Abstract

TOBUREN, MARK CHRISTOPHER. Power Analysis and Instruction Scheduling for Reduced di/dt in the Execution Core of High-Performance Microprocessors. (Under the direction of Dr. Thomas M. Conte.)

Power dissipation is becoming a first-order design issue in high-performance microprocessors as clock speeds and transistor densities continue to increase. As power dissipation levels rise, the cooling and reliability of high-performance processors becomes a major issue. This implies that significant research needs to be done in the area of architectural techniques for reducing power dissipation.

One major contributor to a processor's average peak power dissipation is the presence of high di/dt in its execution core. High-energy instructions scheduled together in a single cycle can result in large current spikes during execution. In the presence of heavily weighted regions of code, these current spikes can cause increases in the processor's average peak power dissipation. However, if the compiler produces large enough regions, a certain amount of *schedule slack* should exist, providing opportunities for scheduling optimizations based on per-cycle energy constraints.

This thesis proposes a novel approach to instruction scheduling based on the concept of schedule slack, which builds energy efficient schedules by limiting the energy dissipated in a single cycle. In this manner, a more uniform di/dt curve is generated resulting in a decrease in the execution core's average peak power dissipation.

POWER ANALYSIS AND INSTRUCTION SCHEDULING FOR

REDUCED di/dt IN THE EXECUTION CORE OF

HIGH-PERFORMANCE MICROPROCESSORS

by

MARK CHRISTOPHER TOBUREN

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the requirements for
the Degree of Master of Science

COMPUTER ENGINEERING

Raleigh

1999

APPROVED BY:

_____   _____

_____

Chair of Advisory Committee

BIOGRAPHY

Mark Christopher Toburen was born on August 16, 1974 in Winston-Salem, North Carolina to Rick and Donna Toburen. Mark has two older brothers, Toby and Will, and a younger sister, Erin. Mark spent the first years of his life moving from place to place while his father climbed the corporate ladder, but his family eventually settled back in Winston-Salem when he was six years old. Mark spent the vast majority of his elementary and high school years swimming for the Winston-Salem YMCA swim team. After a very successful high school swimming career, Mark decided to follow his older brother, Will, to North Carolina State University to swim and study Pre-Med. However, Mark made the decision to change majors after two semesters, and in December of 1996 graduated Cum Laude with a B.S. degree in Computer Engineering. After spending eight months after graduation at Intel Corporation in Oregon, Mark decided to return to N.C. State to attend graduate school. Mark joined the TINKER microprocessor design group in November of 1997, and is currently continuing with research under the advisement of Dr. Tom Conte in the areas of architectural-level power analysis, power-aware performance modeling, and power/performance tradeoffs in high-performance microprocessors.

ACKNOWLEDGEMENTS

First and foremost I must thank God for giving me the talent and ability to perservere in such a rigorous and demanding field of study. Second I would like to thank my parents, Rick and Donna, for always encouraging me and for equipping me with all the tools I would ever need to go out and achieve any goal I set for myself. Thanks to both of my older brothers, Toby and Will, and my little sister, Erin, for all of their support and especially for their great friendship. I would also like to give special thanks to my grandmother and grandfather, Doris and Fred Van Pelt, for always being a great source of encouragement.

Thanks to all of the teachers I have had throughout the last nineteen years, in particular Mrs. Bistline who never let me get away with anything even when it was not my fault, Mr. Knight who made me realize school could actually be fun, Mrs. Brown who made me sit out of recess because I did not do my word problems, Mr. Grubbs who instilled in me a love for reading and who introduced me to J.R.R. Tolkein, Mrs. McFadden who forced a lazy young prankster to reach his potential, Mr. Lawrence who was the only teacher I ever had who could make poetry enjoyable and who gave me an appreciation of great literature, and Mr. Feree who provided a constant source of laughter and inspired numerous practical jokes during calculus class.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Overview

Power dissipation is increasingly becoming a major design constraint in today's high-performance microprocessors. As processor designers continue to push the limits of clock speed and transistor densities, the corresponding levels of total chip power dissipation are increasing proportionally. We can see this effect through the simple equation for dynamic power dissipation:

$$P_d = C_L V_{dd}^2 f \tag{1.1}$$

Where C_L is the total load capacitance on all nodes, V_{dd} is the supply voltage, and f is the operating frequency. As transistor densities increase so does the total load capacitance because there are inherently more nodes. Advanced process technology does provide some relief from the power dissipation problem in that it allows for lower supply voltages. From Equation 1.1 it can be derived that a 1 volt reduction in supply voltage can provide a factor of 4 reduction in the dynamic power dissipation. However,

the rapid growth in clock frequencies and transistor densities are overpowering the advances in process technology, so power dissipation continues to rise along with those two factors.

The problem which is now arising as a result of increasing levels of power dissipation is that of packaging. As power dissipation reaches higher and higher Wattages, packaging technology is becoming a limiting factor. Architects and designers are begining to be forced to implement designs based on projected package capabilities. This has not been a primary design constraint in the past. As an illustration, Table 1.1 shows power dissipation trends in some of today's high-end processors, [1].

Thus, we are faced with an interesting question, "What can be done to enable us as architects to continue to push transistor densities and clock frequencies without having to worry so much about whether the package will be able to handle the heat coming off of the die?" It is the conjecture of this thesis that the answer lies in architectural and compiler optimizations for reduced power dissipation.

