmetic units. This analysis suggests that this could be a viable point-optimization within commercial, high-performance microprocessors. Next we will consider possible methods to extend this technique to reduce power by performing value-based gating within register files and other memory structures.

5.5.3 Pervasive Value Gating: Wordline Disable

The optimizations that discussed so far have primarily focused on utilizing bitwidth information to reduce power within the integer functional units. A clear extension would be to save power on narrow-width values within the memory hierarchy. Potentially there would be opportunity for more power savings with this approach because a large fraction of the overall chip power is dissipated within memory structures.

Our proposed optimization within the memory hierarchy focuses on selectively disabling wordlines at the narrow-bitwidth boundaries. Figure 5.23 shows a diagram of how gating bits could be used to save power by disabling the wordline at the 16-bit boundary. This mechanism saves power because the upper 48 bitlines are not

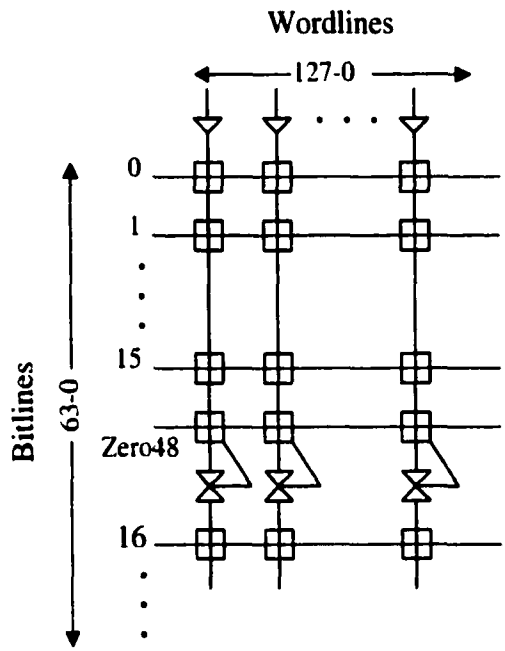

Figure 5.23: Wordline disable scheme in a register file

activated on read operations. A similar technique could be performed to save power on write operations by gating the driving circuitry for the write bitlines.

Power can be reduced farther by disabling the pre-charge circuitry for the unnecessary (upper 48) bitlines. This scheme may be more difficult to implement because the pre-charge logic is often asserted before the register id has been decoded. This requires that an additional control bit accompany the register id indicating that the upper 48-bits were all 0s.

Circuit level analysis was performed on a 128-entry, 64-bit register file to determine potential savings from various gating schemes. In this experiment, we compared schemes to disables the clock (pre-charge) circuits, the wordlines, and applying both simultaneously. These schemes were evaluated under the conditions where the register file values are initialized to all 1s and all 0s. Random values are then read and written to the register file for 100 cycles.

Integer Register File	Init to 0	Init to 1
Clock (pre-charge) disabled	31.1%	26.0%
Wordlines disabled	30.1%	45.1%
Both disabled	58.4%	66.0%

Table 5.4: Reduction in power dissipation of a 128-entry, 64-bit register file under various gating schemes (includes bitline, wordline, and precharge power)

Table 5.4 shows the reduction in power dissipation under the three gating conditions. There is 30%-45% reduction in power dissipation when using the different gating techniques alone, but a 58-66% reduction when they are applied together. Still, this is somewhat less than the ideal 75% reduction corresponding to the ideal scenario where power reduction is directly proportional to the reduction in bits that are fetched from the register files.

Techniques to disable the wordlines and pre-charge circuitry in register files and caches can provide substantial savings in power dissipation throughout the processor, because these memory structures consume a large fraction of the overall chip power dissipation. Recently, other research efforts have also begun to explore value-based clock gating throughout the datapath and memory hierarchy [29; 95].

5.6 Chapter Summary

Recently there has been increased interest in supporting operations with operand widths smaller than the maximum supported by functional units in microprocessors. This interest stems first from the increasing use of multimedia applications, but also from the larger 64-bit word sizes on current microprocessors. Most of the past research in this area has focused on increasing performance by discerning instructions with narrow width operands at compile time and generating code that allows such computations to occur with sub-word parallelism. From this research we can draw

several conclusions.

- **Compiler bitwidth analysis:** Compile-time analysis of operand width is constrained by the fact that the operand range of instructions may vary over the course of a program run depending on the input data. In addition, the compiler must conservatively analyze all potential paths taken. Our work notes that certain uncommon paths may have markedly different operand size characteristics than the typical path through programs.
- **Dynamic bitwidth analysis:** In order to augment compile-time analysis, we present a technique to *dynamically* exploit narrow-width data. This technique reduces power in the integer execution unit with aggressive clock gating, after determining that the upper portion of functional unit is not needed. This results in a 45%-60% reduction in the integer unit's power consumption for the benchmarks that we studied. This equates to a 5%-10% full-chip power savings.
- **Avenues to exploit narrow-width values:** Value based clock gating and operation packing both rely on the same core mechanism to achieve their optimization; namely they recognize that the upper bits in the input operands are not needed to perform the computation. Another area offering opportunities for dynamic recognition of low-precision operations is in the memory and I/O hierarchy. These opportunities include: (1) *pin and bandwidth compression* by recognizing that multiple pieces of low-precision data can share the same I/O pins and on-chip wiring, and (2) *low power caches* which save power by writing 16-bits for the value and one signal bit indicating that the stored value is low precision rather than writing the full 64-bits.

A key characteristic of our current proposal is that it requires only a small amount of hardware and no compiler intervention. Because of their common hardware requirements, we foresee systems in which the choice of whether to use the power or performance optimization can also be made dynamically, based on thermal input or other mode controls. More broadly, they represent a further step towards operand-value-based optimization strategies throughout processors.

Chapter 6

Dynamic Thermal Management

With the increasing clock rate and transistor count of today's microprocessors, power dissipation is becoming a critical component of system design complexity. The value-based clock gating technique proposed in Chapter 5 reduces the average power, or energy, of the CPU. However, in addition to energy reduction, we will need to begin to find solutions for thermal and power-delivery issues related to the maximum CPU power dissipation. These issues are becoming especially critical for very high-performance computing systems.

In this chapter, we investigate dynamic thermal management as a technique to control CPU power dissipation. With the increasing usage of clock gating techniques, the average power dissipation typically seen by common applications is becoming much less than the chip's rated maximum power dissipation. For example, while the Alpha 21264 processor is rated as having a maximum power dissipation of 95W when running "max-power" benchmarks, the average power dissipation was found to be only 72W for typical applications [41]. However, system designers still must design thermal heat sinks to withstand the worst-case scenario.

We define and investigate the major components of any dynamic thermal management scheme. Specifically we explore the tradeoffs between several mechanisms for responding to periods of thermal trauma and we consider the effects of hardware and

software implementations. With appropriate dynamic thermal management, the CPU can be designed for a much lower maximum power rating, with minimal performance impact for typical applications.

6.1 Motivation

The system complexity associated with increased power dissipation can be divided into two main areas. First, there is the cost and complexity of designing thermal packaging which can adequately cool the processor. It is estimated that after exceeding 35-40W, additional power dissipation increases the total cost per CPU chip by more than \$1/W [89]. The second major source of design complexity involves power delivery, specifically the on-chip decoupling capacitances required by the power distribution network.

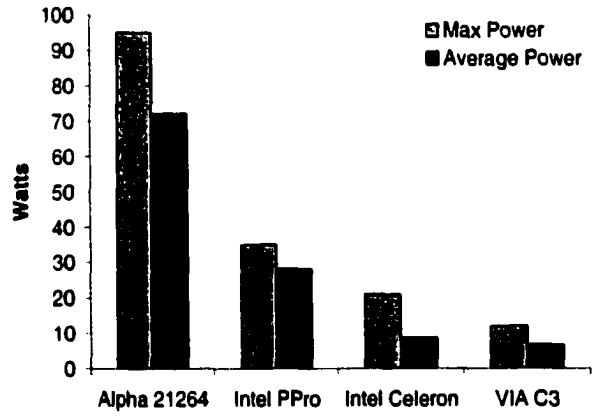

Figure 6.1: Average vs. Maximum power in several microprocessors.

Unfortunately, these cooling techniques must be designed to withstand the *maximum* possible power dissipation of the microprocessor, even if these cases rarely occur in typical applications. Figure 6.1 shows that the average power dissipation is often 30-50% less than the maximum rated chip power for many microprocessors.

The increased use of clock gating and other power management techniques that target average power dissipation will expand this gap even further in future processors. This disparity between the *maximum* possible power dissipation and the *typical* power dissipation suggests dynamic thermal management techniques to ensure that the processor does not reach these maximum power dissipation levels. That is, we seek to explore scenarios where the cooling apparatus is designed for a wattage less than the true maximum power, and dynamic CPU approaches guarantee that this designed-for level is never exceeded during a program run.

With many industrial designers predicting that power delivery and dissipation will be the primary limiters of performance and integration of future high-end processors, we feel that some form of dynamic thermal management will eventually be seen as a performance optimization, enabling larger chips to be built which would otherwise not be feasible [14; 41; 44; 89]. If die area and the number of transistors per chip become constrained by power density, techniques that can constrain the maximum possible power dissipation could allow designers to include more transistors per chip than would otherwise be possible, thus leading to increased performance.