1.2 Power Dissipation In CMOS Circuits

In order to answer the question posed above, there must first exist a fundamental understanding of power dissipation in digital CMOS circuits. There are three major components to CMOS power dissipation: static dissipation, dynamic dissipation, and short-circuit dissipation, [2].

Static dissipation occurs as a result of leakage current from reverse biased parasitic

Table 1.1: Power dissipation trends in modern high-performance processors. The data described below, from left to right, are peak power dissipation, clock speed, chip area, transistor count, and process technology, [1].

Processor	Peak W	Clk (MHz)	Area (mm^2)	# Trans. (10^6)	Tech (μm)
Pentium Pro	29.4	166	195	5.5	0.35
Pentium II Xeon	23.3	400	131	7.5	0.25
Alpha 21164	50	300	299	9.3	0.5
Alpha 21264	72	667	302	15.2	0.35
Alpha 21364	100	1000	350	100	0.18
HP PA 8000	40	180	345	3.9	0.5
MIPS R10000	30	195	298	6.8	0.35
SPARC ULTRA II	25	250	149	5.4	0.35
Power PC 704	85	533	150	2.7	0.5
AMD K6	28.3	233	162	8.8	0.35

diodes between diffusion regions and the device substrate. The total static power dissipation, as given in Equation 1.2, can be obtained by taking the sum of the leakage current times the supply voltage across all devices (n).

$$P_s = \sum_1^n (I_{leakage} * V_{dd}) \qquad (1.2)$$

A more significant contributor to static dissipation may result from subthreshold conduction of the p- and n-type devices [2], [5]. High-performance processors whose designs are driven by speed typically employ devices which have very small threshold voltages. As a result, these devices result in much more significant subthreshold conduction. As an example, the DC power dissipation of the Compaq Alpha 21264, $2W$, is almost fully attributed to subthreshold leakage, [5].

3

The equation for dynamic dissipation was previously given in Equation 1.1, but a short explanation is warranted here. Dynamic dissipation arises as the result of current begin driven to switch capacitive loads on gate outputs which result from wiring capacitance and gate capacitance of fanout devices. Intuitively, as the load capacitance increases, the charging/discharging current must also increase, thereby increasing the dynamic power dissipation. In addition to the load capacitance, the dynamic element is also dependent on the frequency at which the load is switched as well as the supply voltage. Thus giving Equation 1.1 as the dynamic dissipation.

The final element, short-circuit dissipation, occurs as a result of DC paths from V_{dd} to ground (V_{ss}) resulting from simultaneous conduction of p- and n-type devices. This simultaneous conduction allows for short bursts of current between the two supply nodes resulting in increased power. The effects of short-circuit dissipation are brought about by finite signal rise and fall times. During the rising/falling edge of a given signal there is a period of time during which both p- and n-type devices are conducting. However, short-circuit dissipation is typically dominated by the capacitive element of dynamic dissipation in high-performance processors due to their significantly high levels of load capacitance and their typically fast edge rates.

1.3 Previous Work In Low-Power Design

Now that a basic understanding of CMOS power dissipation has been established, answers to the above question become much more evident. The following section

discusses several approaches to alleviating the power dissipation problem ranging from circuit level techniques all the way up to architectural level and compiler techniques. This discussion provides motivation for the remaining chapters of this thesis.

1.3.1 Circuit Techniques

Until recently researchers have focused primarily on attacking the power problem at the circuit level. A large body of work on low-power circuit design techniques exists in the literature. A good overview of modern low-power circuit techniques is given in [3] by Horowitz, et al. In this work the authors discuss several important low-power design issues and evaluate their goodness with respect to the *Energy*Delay* product. More specifically the authors discuss how techniques such as supply voltage scaling, transistor sizing, adiabatic circuits, technology scaling and transition reduction can be used to balance power and performance.

The supply voltage directly affects the dynamic power dissipation as shown in Equation 1.1. If supply voltage is reduced, then dynamic power dissipation can be reduced by a factor of four due to the quadratic term on V_{dd}. However lower supply voltage generally translates into slower circuits as demonstrated by Equation 1.3 given in [4]:

$$t_d = k * (C_L V_{dd}/(V_{dd} - V_{th})^2) \qquad (1.3)$$

If the load capacitance, C_L, and the threshold voltage, V_{th}, are constant, the delay

time, as given in Equation 1.3, will increase as the supply voltage, V_{dd}, decreases. To compensate for the loss in performance, threshold voltage can be scaled down which, if done in proportion to the supply voltage scaling, should provide a corresponding decrease in delay time. Horowitz, et al. refer to this technique as technology scaling. However, they point out that the problem with technology scaling is that threshold voltage does not generally scale with supply voltage.

Horowitz, et al. also discuss techniques for reducing the number of transistions on a node which directly affects the dynamic power dissipation by reducing the frequency term. A final technique, which again implies a power/performance trade-off, is transistor sizing. Smaller transistors result in less energy, but they also lead to reduction in performance. On the other hand, large transistors provide performance gains but are large energy comsumers. Horowitz, et al. define the optimum balance as the point where transistor and wire loading are the same, thus providing the best performance for the least amount of power.

In addition to the summary work done by Horowitz, et al., the work presented in [5] and [6] provides on overview of specific techniques used in the design of the Compaq Alpha 21264 to minimize power dissipation. In [5] Gowan, et al. cite four specific techniques used in the 21264: reduced supply voltage (V_{dd}), low-swing buses, complementary and dynamic logic, and hierarchical clocking schemes. The benefits of lowering the supply voltage have already been discussed. Low-swing buses provide nodes that need limited voltage swing to change state which directly reduces the

power due to switching of capacitive loads. This technique is particularly useful when implementing long, heavily loaded buses. As an example, Gowan, et al. demonstrate the power savings achieved in the 21264's Ebox by implementing the operand buses as low-swing, differential buses. Operating at $600MHz$ and swinging from V_{ss} to V_{dd}, the worst-case power dissipation in the operand buses was estimated at $1.5W$. However, by designing the bus receivers to operate at $200mV$ of differential, the power dissipation was reduced to $15mW$.

The use of dynamic circuits also helped reduce the power dissipation of the 21264 in a number of ways. First, dynamic circuits reduce gate and load capacitance due to the absence of the complementary p-device stack used in static gates. Second, the use of dynamic gates eliminates extra power dissipation due to gitches. Finally, dynamic gates can reduce crossover current by ensuring minimal overlap between precharge and evaluate phases of operation. In contrast, the use of complementary logic allows designers to limit the number of spurious transitions on input nodes. Gowan, et al. found that this technique was especially useful in large blocks of control circuitry.

The final power saving technique discussed in [5] is the use of a hierarchical clocking network. Hierarchical clocking resulted in reductions in both average and peak power dissipation in the 21264. As detailed in [6], hierarchical clocking reduces power in several ways. First, the use of regional major clocks which branch off of the global clock (GCLK) results in the ability to use smaller drivers and a less dense clock grid to provide the same global clock skew and edge rates. In addition, the major clock grids

can be individually sized depending on their loads, thus allowing greater freedom in reducing total clock power. Second, the hierarchical scheme allows for conditional clocking which enables designers to turn off large sections of circuitry when not in use resulting in reduced clock load. However, the drawbacks to the hierarchical scheme are two-fold. First, an adverse effect of conditional clocking is increased di/dt on a cycle-by-cycle basis. Second, timing analysis of this scheme is much more complex than for a single-wire, global clock grid.

All of these techniques were employed in the Alpha 21264 to maintain manageable power dissipation levels, but, despite their benefits, the 21264 still dissipates 72 Watts on average. The Compaq Alpha family of processors is a good example of why low-power circuit techniques are not sufficient and further research into architectural power optimizations is necessary.

1.3.2 Microarchitecture Techniques

General

Although the circuit-level techniques discussed above are quite effective, their effectiveness has a limited scope. In order to achieve significant power savings, researchers and designers must focus on the problem at a higher level. Recently a great deal of interest has been sparked in the area of low-power microarchitecture. In [7] an overview of microarchitectural techniques for reducing energy dissipation is given. It is shown that the majority of the energy dissipation in high-performance superscalar proces-

sors is due to 1) reading and writing to the on-chip caches and register files and 2) the clock network. Gonzalez and Horowitz suggest several optimizations that can be performed to eliminate what they term *wasted energy*. First and foremost, they suggest that *clock gating* be used to eliminate spurious transitions in the processor pipeline. By eliminating unnecessary transitions in the pipeline latches, the total power dissipation in their experimental processor was reduced by close to 15%. The second energy optimization proposed in [7] is selective activation of major macro-cells in the design, specifically the caches and the register file. By restricting cache and register file accesses, 8% of the total energy dissipation was saved. The final optimization discussed is to pre-decode instructions coming from the second level cache and to store additional bits in the I-cache to determine if operands of speculative instructions actually need to be read from the register file. In doing this the total number of register file accesses is reduced without seriously affecting the effectiveness of speculative execution.

The benefits of the techniques presented in [7] are clear, but the problem is that, after optimizing a design to minimize energy dissipation in the on-chip memories and the clock network, there is little room for major energy savings in any single processor block because the remaining energy being dissipated in the chip is somewhat thinly dispersed across the remaining processor units. Thus, the remaining blocks of the processor need to be optimized as a whole in order to obtain additional significant power savings.

Memory Hierarchy

As stated in [7], approximately 50% of the total power dissipated in high-performance processors comes from the clock network and the on-chip memories. Consequently, most research in the realm of low-power microarchitecture has been focused on the memory hierarchy. There is a large body of research which deals with low-power cache and TLB designs. In [8], Su and Despain provide an excellent overview of energy consumption in high-performance caches, and they suggest several novel techniques for providing low-energy, high-performance cache structures. In [8] the total cache energy is broken down into the sum of three components: the address path energy, E_{addr}, the memory cell array energy, E_{cell}, and the I/O path energy, $E_{i/o}$. The authors further break these components down as follows. The address path energy is dominated by the capacitance that is switched on the memory address bus. The capacitance driven by the address decoder is negligible compared to that of the address bus. So E_{addr} is determined by multiplying the address bus swicthing rate by some scale factor. The memory cell array component, E_{cell}, is determined by taking the summation of the tag and memory array access frequencies. The reason behind this is the majority of today's high-performance caches are designed using dynamic, as opposed to static, logic. Therefore, all of the bit lines in the memory arrays are precharged on each access, and whether a specific line's value is a logic-0 or a logic-1 is insignificant because all lines will incur the same voltage swing. So the energy of the memory cell arrays is purely dependent on the number of times the array is accessed.

The final component, $E_{i/o}$, is estimated in much the same way as the address path energy. $E_{i/o}$ is determined by taking the summation of the weighted bit switching rates on the address and data I/O pads.