In this work, we define and examine the generic mechanisms inherent in dynamic thermal management (DTM) schemes. Section 6.2 provides an overview and background on dynamic thermal management. We explore and compare the potential for hardware and software-based implementations of several dynamic thermal management schemes. Section 6.3 discusses the methodology used in the remainder of the chapter. We then break thermal management systems into three components: triggers, responses, and initiation policies, and discuss each of them in Sections 6.4, 6.5, and 6.6 respectively. The core of any DTM system is how it responds to a thermal emergency (e.g. frequency scaling, execution throttling, etc.). While this work provides data on a number of possible responses, we feel that further work may identify even more effective ones. Thus, Section 6.7 outlines a methodology for identifying

promising new response techniques by comparing power and performance correlations. Finally, Section 6.8 offers conclusions.

6.2 Dynamic Thermal Management: Overview and Strategies

This chapter explores policies and mechanisms for implementing dynamic thermal management in current and future high-end CPUs. As we use it, the term dynamic thermal management refers to a range of possible hardware and software strategies which work dynamically, at run-time, to control a chip's operating temperature. Traditionally, the packaging and fans for a CPU or computer system have been designed to maintain a safe operating temperature even when the chip was dissipating the maximum power possible for a sustained period of time, and therefore generating the highest amount of thermal energy. This worst-case thermal scenario is highly unlikely, however, and thus such worst-case packaging is often expensive overkill. DTM allows packaging engineers to design systems for a target sustained thermal value that is much closer to average-case for real benchmarks. If a particular workload operates above this point for sustained periods, a DTM response will work to reduce chip temperature. In essence, DTM allows designers to focus on average, rather than worst-case, thermal conditions in their designs. Until now, techniques developed to reduce *average* CPU power have garnered only moderate interest among the designers of high-end CPUs because thermal considerations, rather than battery life, were their primary concern. Therefore, in addition to reducing packaging costs, DTM improves the leverage of techniques such as clock gating designed to reduce average power [17; 89].

The key goals of DTM can be stated as follows: (i) to provide inexpensive hardware or software responses, (ii) that reliably reduce power, (iii) while impacting performance as little as possible. Voltage and frequency scaling are two methods for DTM that have been implemented in current chips [62; 92]. Unfortunately, little work has been done on quantifying the impact of voltage or frequency scaling on application performance. This work seeks to address this need, while also proposing other microarchitectural approaches for implementing DTM. We also propose a methodology based on performance and power correlations for seeking out new DTM responses.

6.2.1 Overview and Terminology

We are primarily concerned with reducing the maximum power dissipation of the processor. From a pure hardware point of view, the maximum power dissipation occurs when all of the structures within the processor are active with maximum switching activity. However, mutual exclusions in the underlying control structures make this scenario impossible. In reality, the maximum power dissipation is constrained by the software program that can maximize the usage and switching activity of the hardware. Special max-power benchmarks can be written to maximize the switching activity of the processor. These benchmarks are often quite esoteric, perform no meaningful computation, and dissipate higher power than "real" programs. Thus, DTM techniques could be used solely to target power levels seen in maximum power benchmarks and would rarely be invoked during the course of typical applications. In this work, we also consider more aggressive DTM designs which seek to further reduce the amount of cooling hardware necessary in machines. In Section 6.4 we discuss the tradeoffs between cooling hardware and performance loss in more detail.

Figure 6.2 offers a motivating example of how dynamic thermal management (DTM) can work. This figure plots chip temperature versus time (in cycles). In

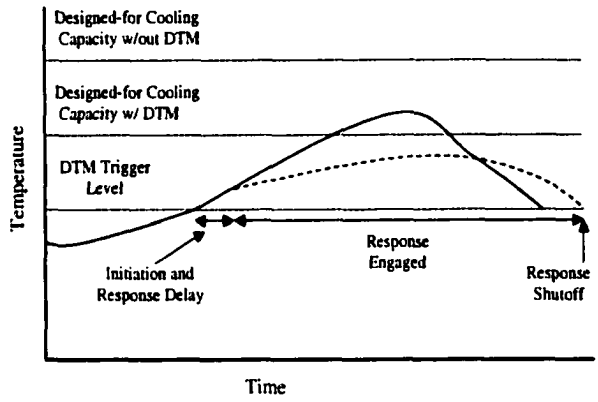

Figure 6.2: Overview of Dynamic Thermal Management (DTM) technique.

this figure, there are three horizontal dashed lines. The top-most line shows the designed-for cooling capacity of the machine without DTM. The second line shows that the cooling capacity could be reduced if dynamic techniques were implemented, because DTM reduces the effective maximum power dissipation of the machine. Finally, the lowest horizontal line shows the DTM trigger level. This is the temperature at which the DTM techniques are engaged.

Figure 6.2 has two curves which show chip temperature for some sequence of code being executed on the machine. The upper, solid curve is executed on the machine without DTM, and the lower, dotted curve is executed on a machine that has implemented DTM. Both curves are the same until the DTM trigger level is exceeded. At this point, after a small delay to engage the response, the curves diverge. In the uppermost curve the chip temperature slowly increases and then falls back below the trigger level. The lower curve shows how DTM would affect the same sequence of code. In this case, the DTM response is able to reduce the power dissipation and hence the chip temperature; the temperature never exceeds the designed-for cooling capacity. Eventually, the temperature decreases (as in the non-DTM curve), and the response is dis-engaged with some performance delay relative to the non-DTM curve.

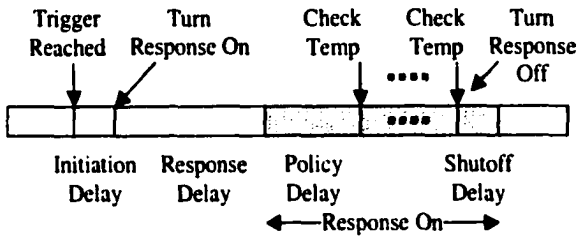

Figure 6.3: Mechanisms for Dynamic Thermal Management.

Figure 6.3 breaks down a DTM instance into several components. First, DTM is *triggered.* The triggering event may be a thermal sensor, a power estimator, or other gauge which indicates when DTM is needed. Once the trigger goes off, there is some *initiation delay* while, for example, an operating system interrupt and handler are invoked to interpret the triggering event. Once the handler has been executed, some DTM *response* begins. For example, possible responses include voltage or frequency scaling [72], or some of the microarchitectural ideas we discuss in later sections. Depending on the type of response chosen, there may be some delay inherent in invoking it; we refer to this time as *response delay.* Once the response is in effect, the next issue concerns when to turn it off. Turning the response off as soon as the temperature dips below the threshold may be unwise; temperature may fluctuate around the threshold and warrant keeping the response turned on. We use the term *policy delay* to refer to the number of cycles we wait before checking to see if the temperature has dipped below the triggering level. Finally, once the DTM system has determined that the response should be turned off, there is often a *shutoff delay* while, for example, the voltage or frequency is readjusted.

Implementing an effective DTM system, therefore, involves several key design choices which we consider throughout the remainder of this chapter:

- Selecting simple and effective triggers (Section 6.4),
- Identifying useful response mechanisms (Section 6.5),

- Developing policies for when to turn responses on and off (Section 6.6).

6.2.2 Background and Related Work

Some dynamic thermal management techniques have previously been explored. For example, the G3 and G4 PowerPC microprocessors from Motorola include a thermal temperature sensor in hardware and an interrupt capability to notify software of when a particular temperature has been reached [75; 78]. The processor also includes an instruction cache throttling mechanism that allows the processor's fetch bandwidth to be reduced when the CPU reaches a temperature limit.

The Transmeta Crusoe processor includes "LongRun" technology which dynamically adjusts CPU supply voltage and frequency to reduce power consumption [92]. While voltage and frequency tuning are quite effective at reducing power consumption since power scales linearly with clock frequency and with the square of the supply voltage, the delay in triggering these responses is necessarily higher than with microarchitectural techniques that are more localized. One of the goals of this work is to provide an overall view for the tradeoffs between initiation delay, response delay, and performance overhead for a number of techniques including both previously published techniques as well as ones newly-proposed in this thesis.

In addition to the fairly-recent Crusoe work, the ACPI (Advanced Configuration and Power Interface) specification likewise works to have hardware and software cooperate to manage power dynamically [1]. Unlike our work or those described above, ACPI is very coarse-grained. That is, power management in ACPI involves actions like turning on or off I/O devices or managing multiple batteries. Our work seeks to provide a much more fine-grained solution to thermal problems within the CPU itself. Recent work, including our own, has operated on different thrusts to explore this domain [18; 20; 46; 80]. These papers and our own work each focus on distinct

classes of processor architectures. Rohou and Smith have also considered using temperature feedback to guide the operating system in controlling CPU activity on a per-application basis [77].

Finally, we also note that the work by both Motorola and Transmeta is mainly geared toward improving battery life in portable machines. Our work, in contrast, has thermal packaging in high-end CPUs as its main thrust. This context is more performance sensitive than is power management for laptops. Our overall goal is to guarantee much lower worst-case power consumption, so that cheaper packaging can be used, with as little impact on performance as possible.