Su and Despain discuss three novel techniques for building low-energy caches in [8]. The first technique they discuss is the idea of *block buffering* which involves placing a small cache-like buffer between the CPU and the first-level cache which holds the most recently accessed blocks from the cache. The idea behind this technique is that if there is a significant amount of spatial and temporal locality in the cache access pattern, then the number of cache accesses can be reduced by way of the block buffer providing the neccessary data. Experiments in [8] show that block buffering can lead to cache energy consumption which is only 25% and 60% of that for traditional I-cache and D-cache designs, respectively. Su and Despain also show that block buffering can have a performance advantage over traditional cache designs. Since the access time of the block buffer is short as opposed to typical cache access times, the clock speed of a processor with block buffering caches may be increased.

The second low-energy cache design technique discussed in [8] is *cache sub-banking*. Cache sub-banking is based on the concept that the specific location of the requested data inside of a cache block is known from the cache address. Hence if the data memory is split into sub-banks, or columns, which are one data word long, then only the desired sub-bank needs to be accessed in order to fetch the requested data chunk. In this way the amount of extra energy consumed by accessing unneeded data is

11

significantly reduced. The authors point out that the advantage of sub-banking over block buffering is that the amount of energy savings is independent of the spatial locality of the data being accessed. In addition the performance of the sub-banked cache is comparable to that of traditional cache designs since the sub-bank selection logic is simple and does not affect the cache access time. Sub-banking does have one disadvantage with respect to block buffering which is that sub-banking does not reduce the energy consumed in the tag memory array. The authors point out that this is more of a problem when the block size is relatively small. In [8] the energy savings of the sub-banking technique is shown to be approximately 90% of the energy consumed by traditional I-cache and D-cache designs.

The final low-energy technique presented by Su and Despain involves optimizing the energy in the address and I/O paths. The authors propose the use of *Gray code addressing* in order to minimize the amount of bit switching on the cache address and I/O address paths which dominate the energy dissipation in those two areas as discussed previously. In [8] the data presented shows that the use of Gray code addressing can significantly decrease the amount of bit switching in both paths resulting in further energy savings in the cache hierarchy[1].

In a related work on low-energy cache design, [9], Kin, et al. introduce a structure called the *filter cache*. The filter cache resembles the block buffer of [8] discussed

---

[1]The study of Gray code addressing and other encoding techniques for reducing power is not limited to cache buses but has been applied with respect to system buses and state assignment as well and is extensively covered in the literature.

above. However, the filter cache is a significantly larger structure than the block buffer. The filter cache acts as an L0 cache in front of the traditional L1 cache(s). This organization does lead to increased access time to the L1 cache, but the authors show that the performance degradation incurred is more than compensated for with the reduction in energy consumption provided. [9] shows an average reduction of the *Energy*Delay* product, [7], of 51%. Although the performance loss of the filter cache technique may not be acceptabel for high-performance desktop processors, it may be a viable architectural option for embedded processors.

The discussion on low-power caches above is certainly not all encompassing. There is much more work in the research community which deals with reducing the power consumption and dissipation caused by processor caches and TLBs, [10], [11], [12]. However, the above discussion provides a solid overview of the type of research that is being conducted in this area.

Processor Pipeline

Most of today's high-performance processors employ such features as out-of-order execution, speculative execution through accurate branch prediction, and advanced compiler optimizations to increase the amount of *instruction-level parallelism* (ILP) seen by the processor. By doing so, processor resources such as execution units, caches, etc. experience higher utilization. However, in addition to the performance enhancements these features provide, the increased utilization of processor resources

results in increased power dissipation.

Dynamically scheduled processors depend heavily on accurate branch prediction in order to speculatively execute instructions across control-flow boundaries. However many applications have relatively high branch misprediction rates. When branches are mispredicted, the processor temporarily issues and executes instructions which lie along the mispredicted control-flow path. This wrong-path execution results in excess power dissipation. A technique to alleviate this problem was introduced in [13] which gates the processor pipeline when a specific number of outstanding *low-confidence* branches exist in the fetch and decode stages. The pipeline is stalled until those branches are resolved and the processor can be sure that it is issuing and executing correct-path instructions. This technique, called *pipeline gating*, has been shown to provide up to a 38% reduction in the number of wrong-path instructions while only incurring an approximate 1% loss in performance, [13].

1.3.3 Compiler Techniques

In addition to the techniques discussed up to this point, there has been a small amount of research in the area of using software, specifically compiler and scheduling technology, to help manage and reduce processor power dissipation. This is the area from which this thesis approaches the power dissipation problem, so the following discussion is key in that it describes the previous software approaches, points out their weaknesses, and gives insight into how compilers and instruction scheduling can

be better used to combat this problem.

In [14] a technique called *cold scheduling* was combined with Gray code addressing to reduce the switching activity in the control path of high-performance processors. The cold scheduling algorithm works much the same as traditional *list scheduling* with the exception that it schedules instructions based on a modified priority scheme. In cold scheduling the priority of an instruction is based on the power cost of that instruction when it is scheduled following the previously scheduled instruction. The power cost of an instruction is given by $S(I, J)$ where S represents the switching activity caused by executing instruction I after instruction J. The power costs are stored as entries in a power table. Instructions with lower S values have higher priority, and after each instruction is scheduled the power cost of each remaining instruction in the ready-list must be recalculated before scheduling the next instruction.

Although the authors in [14] report an average reduction in switching activity of 20-30% due to the combination of cold scheduling and Gray code addressing, there are several problems with this approach that need to be pointed out. First, it has been argued by some researchers that the switching overhead between different instructions in the pipeline control path is not significant enough to warrant such approaches. Indeed, the work in [15] and [16] support this stance. However, as shown in [17], this conjecture does not hold for all processors. Therefore, this technique is not applicable for all processor architectures. Second, the cold scheduling algorithm relies on a large table which stores the power cost of all possible instruction combinations for a region

of code. Depending on the region formation algorithm, the size of this table could grow rather large. Finally, this table must be accessed several times after each instruction is scheduled. These last two points may result in significant increases in schedule time.

A differing approach to low-power instruction scheduling from that proposed in [14] is introduced by Tiwari, et al. in [15] and [16]. The goal in these works is to reschedule code such that instructions are more judiciously chosen as opposed to actually reordering instructions to reduce power. In order to achieve their goal of low-power schedules, the authors implement a methodology for determining the energy consumption of a single instruction in which they actually hand measure the current through the processor when that instruction is executed. From these measurements a table of energy data for individual instructions and pairs of instructions is built and referenced during the instruction scheduling phase. In their work Tiwari, et al. reschedule code to use instructions and instruction pairs which require less energy - basically, strength reduction and instruction collapsing. The selection of these instructions is based on the energy analysis mentioned above and on factors such as register accesses as opposed to memory accesses and lower latency instructions which provide the same functionality. In addition to the individual instruction analysis the authors consider *circuit state overhead* which is the overhead refered to in [14]. In [15] and [16] the circuit state overhead is argued to be an insignificant factor in the overall processor power consumption. However, after anaylzing a different processor

architecture in [17] in which circuit state overhead was a significant contributor to the total power comsumption, it was determined that the effect of circuit state overhead is dependent on the processor in question.

As in [14], there are problems with the approach presented by Tiwari, et al. in [15], [16], and [17] despite the reported energy savings of up to 40% after code rescheduling. First, the process of hand measuring the current characteristics for all instructions and instruction pairs, although valuable, is extremely time consuming especially in light of the large instruction sets used in today's high-performance processors. Second, like Su, et al., there is a rather large table needed for energy characteristic lookup during code rescheduling. Again, this will lead to significantly slower scheduling times especially for large applications. Finally, the studies presented by Tiwari, et al. deal only with embedded DSP processors, and they are insterested in power consumption as opposed to power dissipation.

1.4   Research Contributions

The goal of this thesis is to present a novel approach to aid in solving the power dissipation problem in today's high-performance processors. Specifically, this thesis deals with power dissipation in the context of statically scheduled, VLIW processors. The techniques presented herein take advantage of modern compiler technology and constraint-driven instruction scheduling to schedule code in a manner which controls the levels of peak power dissipated by the processor execution core by eliminating

17

large current spikes in the core during program execution.

1.5   Thesis Outline

The remainder of this thesis provides the details of the low-power scheduling algorithm along with the energy analysis methods used to determine the energy characteristics of the functional units used in this study. Chapter 2 discusses those techniques and describes the software interface which provides the necessary energy information to the instruction scheduler. Chapter 3 discusses the concepts which form the basis of the work presented in this thesis along with in-depth analysis of the proposed algorithm and the results from the studies performed on it. Chapter 4 discusses an alternative approach to low-power scheduling aimed at amortizing the performance cost of the base algorithm. Chapter 5 concludes the thesis and discusses potential avenues of future research regarding low-power scheduling and low-power processor architecture in general.

1.6   Experimental Framework

All of the scheduling experiments in this thesis were run using the experimental LEGO compiler developed by the TINKER microprocessor research group at North Carolina State University. LEGO works with the TINKER instruction set, which is based upon the PlayDoh architecture defined by the Compiler and Architecture Research Group at HP Labs, [18].

All experiments were performed using the SPEC95 integer benchmark suite. In all cases, except when otherwise noted, scheduling was performed for an eight-issue VLIW machine model called TINKER-8, which consists of three integer ALU units, two load/store units, two floating-point units, and a single branch unit. All instructions have single-cycle latency except for loads, floating-point multiply, and floating-point divide, which have latencies of two, three, and nine cycles, respectively. In addition, all non-unit latency function units are fully pipelined.

The results for the low-power scheduling algorithm given in Chapter 3 with respect to slowdown and di/dt savings are given as percentages of the total number of execution cycles and total energy per benchmark, respectively, for normal list-scheduling using the region type under examination.

Chapter 2

Energy Modeling

In order to perform any type of software scheduling for reduced power dissipation, the compiler must have knowledge of the energy dissipation characteristics for each of the processor's functional units. In addition, the energy models must be accurate for all possible input combinations. For example, the energy model for a processor's integer ALU must accurately reflect the energy dissipated from that unit with respect to all operation types and all possible operand combinations. The need for such accuracy presents a problem when trying to develop highly accurate methods for generating such models with respect to large *bit-dependent* circuits such as ALUs. The problem that arises is due to the inherent dependency of the energy characteristics of such structures on their input operands. Thus, to obtain highly accurate energy models, one would need to run circuit simulations across the entire boolean space of the circuit. However, for large structures such as a 64-bit Manchester adder, this is not pheasible due to the size of that space and the time it would take to run such an extremely large number of simulations. Therefore, the first portion of this chapter deals with previous work on bit-dependent circuit energy analysis and how best to

generate accurate, average energy models for the purpose of low-power instruction scheduling.