6.3 Methodology

We have used Wattch for the performance and power estimation results discussed in this chapter. Our results assume a model of a processor with the configuration parameters shown in Table 2.3. For technology parameters, we use the process parameters for a .35um process at 600MHz. We use Wattch's aggressive clock gating style for all results. This models power scaling which is linear with the number of active ports on any particular unit.

6.3.1 Power vs. Temperature

Wattch provides per-cycle power estimates, but one challenge in this research has been translating these power estimates into chip-level temperature variations. The most accurate approach would be to develop a model for the chip packaging and heat sink in a microprocessor. We are currently discussing such models with packaging engineers, but have abstracted them for the research presented here. We use the average power over a suitably large chunk of cycles (10k, 100k, and 1M) as a proxy for temperature [73]. The Tempest project at Intel seeks to eventually provide a

robust model to link power dissipation and chip temperature [33].

6.4 Dynamic Thermal Management: Trigger Mechanisms

Any dynamic response technique requires a trigger mechanism to engage the response during program execution. In this section, we consider two aspects of the trigger mechanism. First, we describe several possible trigger mechanisms for dynamic thermal management. Second, we discuss the rationale for determining an appropriate trigger limit to use in the DTM system. Sections 6.5 and 6.6 discuss the other key parts of the system: response techniques and initiation mechanisms.

6.4.1 Trigger Mechanisms

For our experimental setup we use an abstraction of chip temperature by using the moving average of power dissipation for the last 10,000 cycles of the processor's operation. This trigger mechanism is similar to an on-chip temperature sensor. We will discuss the details of the temperature sensor as well as several other trigger mechanisms that could be used as abstractions for temperature.

- **Temperature Sensors for Thermal Feedback**

  In the PowerPC dynamic thermal management system, thermal feedback from an on-chip temperature sensor is used as the trigger mechanism [78]. In the proposed scheme, the temperature sensor compares the junction temperature with a user programmable threshold. If the value is exceeded an interrupt is triggered allowing the operating system to invoke a response mechanism. This is the basic trigger mechanism that we evaluate in Section 6.5 with a variety of response mechanisms.

- **On-chip Activity Counters**

  Another possible source of information regarding the current chip temperature is through the use of activity monitors or on-chip performance counters [26]. These devices record "activity factors" for various structures within the processor and thus provide a gauge of how much work is being done and, correspondingly, the thermal state of the machine.

- **Dynamic profiling analysis**

  The runtime system of the machine can be responsible for determining when the application or user-behavior does not require the full resources of the computing system and then triggering a response. For example, operating systems often provide a wait process which is entered when there is no work to be performed, or address access information can be used to determine when the processor is idling [68].

  In addition, certain real-time and user-interactive applications inherently set certain acceptable performance levels. These types of applications would allow dynamic thermal management to occur when the specified rate is exceeded [39].

- **Compile-time trigger requirements**

  Static analysis at compile time can be used to estimate the performance of applications. In a similar manner, the compiler could estimate the high-power code segments and insert instructions specifying that DTM triggers should occur. In EPIC or VLIW where more of the parallelism is exposed by the compiler, this method would be more fruitful.

Comparing the viability of various trigger mechanisms is a topic for future research in this area. Relying exclusively on chip temperature sensors may have some drawbacks. First, the temperature sensor only approximates the average chip temperature;

multiple sensors may be needed on large chips. Second, there may be hysteresis between the temperature reading and the actual temperature. Pure hardware solutions also do not provide information about the workload; a combination of temperature sensors, activity counters, and software analysis may more effective than any of the techniques taken alone. For the results presented in this thesis, we use an abstracted trigger mechanism based on interrupts when modeled power reaches a a pre-set trigger threshold. This approximates the situation of a CPU with a single temperature sensor.

6.4.2 Thermal Trigger and Emergency Settings

The second decision that must be made within the trigger mechanism is the pre-set trigger threshold. We will define a "thermal trigger" to be the temperature threshold at which the trigger mechanism initiates the response mechanism to begin to cool the processor's temperature. A "thermal emergency" is a second temperature threshold set to a higher level and is used as a gauge of how successful the response mechanism was in dealing with the increase in temperature. Except where noted, in our simulation environment thermal triggers and emergencies occur if the moving average of full chip power dissipation for the past 10,000 cycles exceeds the pre-set trigger and emergency wattage values. Likewise there are also triggers that indicate the CPU has returned to a safe temperature. At these trigger points, the CPU can begin returning to normal operation.

In the next two sections we present analysis for the case where the response is triggered when the 10k moving average exceeds 24W and a full-fledged thermal emergency is considered to occur when the 10k moving average exceeds 25W. In Section 6.4, we consider the effects of varying the trigger level and the 10k cycle thermal window, but for the rest of the results we will use these values. Table 6.1 shows the percent of cycles that were above the thermal emergency threshold for the

Benchmark	Cycles in Emergency	Average Power
go	1.0%	22.7W
cc1	1.6%	21.6W
ijpeg	32.7%	24.3W
li	50.9%	24.8W
vortex	61.6%	24.6W
su2cor	70.5%	25.1W
tomcatv	96.1%	25.5W
fpppp	98.4%	32.9W

Table 6.1: Average Power and Percent of Cycles in Emergency for Simulated Processor

baseline system without DTM with the 24W trigger. There are three main categories of applications; the remainder of our charts will be sorted as follows:

- Mild Thermal Demands: The first two benchmarks have less than 10% of their cycles in thermal emergencies with average powers much less than the emergency level.
- Intensive Thermal Demands: The second group of four benchmarks ranges from 32% to 96%. *Tomcatv* fell into this class because its average power is only just above the emergency level.
- Extreme Thermal Demands: *Fpppp* is the extreme case in which 98% of the cycles exceeded the thermal threshold and the average power was 7W above the threshold.

We selected this trigger setting and this set of applications so we could observe the impact of DTM in a range of scenarios with varying thermal demands. We have neglected *compress* and *m88ksim* in this analysis because neither application had any cycles exceeding the chosen emergency point.

6.5 Dynamic Thermal Management: Response Mechanisms

In this section we consider the second basic mechanism within a dynamic thermal management architecture. The goal of designing a good DTM scheme is to reduce power with as small a performance loss as possible. A key part to realizing this goal is the response mechanism that throttles power dissipation in the system.

In this work, we consider five response mechanisms. Three of these are microarchitectural responses: I-cache-toggling, speculation control by restricting the number of unresolved branches, and decode bandwidth throttling (similar to Motorola's I-cache throttling [78]). We also consider clock frequency scaling and a combination of clock frequency scaling and voltage scaling.

- **Clock Frequency Scaling**

  Clock frequency scaling essentially trades a linear performance loss for a linear power savings. While in principle clock frequency scaling is trivial to implement, there may be delays incurred when changing clock rates. Furthermore, communicating with synchronous devices on the system bus may become more complicated.

- **Voltage and Frequency Scaling**

  Transmeta's LongRun technology performs dynamic clock frequency scaling along with dynamic voltage scaling to reduce power dissipation when necessary [92]. Obviously this requires detailed timing analysis and careful attention to circuit design choices [23]. Furthermore, as future process technologies scale to lower base supply voltages, dynamic voltage scaling may become more difficult. This is especially true when standby leakage currents become important. Leakage currents are directly related to the supply voltage; lowering the supply voltage to dynamically reduce dynamic power would have a corresponding

increase in the standby leakage current.

- **Decode Throttling**

  The PowerPC G3 microprocessor uses a micro-architectural level dynamic thermal management technique called instruction cache throttling to restrict the flow of instructions to the processor core [78]. This scheme relies on clock gating to reduce power dissipation as the flow of instructions is restricted. As motivation for selecting I-cache throttling instead of clock frequency scaling, the authors cite the difficulty in implementing dynamic clock control for the on-chip PLL as well as the fact the chip's L2 cache interface operates at a different clock rate from the chip's core.

- **Speculation Control**

  Speculation control is similar to Manne's work on speculative pipeline gating based on branch confidence estimation [42]. However, with the method proposed here, instead of basing the speculation control on branch confidence as in [42], we arbitrarily restrict the amount of speculation in the pipeline whenever a thermal trigger level is reached. To implement this, a counter is incremented whenever a branch is decoded and decremented whenever a branch resolves. If the counter exceeds a software-set limit, the decode stage stalls until enough branches have been resolved. The infrastructure for restricting the number of resolved branches is most likely already in place in most processors, since they limit the number of branches in the pipeline to restrict the additional state required for each active branch.

- **I-cache Toggling**

  We also propose a microarchitectural response technique called *I-cache toggling*. This response involves disabling the instruction fetch unit (I-cache and branch prediction) and using the instruction fetch queue to feed the pipeline. The fetch unit can be disabled every cycle, every other cycle, or at any specified interval

as specified by the interrupt call.

Obviously other techniques, or combinations of techniques, could be used as the response mechanism. In Section 6.7, we discuss a systematic methodology for determining new response techniques.