In addition, to contrast the bit-dependent case, *bit-independent* circuit structures, such as caches and register files, will also be dealt with in terms of how to model them with respect to energy dissipation. Bit-independent structures can be modeled much easier than their bit-dependent counterparts since the energy dissipated from those structures is not strictly dependent upon current and previous operand values. In fact, most of these structures can be characterized by hand using implementation specific data and a basic understanding of the power breakdown between the various parts of each structure. The discussion of bit-independent circuit energy modeling will be centered around the development of the energy models for the register file and the level-one caches used in the studies on low-power scheduling presented in the next chapter.

2.1   Previous Work In Energy Analysis And Modeling

Fast, accurate circuit energy modeling has become an increasingly important area of research in the microarchitecture and circuit design communities due to the emergence of hand-held and portable devices requiring minimal power consumption and dissipation for increased battery life and reliability. In addition, general-purpose processor manufacturers are also becoming increasingly involved in low-power design research due to the potential problems of package cooling and device reliability that arise with

increasing clock speeds and single-die transistor densities.

As discussed in the first chapter, power dissipation in a digital CMOS circuit is primarily dependent on the switching characteristics of the circuit. In terms of estimating circuit power dissipation, this quality significantly complicates matters because the switching characteristics of a piece of combinational circuitry may vary significantly depending upon the circuit's primary inputs. For large circuits, the Boolean space of possible input combinations can be extremely large. In general, the Boolean space of a combinational circuit with n inputs is of size 2^n. However, in reality the switching characteristics of a circuit depend on both the current inputs as well as the previous inputs. So the actual number of possible input combinations increases to 2^{2n}. This characteristic alone precludes the possibility of running detailed device-level simulations across the entire Boolean space in order to characterize the average power dissipated by a large combinational circuit. Such large combinational structures are termed *strongly pattern-dependent*, [19].

This section aims at providing sufficient background information on effective techniques for performing fast and accurate energy modeling for large CMOS circuits. In general, two approaches to energy modeling will be discussed - *probabilistic* and *deterministic*.

2.1.1 Probabilistic Energy Modeling

Over the course of the past five to ten years, researchers have proposed several different methodologies for estimating average power in large circuits by characterizing input vector sets probabilistically, eliminating the need to run repeated detailed simulations over large vector suites, [19], [20], [21], [22], [23]. In these approaches, the primary inputs to the various energy analysis tools are represented as probabilistic approximations of a larger set of real input vectors. These probabilistic inputs are then propagated into the internal nodes of the circuit to obtain the probabilistic characteristics of these signals. In essence, these techniques are still pattern-dependent but much more weakly so than the direct simulation based techniques alluded to previously. Thus, a key issue in the design of a probabilistic energy modeling algorithm is how to generate the transition probabilities and how best to represent them.

Two of the most common ways of defining probability measures for signal characteristics are the *signal probability* and the *transition density*. The signal probability for a node x is defined in [19] as "the average fraction of clock cycles in which the steady state value of x is a logic high." The problem with using this measure and similar measures of probability is that it is not affected in any way by the internal delays of the circuit being analyzed. For many circuits this is a serious pitfall because internal nodes may switch multiple times before settling down into their steady state. This effect is commonly known as *glitching* and may contribute significantly to the total energy dissipated in a circuit. A probabilistic measure which accounts for the

glitchy nature of internal circuit nodes is the transition density, which is defined in [19] for a node x as "the average number of trasitions per second at node x."

A second issue which can seriously affect the accuracy of probabilistic techniques is the modeling of *spatial* and *temporal* correlations among signals. Spatial correlations refer to the dependence between the switching characteristics of separate, adjacent nodes. Temporal correlations refer to the dependence between signal values on the same node at consecutive time steps. The inclusion and proper modeling of spatial and temporal correlations can have a significant effect on how accurately a specific algorithm is in estimating circuit energy. By probabilistically modeling signals which are strongly spatially and/or temporally correlated one can easily overestimate energy by including signal combinations in both realms which are not actually realizable in the circuit. This effect is especially prevalent in the modeling of control circuitry.

The main benefit of probabilistic energy modeling is the reduction in computation time necessary to characterize large circuits. However, this comes at the cost of reduced accuracy in the generated energy models as compared to more deterministic approaches. The loss in accuracy is dependent on what measures of probability are employed and how spatial and temporal correlations are handled. Typically, employing more detailed probabilistic measures and detailed handling of signal correlations results in increased computation time but produces better results.

2.1.2 Deterministic Energy Modeling

To contrast the previous section on probabilistic energy modeling techniques, this section presents the concept of deterministic energy modeling. Although these techniques do involve detailed device-level simulations, they should not be characterized as *fully deterministic* in the sense that they do not perform exhaustive simulations accross the entire Boolean space in order to obtain energy characteristics. Instead, these techniques propose that, given a set of test vectors, after a certain amount of simulation time the observed power output from the given simulator will converge, within certain error and confidence bounds, to the average for the circuit under investigation. In essence, they are *statistically deterministic*. Works representative of such techniques are described in [19], [24], [25], and [26].

The key to the success of these approaches is the manner in which the inputs are provided and the criterion used to determine when the average power has been reached. In each of the techniques described in the above works, the inputs to the simulation engines employed are randomly generated. The main point in which these techniques differ is in the manner in which they determine when the average power for the circuit in question has been reached. However, they are all statistical in nature and each implementation allows the user to specify the relative error and confidence levels desired to control the simulation accuracy.

The benefits of the *statistically deterministic* methods are two-fold. First, they can provide high levels of accuracy by using detailed simulations. In doing so, these

approaches do not have to worry about the issues of spatial and temporal correlation because they are handled naturally by virtue of fact that the actual internal node transitions are modeled explicitly. Second, by using statistical methods to determine when the average power has been reached by the simulator, the time necessary to produce accurate results is greatly reduced as compared to a more *fully deterministic* approach.