Both the trigger and the various response mechanisms that have been discussed could be programmable, allowing system designers to specify thermal management levels based on the amount of heat-sink technology in the system. For example, more expensive high-end server systems could have higher trigger limits and allow more unresolved branches, while cheaper low-end desktop systems would have lower trigger limits corresponding to their smaller heat-sinks. In addition, the individual response mechanisms allow a variation in the amount of throttling to be performed.

6.5.1 Response Mechanism Results

We use two metrics to evaluate the DTM schemes. First, the scheme should reduce the number of cycles in which the processor's temperature exceeds the thermal emergency threshold. The second metric that we use is the overall performance loss that the DTM technique incurs. Since the schemes we evaluate rely on microarchitectural as well as frequency scaling techniques, we consider total execution time as our performance metric.

We present analysis for the case where the response is triggered when the 10k moving average exceeds 24W and a full-fledged thermal emergency is considered to occur when the 10k moving average exceeds 25W. We also assume here that the various responses are initiated by a 250-cycle interrupt from the operating system in a manner similar to that of the PowerPC, but in Section 6.6 we consider additional hardware support which improves the performance of thermal management by eliminating this operating system overhead.

For the response that includes voltage scaling we assume a 10 microsecond delay to switch frequencies and a 20 microsecond response delay to switch voltages. During this delay the processor is stalled; this is consistent with the delay that Transmeta reports when switching between frequency and voltage states [60]. For the scaling techniques we set the policy delay to be 15 microseconds; in Section 6.6 we consider extending this delay to reduce the performance overhead of initiating the response. Finally, we assume that the processor voltage scales proportionally to what is reported in [60] for each frequency level. For example, when we scale frequency down by 10%, voltage is scaled down by 4.2% for the combined voltage and frequency scaling technique.

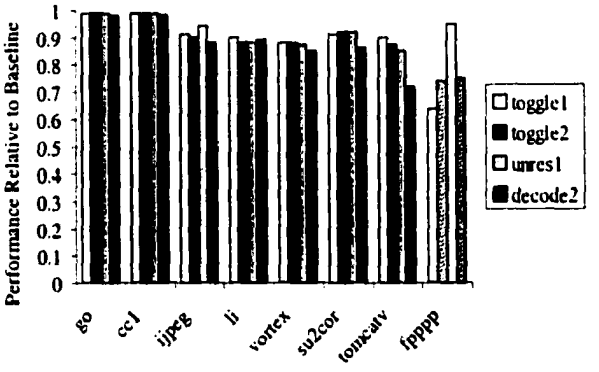

Figure 6.4: Reduction in performance compared to baseline for microarchitecture techniques.

Figure 6.4 shows the overall program performance reduction from the baseline for the microarchitectural techniques. Figure 6.5 shows the same results for the frequency/voltage scaling based techniques. Within these figures the first two sets of bars correspond to the benchmarks with mild thermal demands. The next five bars have intensive thermal demands. Finally, we show the *fpppp*, the extreme case benchmark.

Within Figure 6.4 there are four bars for each benchmark. The first two bars

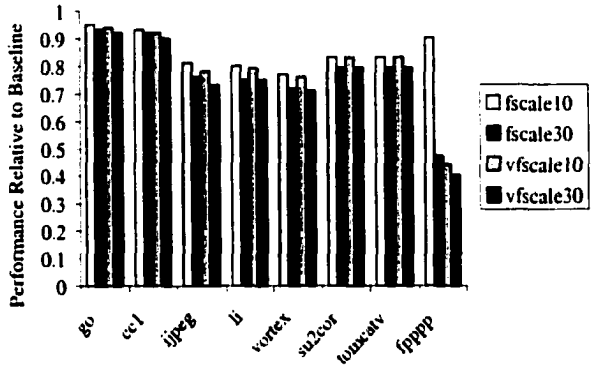

Figure 6.5: Reduction in performance compared to baseline for frequency/voltage scaling techniques.

correspond to I-cache toggling; *toggle1* is the case where the I-cache is disabled every cycle during the response, *toggle2* corresponds to the case in which it is disabled every other cycle. The third bar labeled *unres1* indicates that the maximum number of unresolved branches allowed in the pipeline is restricted to one before the decode stage is stalled. The final bar *decode2* indicates that the decode bandwidth is reduced by two instructions per cycle respectively. (We have considered additional settings for the above parameters, but to save space, we have selected the parameters that performed the best.) Within both Figure 6.4 and 6.5, bars which are cross-hatched (for example, *fpppp*'s *toggle2*, *unres1*, and *decode2* bars) indicate that the thermal emergencies were not entirely reduced for this configuration. Figure 6.5 also has four bars per benchmark. The first two bars correspond to scaling down the frequency by 30% and 10%. The last two bars correspond to scaling both the frequency and the voltage by 30% and 10%.

For many of the benchmarks, all of the techniques were able to entirely eliminate the thermal emergencies in the machine at this trigger level. DTM was not successful in entirely removing thermal emergencies with *ijpeg* with the *unres1* technique and *fpppp* with three of the microarchitectural techniques and *fscale10*.

For the benchmarks with mild thermal demands, the microarchitectural level techniques incurred an average performance penalty was 2%; the voltage and frequency scaling techniques had a 7% drop. For the benchmarks with intensive thermal demands, the reduction in thermal emergencies incurred a 12% performance penalty for the microarchitectural techniques and a 22% performance penalty for the scaling techniques. Only *toggle1*, *fscale30%*, *vfscale10*, and *vfscale30* were effective at reducing the number of thermal emergencies with *fpppp*; this came at over a 35% performance penalty.

Clearly the performance degradation of DTM at this trigger/emergency level is significant for the applications with large thermal demands. The performance degradation from these techniques can be broken down into two components. The first component is the performance drop due to invocation of the techniques. This includes the overhead of the operating system interrupt calls and the time needed to dynamically adjust the frequency and voltage scaling of the system. The second component is the IPC drop of the microarchitectural techniques or the frequency degradation penalty of the scaling techniques. For example with *su2cor* and the *unres1* trigger, 26% of the performance degradation was due to the interrupt overhead to engage and disengage the trigger. The remainder of the performance drop was the IPC degradation due to restricting the number of unresolved branches. As expected, the trigger overhead with frequency and voltage scaling techniques is much higher; over 70% of the performance loss is incurred due to the interrupt calls and overhead in adjusting the clock rate with frequency scaling and over 75% with combined voltage and frequency scaling.

These trends tend to hold across the benchmarks and across the different styles of responses. There are two major reasons for the larger invocation overhead of the frequency and voltage scaling techniques. First, the overhead of frequency and voltage scaling is significantly higher than that of the microarchitectural techniques. Second,

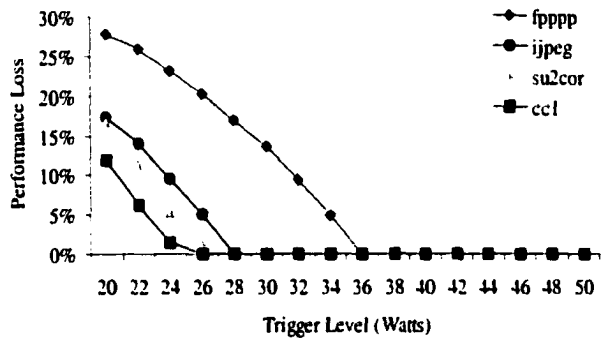

Figure 6.6: Performance Loss at various trigger levels. Higher trigger levels (30-50 Watts) offer less packaging savings, but have nearly zero performance impact. More aggressive trigger settings (25-30 Watts) begin to show modest performance impact. 50W is the max power for the modeled chip.

because of variations in application behavior which cause changes in the thermal behavior of the system these policies may be enabled or disabled many times during program execution. This is especially true when DTM mechanisms are in place to regulate temperature. Obviously for these applications, the large invocation overhead is magnified. In the next section we consider additional hardware and other techniques that can reduce the performance overhead of trigger engagement. The results also show that there is room for application specific selection of response techniques; certain response techniques perform much better than others for individual benchmarks.

6.5.2 Thermal Trigger and Emergency Settings

In Figure 6.6 we consider an idealized version of the *ufscale30* policy that has no initiation delay. This figure shows the percent performance loss relative to the total execution time of the baseline system for DTM while varying the thermal trigger settings ranging from 20-34W. For example, *fpppp* runs 27% slower with a trigger of 20W than it does with no DTM, but its max power without DTM exceeds 40W

in some cases. This performance penalty is incurred by the response mechanism; in this case the response is a version of frequency scaling. When the trigger is set at a conservative range (above 30W for these benchmarks), most of the benchmarks see very little performance degradation. Even with the most conservative approach, dynamic thermal management allows the chip's maximum power rating to be reduced considerably. In this design, the maximum power was around 50W; with DTM this could be easily reduced to 35-40W.

A more aggressive design would set the trigger somewhere around 25W for these applications. Being more aggressive in the trigger setting allows for more significant packaging savings, about $1 per watt per CPU chip. But this savings may come at the price of reduced performance for some applications. Thus, a key goal of this work is to propose streamlined mechanisms for DTM that offer the best possible performance.