2.2 Initial Energy Model and Estimation Methods

This section presents the methodology used to generate the inital energy model used in the next chapter's low-power instruction scheduling experiments. In particular, it provides a detailed analysis of how the energy models for the bit-independent circuit structures, namely the caches and register file, were developed.

2.2.1 Estimation Methods for Bit-Independent Structures

Bit-independent circuit structures are circuits whose energy dissipation is not heavily dependent upon the inputs to the circuit and in which one bit-slice's operation does not generally affect another bit-slice's operation. Examples of bit-independent structures are registers, register files, caches, and some logical operators. In contrast to their bit-dependent counterparts, energy characterization of bit-independent circuits can be easily and quickly carried out through simple, straight forward hand calculations. For the studies presented in this thesis relating to low-power scheduling, the

major bit-independent circuits that were characterized were the register files and the level-one (L1) caches.

Register File Modeling

Most of the energy dissipated from the register file comes from the charging and discharging of the bit lines in the data array and the address lines leading into the address decoders. Thus, the critical element in determining the energy dissipated by the register file is the number of read and write ports, R and W respectively, since these factors determine the height of the register file. From the number of read and write ports the total height of the register file can be defined by Equation 2.1 as follows:

$$
H_{rf} = \begin{cases} 50\lambda * N_w, & \text{if } R + W \leq 5, \\ 10\lambda * (R + W) * N_w, & \text{otherwise.} \end{cases} \tag{2.1}
$$

where N_w is the number of words or registers in the register file and λ is the technology dependent scale factor. The equation above is based on the premise that a minimum sized, stable storage cell is about 50λ units in height and that, beyond the 50λ minimum, each additional word line is approximately 10λ units in height, [27].

For the studies presented herein, it was assumed that register read operations require differential bit lines, register write operations require only a single bit line, and that all bit lines run the entire height of the data array. We also assume that, during both read and write operations, half of the bit lines are discharged. Thus,

27

the power supply will be required to charge those lines back to the precharge voltage after they are discharged by the operation. This translates to the following equation which determines the total amount of capacitance in the bit lines that must be charged/discharged on a read operation:

$$C_r = H_{rf} * B * (0.06fF/\lambda) \qquad (2.2)$$

where H_{rf} is the height of the register file as determined in Equation 2.1, B is the bit width of the register file, and the final term represents the capacitance per unit length on the bit-line wires. Similarly, the total amount of capacitance charged in the bit lines by a write operation is given in Equation 2.3. Again, it is assumed here that write operations do not require differential bit ines, so the total capacitance charged/discharged in the bit lines by a write is only half of that charged/discharged by a read.

$$C_w = H_{rf} * (B/2) * (0.06fF/\lambda) \qquad (2.3)$$

Equations 2.2 and 2.3 cover the capacitance charged in the bit lines of the data array. The second major contributor to the register file energy dissipation is the capacitance charged by the address lines feeding the address decoders. For each register file access, a register address must be driven into a decoder which is H_{rf} units in height. Equation 2.4 gives the total capacitance charged by the address lines

feeding the decoder:

$$C_d = H_{rf} * log_2(N_w) * (0.7fF/\lambda) + (N_w/2) * W_p * (0.6fF/\lambda) \qquad (2.4)$$

where W_p is the width of a pulldown device and there are $N_w/2$ pulldown devices seen by each of the $log_2 N_w$ wires entering the decoder array. From Equations 2.2, 2.3, and 2.4 the total energy dissipated for read and write register file accesses can be estimated by Equations 2.5 and 2.6, respectively.

$$E_r = ((C_r + C_d) * V_{dd}^2)/2 \qquad (2.5)$$

$$E_w = ((C_w + C_d) * V_{dd}^2)/2 \qquad (2.6)$$

Level-One Cache Modeling

Energy modeling of level-one (L1) caches follows along the same logic as that for the register file with a few exceptions. The first major difference in L1 cache energy modeling is that there is only one port each for read and write accesses and one address decoder. Second, for the studies presented in this thesis, it is assumed that words are packed four to a cache line, so N_w must be divided by four when calculating the height of the cache as shown in Equation 2.7.

$$H_{cache} = 50\lambda * N_w/4 \qquad (2.7)$$

29

Recall from the discussion on register file energy modeling in the previous section that 50λ is the minimum height of a stable storage cell. Third, since the words are packed four to a cache line, there are four times as many bit lines to be charged/discharged during cache reads and writes. In addition, assuming caches constructed out of static RAM (SRAM) cells, cache writes cost the same as cache reads in terms of the capacitance charged/discharged in the bit lines[1].

$$C = H_{cache} * (B * 4) * (0.06fF/\lambda) \qquad (2.8)$$

Equation 2.8 shows the modification necessary to the register file capacitance equations for L1 cache reads and writes. In addition, the capacitance switched in the address lines feeding the decoder, C_d, follows Equation 2.4 where the height of the register file, H_{rf}, is replaced by the height of the cache, H_{cache}. Finally, the energy required for any cache access is given by Equation 2.9:

$$E_c = ((C + C_d) * V_{dd}^2)/2 \qquad (2.9)$$

2.2.2 Base Processor Energy Model

This section will briefly describe the energy models used in the low-power scheduling studies presented in the next chapter. This model is based solely upon hand estimation of the basic function unit (FU) types defined in the studied architecture. Several

---

[1]Both reads and writes are differential in SRAM.

assumptions were made with respect to these calculations, and they will be presented as the discussion progresses. The basis for these calculations were obtained from [27], and portions of the data were extrapolated from a proprietary process technology and adjusted so as to prevent disclosure of any proprietary information.