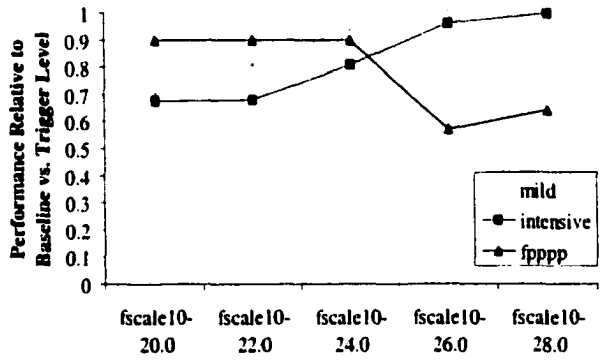

Figure 6.7: Performance Loss at various trigger levels for the *fscale10* response technique.

Now we consider the effect of the trigger value with our standard (including all delays) *fscale10* technique. Figures 6.7 and 6.8 show the effects of varying the trigger level for the *fscale10* technique. Each data point shows the performance and number of thermal emergencies relative to the baseline configuration without DTM at the specified trigger level; the level that we consider to be an emergency is always set to

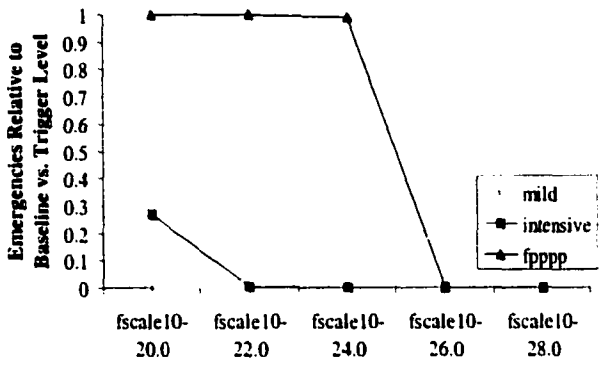

Figure 6.8: Emergency Reduction at various trigger levels for the *fscale10* response technique.

be 1W above the trigger level. From Figure 6.7 we can see that for the set of mild and intensive benchmarks, performance degrades further as we set more aggressive trigger levels. However, from Figure 6.8 we see that the machine does not exceed the thermal emergency threshold until we reach a trigger of 20W. *Fpppp* performs quite differently. At trigger levels between 20-24W, the number of thermal emergencies has not been reduced at all; the *fscale10* policy is continuously engaged leading to a constant 10% performance penalty. However, at 26W and 28W the *fscale10* policy begins to be effective. At the 26W trigger level there is a corresponding drop in performance as we start to see the effect of the trigger being engaged and disengaged during execution. At 28W and upwards, this performance penalty diminishes.

We have seen similar patterns with the other voltage and frequency scaling techniques as well as with the microarchitectural techniques. Overall, the choice of the trigger level is an important lever for system designers to use when deciding whether to trade off performance for some of the most extreme benchmarks such as *fpppp* against the amount of cooling hardware to build into the system.

6.6 Dynamic Thermal Management: Initiation Mechanisms

In Section 6.5 we consider a variety of dynamic response mechanisms. In that section, we assume an implementation where the operating system calls an interrupt handler to invoke the dynamic response mechanism which incurs significant overhead. To mitigate this overhead, we consider two modifications to the initiation mechanism of DTM. First, we consider additional hardware support in the microarchitecture to remove the interrupt overhead. Second, we modified the policy delay to allow the response mechanism to remain engaged for longer periods of time, better amortizing the cost of the trigger's response delay over the program run.

6.6.1 Hardware Support for Initiating Responses

Eliminating interrupt call overhead is the obvious benefit from additional hardware support. However, avoiding interrupt handling also allows more fine-grained control of the response scheme. This reduces the performance overhead of DTM because the performance-limiting response will only be engaged when it is needed. Finally, more fine-grained control of the response mechanism could have a benefit on reducing the number of cycles with thermal emergencies, because the mechanism will be engaged faster.

To eliminate the trigger overhead, the trigger mechanism must be directly integrated into the microarchitecture. For example, the temperature sensor or hardware activity counter could generate a signal indicating that the trigger limit has been exceeded and this signal could be sent to microarchitectural state machines that would engage the trigger. The operating system would only need to program internal registers at the beginning of the application's execution to adjust the amount of throttling

that the response should use. To evaluate the effects of this additional hardware we have simulated the microarchitectural response techniques with a 0-cycle initiation delay; this assumes that the 250-cycle interrupt overhead can be removed.

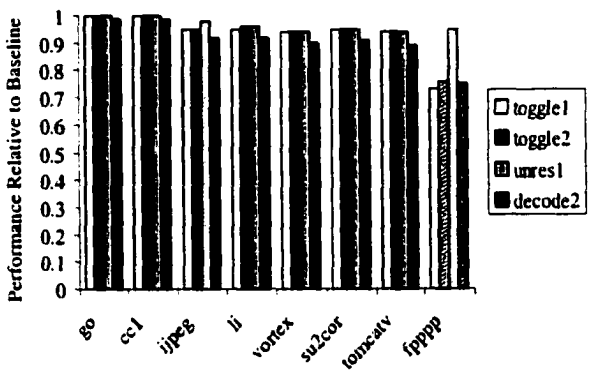

Figure 6.9: Reduction in performance compared to baseline.

Figure 6.9 shows the results for the microarchitectural response mechanisms assuming that the trigger mechanism is integrated into the microarchitecture. The reduction in number of thermal emergencies is unchanged. However, there is a reduction in performance penalty. For the mildly intensive four benchmarks, the performance penalty is on average 5%; this compares to a 7% performance hit without the hardware support. For the next group of four benchmarks with more intensive thermal demands, the performance reduction is 13% compared to 16% with OS overhead. Since *fpppp* spends a large amount of time with the triggers engaged, speeding up the interrupt overhead had a small effect on the performance using this scheme.

6.6.2 Policy and Thermal Window Effects on Voltage/Frequency Scaling

In the previous section, we considered the use of hardware support to reduce the overhead of initiating the response mechanism. This overhead is even larger for the

voltage and frequency scaling techniques. The majority of the time required to initiate these techniques is spent scaling the frequency and internal voltage of the processor to a new level. Since this overhead is not related to the operating system, reducing the interrupt time will only have a small effect on performance. In this section we consider two techniques to reduce this delay. First, we consider increasing the policy delay, or the amount of time that the mechanism is enabled before it is eligible to be disabled. Increasing the policy delay allows the response and shutoff overhead to be amortized over a larger portion of the run. On the other hand, if the policy delay is too long, the response will be engaged during unnecessary stretches of program execution. The second technique we consider is using a larger thermal window to estimate the temperature of the chip. For all of the previous results, we have used a window of 10K cycles. In this section, we consider increasing this window to be 100K cycles. This has the effect of smoothing out short thermal spikes which could unnecessarily cause the response to be triggered. For the more coarse-grained frequency and voltage scaling techniques, we would like to minimize these situations.

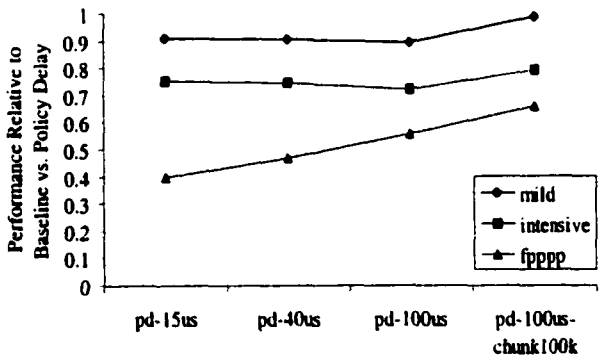

Figure 6.10: Reduction in performance compared to baseline.

We consider varying the policy delay with values of 15 microseconds, 40 microseconds, and 100 microseconds. We have chosen 40 microseconds because it is the combined response and shutoff delays of frequency+voltage scaling response mechanism.

Finally, we show the effect of increasing the thermal window from 10K cycles to 100K cycles with a policy delay of 100 microseconds. Figure 6.10 shows the the performance effect of the techniques when using the *vfscale30*. In this figure, the first three data points report the performance relative to the baseline while varying the policy delay; the final point shows the performance as the thermal window is increased to 100K cycles.

From this Figure 6.10 we see that there was very little effect on performance for the mild and intensive benchmark suite; in fact, there was a slight degradation in performance as we increase the policy delay. This is because although the initiation overhead was decreased, the amount of time spent with frequency and voltage scaling engaged increased. On the other hand, *fpppp* had a substantial performance improvement with increased policy delay. For this benchmark, the performance loss to the baseline decreased from 60% with 15 microsecond policy delay to 44% with 100 microsecond policy delay. Finally, we see that increasing the thermal window had a positive effect on all three classes of applications. When moving from the 10K cycle window to the 100K cycle window the performance loss decreased to 34% for *fpppp*.

For the benchmarks with intensive thermal demands, the performance loss decreased to 20%. On the other hand, we found that increasing the size of the thermal window had a much smaller (1-2%) performance benefit for the microarchitectural techniques. Since these techniques are much more fine-grained in nature, they suffer less from short thermal transients.

We have found that the initiation mechanism is a key factor to the performance degradation of DTM. We have investigated two techniques which show promise for reducing the performance overhead. Future work could address additional techniques to reduce this overhead either through more efficient methods to initiate the responses or smarter techniques to enable and disable responses.

6.7 Method for Identifying DTM Responses

In this work we have compared the benefits of dynamic thermal management via several microarchitectural techniques as well as clock frequency and voltage scaling. In considering other schemes for thermal management, we would like to develop a more systematic approach to identifying potential techniques.

We propose here a method based on correlation. That is, we wish to find levers that reduce power with a less-than-proportional reduction in performance. We have performed simulations using Wattch to correlate power dissipation with other processor statistics such as instruction fetch rate, branch prediction accuracy, data and instruction cache hit rates, execution bandwidth, and IPC. We use this method to isolate certain processor statistics that track more closely with power than with IPC.

Correlation	Fetch Rate	BPred Rate	DC Hit Rate	IC Hit Rate	Exec BW
Power vs.	0.82	0.37	0.40	0.50	0.83
IPC vs.	0.77	0.61	0.25	0.48	0.81
Difference	0.05	-0.24	0.15	0.02	0.02

Table 6.2: Correlation Data for Average of Benchmarks

We collected the average power and performance statistical data for fixed chunks of 10,000 cycles. These statistics were then correlated with each other after the simulation completed. An example of the correlation data for the average of our benchmark suite is shown in Table 6.2. The first line of this table shows the correlation between processor power dissipation and instruction fetch rate (avg. number of instructions fetched per cycle), branch prediction accuracy, cache hit rates, and execution bandwidth (committed + mis-speculated instructions/cycle). As expected,

power correlates very strongly with execution bandwidth. Instruction fetch bandwidth correlates also correlates strongly with power. Branch direction prediction accuracy, a secondary indicator of application performance, also correlates with power but to a lesser degree. Data and instruction-cache hit rates correlate slightly more than branch predictor accuracy with power.

The second line of Table 6.2 shows the correlation between IPC and the processor statistics. From this table, we see execution bandwidth, branch predictor accuracy, and fetch bandwidth correlate the most with performance.

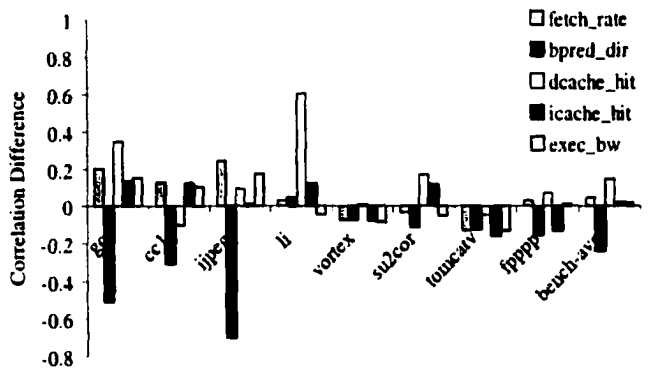

Figure 6.11: Correlation between power and several performance statistics.

Figure 6.11 plots power correlation minus IPC correlation for each point for the individual benchmarks. Thus, a positive data point in this graph corresponds to a case where power dissipation is more strongly correlated with the metric (eg fetch rate) than IPC is. Looking for possible DTM responses with strong power correlations lets us seek out "wasted work" that may lead to good power reductions with minimal performance impact. This may reveal strategies that would be most useful for dynamic thermal management.

This data reveals some interesting trends. For example, for almost all of the benchmarks, branch predictor accuracy correlated much more with performance than

with power. On other other hand, cache hit rates and instruction fetch bandwidth correlated more with power than with IPC for many of the benchmarks. Execution bandwidth correlates more with power than with IPC for four of the benchmarks. This lends support to our decision to evaluate I-cache toggling and speculation control as methods for dynamic thermal management. We plan future work that will broaden the types of microarchitectural response mechanisms that we investigate with correlation analysis.

6.8 Chapter Summary

We have proposed and evaluated the benefits of using dynamic thermal management to reduce the cooling system costs of CPUs. From this initial research effort, we have drawn several conclusions which we feel can help guide future research in this area.

- **Trigger Selection:** Dynamic thermal management allows arbitrary tradeoffs between performance and savings in cooling hardware. Conservative target selections can still lead to significant cost improvements with essentially zero performance impact, because the trigger point is rarely reached for many applications.
- **Designers Can Focus on Average Power:** In addition, DTM makes other techniques targeting average power more interesting to the designers of high-end CPUs. Effective DTM makes average power the metric of interest even for high-end CPU designers, since packages need no longer be designed for worst-case power. With DTM, lowering average CPU power will reduce the trigger value needed for a particular level of performance, and thus will reduce packaging costs.
- **Trigger Activation Time is Significant:** Not unexpectedly, the triggering delay is a key factor in the performance overhead of DTM. We have found that

more fine-grained control of the trigger mechanism is especially important in the context that we consider: reducing thermal traumas in high-performance CPUs. Unfortunately, our data show that some of the most promising techniques in DTM today, such as voltage or frequency scaling, are typically implemented with very high activation delays. These lead to significant performance overheads across most applications.

- **Lightweight Policies Are Effective:** More lightweight, fine-grained policies, such as the microarchitectural techniques we have discussed, often allow the temperature to stay close to the target level with a small performance penalty. In addition, the fine-grained policies are less affected by rapid fluctuations in the temperature.
- **Methodology for Identification of Future Techniques:** Because of these growing opportunities for microarchitectural DTM techniques, we have also proposed a methodology for evaluating new DTM approaches. This mechanism correlates power and performance, and looks for "low-hanging fruit"; that is, our correlators look for techniques that can cut power by significantly more than they hurt performance. Identifying these sorts of wasted work, particularly on an application-specific basis, appears to be a promising way of discovering new microarchitectural DTM techniques in the future.

Chapter 7

Conclusions

This thesis has explored modeling and architectural techniques for high-performance, power-efficient microprocessors. Establishing good modeling infrastructures to allow us to develop new high-level techniques for power-efficient design is crucial to the development of our next generation computing systems.

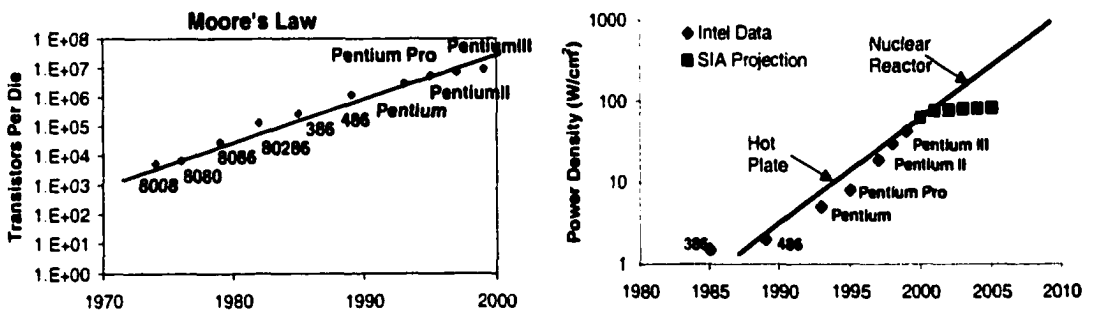

Figure 7.1: Moore's Law of increasing transistors and corresponding increases in Power Density.

Figure 7.1 illustrates the trends that we will need to overcome in the future. This figure shows the exponential growth in transistors per die which has come to be known as Moore's Law. This growth has been sustained for the past thirty years and is projected to continue for at least the next 10-15 years. This transistor growth has for the most part directly contributed to the growth in computing performance.

The second graph in this figure shows the power density for the same selection of microprocessors. Unfortunately, the exponential growth in transistors per die has also led to an exponential growth in power density. We are currently reaching the limits of cost-effective packaging technology to build chips with these high power densities. Because of this, the Semiconductor Industry Association (SIA) projects that power density will begin to flatten out over the next several years [79]. This projection is shown as squares in Figure 7.1. The challenge that we have as chip designers is to continue to extract performance from our ever increasing number of transistors, while restricting the corresponding increases in power consumption.

7.1 Contributions

This thesis has had two major thrusts focusing on the power problem. First, we have developed a methodology for estimating power dissipation within traditional architectural performance simulators. We developed two tools with this methodology: Wattch and PowerTimer. The energy models within Wattch are very useful for early stage design experiments with a microarchitecture that does not have existing power data to scale from. On the other hand, PowerTimer is less flexible, but more useful in making projections to chips that are follow-on products to an existing architecture. Research in validating the accuracy of these models is critical to establish confidence that our estimates can be useful in practice. We have performed validation at many different levels including a detailed analysis of the robustness of these models under many different error conditions.

The second major focus of this thesis has been the development of techniques to reduce power dissipation and thermal issues in high-performance microprocessors. Value-based clock gating proves to be a useful point-optimization which can significantly reduce power dissipation in the functional units as well as in the memory

hierarchy. This optimization has been explored in detail within the context of a real commercial high-performance microprocessor. Our results show that this technique leads to a 50% reduction in the power dissipation of integer units.

Dynamic thermal management is a technique that addresses thermal issues related to the maximum power dissipation of high-performance microprocessors. This technique seeks to dynamically throttle processor resources for sections of application behavior that exhibit very high power and heat dissipation. This throttling allows the processors to be designed with a heat solution for something approaching that of the average power dissipation rather than the worst case power dissipation. We have shown that dynamic thermal management can reduce the effective processor wattage by about 30% with minimal performance loss for most applications.

The approaches described in this thesis demonstrate an initial step towards power modeling at the architectural level and presents two key architectural level power optimizations. These techniques, and similar ones, have begun to attract interest within industrial research and design groups. However, in the future we will need to continue to develop even more robust, flexible, and fast architectural power models and subsequently generate and evaluate ideas to reduce power and thermal issues.

7.2 Future Directions

The field of architectural level power-efficient modeling and design is in its infancy. Researchers in this field will need to focus on both modeling and the development of new architectural ideas for power efficient design. This is a very fruitful area for future research, not only because the area is relatively new, but because there are many challenges to overcome. In the next few sections I will discuss some of the major challenges that will need to be overcome in the areas of power modeling and techniques for low power design.

- **Modeling Different Design Styles:** Wattch and PowerTimer were both written to model very high-performance microprocessors with custom design styles. These classes of processors tend to use circuit design styles which focus on high-performance, with power as a secondary consideration. Because of the focus on this particular design style, the models developed may not be applicable to other classes of architectures such as low-power, embedded microprocessors. However, the methodology presented will certainly apply to other design styles. Ideally, our architectural toolsets would allow the user to choose design styles appropriate to the chip under development. For example, very high-performance, but also less power efficient structures could be chosen for high-end processors, and low-power, but slower, structures could be chosen for embedded processors.
- **Chip and System Floorplans:** Architectural level power estimation tools could take into account chip floorplans estimates which would allow more accurate models for interconnection power to be developed. A major area to be explored is to look at system-level power modeling for both on-chip and off-chip interconnection networks. This area will be especially important in the future as many research projects have been looking at multiple cores on a chip [9; 45].
- **Thermal and Packaging Models:** In the future, techniques that seek to reduce temperature of the processor die will need to focus on both the chip floorplan, to identify local hotspots, and the chip package and heatsink, to estimate how these local hotspots relate to each other. While detailed packaging models do exist, these models are not well-suited to an architectural level tool. Abstractions will need to be developed to provide the salient details of these models to architects. Packaging models focusing on the chip pins will become important as we focus on reducing the amount of di/dt noise in microprocessors. Di/dt noise is due to large swings in the power dissipation of the chip on a cycle

to cycle basis. These large swings can cause large disturbances in the values of the power and ground nodes, causing the chip to malfunction.

- **Leakage Estimates:** Leakage energy, or static power dissipation, is expected to grow at approximately 5x per generation unless major steps are taken to reduce it. Even if known techniques are applied to reduce leakage energy it will still become a large fraction of overall chip power dissipation within a few generations. Architects are beginning to propose models [25] and architectural techniques [51; 97] to reduce leakage energy. Because leakage is very temperature dependent, we may need to couple leakage models with chip thermal models.
- **Higher Level Power Estimates:** Another important area for power modeling is to focus on pushing power estimates to higher levels in the system such as the compiler and system software (OS). Estimating power in the compiler could allow for power-aware instruction scheduling to reduce energy or to provide a smooth flow of instructions so as to reduce di/dt noise. Providing hooks within the operating system to monitor the power dissipation of running processes would permit the system software to enforce power and energy budgets to processes. The OS could also play an active role in techniques like dynamic thermal management by monitoring and reacting to thermal emergencies in the machine. This is an area that has begun to receive some attention with researchers proposing that the OS sample on-chip performance counters to provide power estimates [10; 49].
- **Dynamic Program Behavior:** Many of the power-efficient architectural techniques exploit some form of dynamic behavior to reduce power with a minimal effect on performance. These techniques are exploiting the fact that general purpose architectures will inherently be inefficient for certain applications or

phases of applications. This will be a key area for finding additional optimizations to improve both performance and power. Software infrastructures are currently being developed to support these types of optimizations [8; 82].

7.3 Summary

Power dissipation is a first-order design constraint in nearly all types of computing systems. Chip designers will need to develop techniques at all levels of the design hierarchy to meet the power challenges that we will have to face in building the next few generations of microprocessors. This thesis has demonstrated a methodology for estimating power at the architectural level and has shown how architectural techniques can be effective in reducing energy and thermal issues in high-performance microprocessors. In the future, we will need to continue to develop models and additional techniques to cope with power dissipation.

BIBLIOGRAPHY

[1] Advanced Configuration and Power Interface. http://www.teleport.com/ acpi/.

[2] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-based sequential logic optimization for low power. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):426–36, December 1994.

[3] B. Alpern, L. Carter, and K. Gatlin. Microparallelism and high-performance protein matching. In *Proceedings of SuperComputing 95*, 1995.

[4] B. Amrutur and M. Horowitz. Speed and power scaling of sram's. *IEEE Journal of Solid-State Circuits*, 35(2):175–185, 2000.

[5] K. Asanovic, B. Kingsbury, B. Irissou, J. Beck, and J. Wawrzynek. T0: A single-chip vector microprocessor with reconfigurable pipelines. In *Proceedings of the 22nd European Solid-State Circuits Conference*, 1996.

[6] M. Azam, P. Franzon, W. Liu, and T. Conte. Low power data processing by elimination of redundant computations. In *Proc. of Int'l Symposium on Low-Power Electronics and Design*, 1997.

[7] R. I. Bahar, G. Albera, and S. Manne. Power and performance tradeoffs using various caching strategies. In *Proc. of Int'l Symposium on Low-Power Electronics and Design*, 1998.

[8] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: a transparent dynamic optimization system. In *ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI)*, June 2000.

[9] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture: June 12–14, 2000, Vancouver, British Columbia*, 2000.

[10] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proc. of the 9th ACM SIGOPS European Workshop*, 2000.

[11] D. Bhandarkar. *Alpha Implementations and Architecture - Complete Reference and Guide.* Digital Press, 1996.

[12] B. Bishop, T. Kelliher, and M. Irwin. The Design of a Register Renaming Unit. In *Proc. of Great Lakes Symposium on VLSI*, 1999.

[13] M. Borah, R. Owens, and M. Irwin. Transistor sizing for low power CMOS circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):665–71, 1996.

[14] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, July-August 1999.

[15] W. J. Bowhill et al. Circuit Implementation of a 300-MHz 64-bit Second-generation CMOS Alpha CPU. *Digital Technical Journal*, 7(1):100–118, 1995.

[16] D. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoğlu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, Nov./Dec. 2000.

[17] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture (HPCA-5)*,

Jan. 1999.

[18] D. Brooks and M. Martonosi. Adaptive thermal management for high-performance microprocessors. In *Workshop on Complexity Effective Design 2000 at ISCA27*, June 2000.

[19] D. Brooks and M. Martonosi. Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance. *ACM Transactions on Computer Systems*, 18(2):89–126, May 2000.

[20] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA-7)*, Jan. 2001.

[21] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[22] D. Brooks, J.-D. Wellman, P. Bose, and M. Martonosi. Power-Performance Modeling and Tradeoff Analysis for a High-End Microprocessor. In *Workshop on Power Aware Computing Systems at the 9th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, Nov. 2000.

[23] T. Burd, T. Pering, A. Stratkos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *ISSCC Digest of Technical Papers*, pages 294–295, 2000.

[24] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *Computer Architecture News*, pages 13–25, June 1997.

[25] J. Butts and G. Sohi. A static power model for architects. In *Proceedings of the 33rd International Symposium on Microarchitecture (MICRO-33)*, Dec.

2000.

[26] C. Georgiou and S. Kirkpatrick and T. Larson. Variable Chip-clocking Mechanism. US Patent 5,189,314, 1993.

[27] G. Cai and C. Lim. Architectural level power/performance optimization and dynamic power estimation . In *Cool Chips Tutorial at MICRO-32*, Nov. 1999.

[28] T. Callaway and J. E.E. Swartzlander. Power-delay characteristics of CMOS multipliers. In *Proceedings of the 13th International Symposium on Computer Arithmetic*, July 1997.

[29] R. Canal, A. Gonzalez, and J. E. Smith. Very low power pipelines using significance compression. In *Proceedings of the 33rd International Symposium on Microarchitecture (MICRO-33)*, Dec. 2000.

[30] D. Citron, D. Feitelson, and L. Rudolph. Accelerating multi-media processing by implementing memoing in multiplication and division units. In *Proceedings of the 8th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, pages 252–261, Oct. 1998.

[31] T. Conte, K. Menezes, and S. Sathaye. A technique to determine power-efficient, high performance superscalar processors. In *Proceedings of the 28th Hawaii Int'l Conference on System Science*, 1995.

[32] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, November 2000.

[33] A. Dhodapkar, C. Lim, and G. Cai. TEM^{2}P^{2}EST: A Thermal Enabled Multi-Model Power/Performance ESTimator. In *Workshop on Power Aware Computing Systems at the 9th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, Nov. 2000.

[34] K. Diefendorff. Power4 focuses on memory bandwidth. *Microprocessor Report*, pages 11–17, Oct. 6, 1999.

[35] C. Dulong. The IA-64 architecture at work. *IEEE Computer*, 31(7):24–32, 1998.

[36] H. Fair and D. Bailey. Clocking Design and Analysis for a 600MHz Alpha Microprocessor. In *ISSCC Digest of Technical Papers*, pages 398–399, February 1998.

[37] F. Gabbay and A. Mendelson. Using value prediction to increase the power of speculative execution hardware. *ACM Transactions on Computer Systems*, Aug. 1998.

[38] G. Gerosa et al. A 2.2W, 80 MHz superscalar RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 29(12):1440–54, 1994.

[39] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity-Effective Design at the 27th Int'l Symposium on Computer Architecture (ISCA27)*, June 2000.

[40] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–84, 1996.

[41] M. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *35th Design Automation Conference*, 1998.

[42] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun. Confidence estimation for speculation control. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA-25)*, pages 122–31, June 1998.

[43] L. Gwennap. Intel's P6 uses decoupled superscalar design. *Microprocessor Report*, pages 9–15, Feb. 16, 1995.

[44] L. Gwennap. Power issues may limit future CPUs. *Microprocessor Report*, August 1996.

[45] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun. The Stanford Hydra CMP. *IEEE Micro*, 20(2):71–84, Mar./Apr. 2000. Presented at Hot Chips 11 Conference, Stanford University, Stanford, California, August 15–17, 1999.

[46] W. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd International Symposium on Microarchitecture (MICRO-33)*, Dec. 2000.

[47] IEEE Standards Board. IEEE Standards for Binary Floating-Point Arithmetic. Technical Report ANSI/IEEE Std. 754-1985, Institute of Electrical and Electronics Engineers, 1985.

[48] Q. Jacobson and J. Smith. Instruction pre-processing in trace processors. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture (HPCA-5)*, Jan. 1999.

[49] R. Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *Proc. of Int'l Symposium on Low-Power Electronics and Design*, 2001.

[50] M. B. Kamble and K. Ghose. Analytical Energy Dissipation Models for Low Power Caches. In *Proc. of Int'l Symposium on Low-Power Electronics and Design*, 1997.

[51] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th International Symposium on Computer Architecture (ISCA-28)*, June 2001.

[52] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, Nov. 1997.

[53] U. Ko, P. Balsara, and A. Nanda. Energy optimization of multilevel cache architectures for RISC and CISC processors. *IEEE Transactions on VLSI Systems*, 6(2):299–308, June 1998.

[54] H. Kojima, D. Gorny, K. Nitta, A. Shridhar, and K. Sasaki. Power analysis of a programmable DSP for architecture and program optimization. *IEICE Transactions on Electronics*, E79-C(12):1686–92, December 1996.

[55] K. Kundert. The designer's guide to spice and spectre, 1995.

[56] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communication systems. In *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, Dec. 1997.

[57] R. Lee. Subword parallelism with MAX-2. *IEEE Micro*, 16(4):51–59, Aug. 1996.

[58] M. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *Proceedings of the 7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 138–47, Oct. 1996.

[59] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA-25)*, pages 132–41, June 1998.

[60] Marc Fleischmann. Crusoe Power Management: Cutting x86 Operating Power Through LongRun. Embedded Processor Forum, June 2000.

[61] Mentor Graphics Corporation, 1999.

[62] J. Montanaro et al. A 160-MHz, 32-b, 0.5W CMOS RISC microprocessor. *Digital Technical Journal*, 9(2):49–62, 1996.

[63] C. Moore. The Power4 System Microarchitecture. *Microprocessor Forum*, Oct 2000.

[64] M. Moudgill, P. Bose, and J. Moreno. Validation of Turandot, a fast processor model for microarchitecture exploration. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 451–457, Feb. 1999.

[65] M. Moudgill, J. Wellman, and J. Moreno. Environment for PowerPC microarchitecture exploration. *IEEE Micro*, 19(3):9–14, May/June 1999.

[66] C. Nagendra, M. Irwin, and R. Owens. Area-time-power tradeoffs in parallel adders. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(10):689–702, October 1996.

[67] P. Ng, P. Balsara, and D. Steiss. Performance of CMOS differential circuits. *IEEE Journal of Solid-State Circuits*, 31(6):841–846, 1996.

[68] O. Ikeda. Power Saving Control System for a Computer System. US Patent 5,504,908, 1996.

[69] S. Palacharla, N. Jouppi, and J. Smith. Complexity-Effective Superscalar Processors. In *Proceedings of the 24th International Symposium on Computer Architecture (ISCA-24)*, 1997.

[70] S. Palacharla, N. Jouppi, and J. Smith. Quantifying the Complexity of Superscalar Processors. In *Univ. of Wisconsin Computer Science Tech. Report 1328*, 1997.

[71] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4):42–50, Aug. 1996.

[72] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of International Symposium on Low Power Electronics and Design*, August 1998.

[73] Prof. Matt Krane. Materials Science Department, Purdue University. Thermal Packaging Models. Personal communication, Dec. 1999.

[74] R. Razdan and M. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the 27th International Symposium on Microarchitecture (MICRO-27)*, Nov. 1994.

[75] P. Reed et al. 250 MHz 5W RISC microprocessor with on-chip L2 cache controller. *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 40:412, 1997.

[76] G. Reinman and N. Jouppi. CACTI 2.0. In *WRL Research Report*, 1999.

[77] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *2nd Workshop on Feedback-Directed Optimization at the 32nd International Symposium on Microarchitecture (MICRO-32)*, Nov. 1999.

[78] H. Sanchez et al. Thermal management system for high performance powerpc microprocessors. *Digest of Papers - COMPCON - IEEE Computer Society International Conference*, page 325, 1997.

[79] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*. SEMATECH, Inc., 2000 edition, 2000.

[80] J. Seng, D. Tullsen, and G. Cai. Power-sensitive multithreaded architecture. In *International Conference on Computer Design 2000*, Sep. 2000.

[81] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Transactions on Computers*, 48(11):1260–810, Nov. 1999.

[82] J. Smith, T. Heil, S. Sastry, and T. Bezenek. Achieving high performance via co-designed virtual machines. In *International Workshop on Innovative Architectures for Future Generation High-Performance Processors and Systems*, October 1998.

[83] A. Sodani and G. Sohi. Dynamic instruction reuse. In *Proceedings of the 24th International Symposium on Computer Architecture (ISCA-24)*, May 1997.

[84] G. S. Sohi and A. S. Vajapeyam. Instruction issue logic for high-performance, interruptible pipelined processors. In *Proceedings of the 14th International Symposium on Computer Architecture (ISCA-14)*, pages 27–34, June 1987.

[85] D. Stefanovic and M. Martonosi. On availability of bit-narrow operations in general-purpose applications. In *10th International Conference on Field Programmable Logic and Applications (FPL 2000)*, August 2000.

[86] M. Stephenson, J. Babb, and S. P. Amarasinghe. Bidwidth analysis with application to silicon compilation. In *ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI)*, pages 108–120, June 2000.

[87] C. Su and A. Despain. Cache Designs for Energy Efficiency. In *Proceedings of the 28th Hawaii Int'l Conference on System Science*, 1995.

[88] Synopsys Corporation. Powermill Data Sheet, 1999.

[89] V. Tiwari et al. Reducing power in high-performance microprocessors. In *35th Design Automation Conference*, 1998.

[90] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: Pushing power management to logic synthesis/design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):1051–60, October 1998.

[91] Y. Tong, R. Rutenbar, and D. Nagle. Minimizing Floating-Point Power Dissipation Via Bit-Width Reduction. In *Power-Driven Microarchitecture Workshop at the 25th International Symposium on Computer Architecture*, 1998.

[92] Transmeta Corp. The Technology Behind the Crusoe Processor Whitepaper, 2000.

[93] M. Tremblay, J. O'Connor, V. Narayanan, and L. He. The Visual Instruction Set (VIS) in UltraSPARC. *IEEE Micro*, 16(4):10–20, Aug. 1996.

[94] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[95] L. Villa, M. Zhang, and K. Asanovic. Dynamic zero compression for cache energy reduction. In *Proceedings of the 33rd International Symposium on Microarchitecture (MICRO-33)*, Dec. 2000.

[96] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-chip Caches. In *WRL Research Report 93/5, DEC Western Research Laboratory*, 1994.

[97] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA-7)*, Jan. 2001.

[98] R. Zimmermann and W. Fichtner. Low-power logic styles: CMOS versus pass-transistor logic. *IEEE Journal of Solid-State Circuits*, 32(7):1079–90, 1997.

[99] V. Zyuban. *Inherently Lower Power High Performance Superscalar Architectures.* PhD thesis, University of Notre Dame, March 2000.

[100] V. Zyuban and P. Kogge. The energy complexity of register files. In *Proc. of Int'l*

Symposium on Low-Power Electronics and Design, pages 305–310, 1998.

[101] V. Zyuban and P. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Proc. of Int'l Symposium on Low-Power Electronics and Design*, pages 84–89, 2000.