Base Energy Model

Table 2.1 shows the energy values used to chracterize the FUs used in our VLIW machine model. As mentioned previously, the energy values listed in the table calculated by hand based on data received from [27].

Table 2.1: Function unit energy models. The data presented here is in nJ. All numbers are average values for each FU type and have been normalized to a logical unit whose energy dissipation is equal to $1nJ$.

Function Unit Type	Average Energy Dissipation (nJ)
Integer ALU	1.87
Load/Store	4.4
Branch	1.17

Several asssumptions were made during the initial model development. First, the integer ALU (IALU) and branch unit energy values are normalized to a $1nJ$ per operation logic block. Models for both units include the cost of register file accesses for reading and writing source and destination operands. The IALU energy model is based on a 64-bit, blocked Manchester adder. Due to the time needed to develop

31

good energy models, there currently is no differentiation in terms of energy for various types of integer operations such as multiplies, divides, and shifts. Although the use of the adder model is sufficient to provide the necessary information to the compiler for use in low-power scheduling, this generalization with respect to the IALU energy will result in reduced accuracy in terms of determining realistic cycle-by-cycle energy dissipation during scheduling. The modeling of the various constituent parts of an IALU, as well as floating-point structures and various other finer-grain function units, is the subject of current work, and the inclusion of these finer-grain models in the low-power scheduling infrastructure is discussed as a topic of future work in the concluding chapter of this thesis.

Second, as previously mentioned in the sections on register file and L1 cache modeling, the capacitance per unit area values used to determine the load capacitance values were scaled to prevent disclosure of proprietary process information. The register file parameters used in determining the energy cost of a register file access are as follows: 80 registers deep, 64 bits wide, 8 read ports, and 4 write ports. Similarly, the parameters used to characterize the L1 caches are: 64KBytes total size and four 64-bit words to a cache line. Finally, loads and stores are assumed to be to the L1 caches only - we assume perfect L1 caches. The effects of cache misses on the total power of a given schedule is left for future research.

The effects of the overall processor energy model on the proposed low-power scheduling algorithm will be discussed in detail in the following chapter. In brief,

the selection of per-cycle energy threshold values is heavily dependent upon the overall processor energy model in question, and the variance in individual FU energy values bears directly upon the effectiveness of the proposed scheduling approach.

2.3 Energy Models And The MDES Machine Description

In order to perform static scheduling for a VLIW processor, the compiler must have detailed information on all of the processor's hardware resources such as the type of each functional unit, the functional unit latencies, and the available scheduling options for each instruction type. In addition to this information, the compiler must also have knowledge of the energy dissipation characteristics of each functional unit in order to perform low-power scheduling. Therefore, a mechanism must exist which can relate all of this relevant information to the compiler. For the studies presented in this thesis, the MDES machine description language is used to interface with the LEGO compiler and provide the hardware descriptions necessary, [28], [29].

The MDES language was developed in order to provide a simple mechanism for defining new machine architectures for use in ILP architecture studies. The MDES language provides mechanisms for defining different types and numbers of functional units, registers and register files, reservation tables, operation types and different scheduling options for each type, etc. In addition, the MDES language and its associated compiler are designed in such a way as to allow the user to define new architectural features.

It is through MDES that the energy dissipation characteristics are defined and communicated to the compiler. Each type of functional unit that is defined in the MDES is given an associated energy dissipation value which corresponds to the average energy dissipated from that unit in a single cycle. For pipelined functional units we assume that the average energy value is dissipated during each individual cycle of the unit's execution latency. The energy values for each functional unit are stored in a table in the MDES internal representation, and an energy dissipation value is looked up from that table for each instruction that is scheduled. The interaction between the MDES and the compiler as well the low-power scheduling algorithm itself is the topic of the following chapter.

Chapter 3

Low-Power Scheduling

3.1 Fundamental Concepts

The overall goal of the instruction scheduling technique presented here is to limit the effects of high-energy instructions executing in parallel. If the compiler's instruction scheduler schedules instructions which execute on high-energy functional units in the same cycle and these instructions reside in heavily weighted blocks of code, then current spikes will occur repeatedly during that block's execution. It is these repeated current spikes which result in increased peak power dissipation in the processor's execution core.

In order to reduce the total power dissipation in the FUs, a per-cycle energy dissipation threshold is set inside the compiler so that the scheduler can schedule code such that the energy dissipation across multiple cycles is more uniformly distributed resulting in a more smooth di/dt curve across the execution time of the program. Although the same amount of total energy will be dissipated, it is the conjecture of this thesis that a more uniform distribution of energy dissipation across a program's

execution will result in a lower average peak power dissipation.

3.1.1 Schedule Slack

In addition to scheduling for reduced di/dt, care must be taken to ensure that program performance is maintained. In order to maintain reasonable levels of performance there must exist a sufficient amount of *schedule slack* in the *data dependence graph* (DDG) of each region being scheduled. Schedule slack provides the scheduler room to move instructions down in the schedule into free schedule slots which result from single, long dependence chains without negatively impacting the length of the schedule.

A simple example DDG with schedule slack is shown in Figure 3.1. In this example, the dependence chain from I1 to I3 determines the schedule length of the code. For this example, assume these six instructions constitute a single basic block and that I4, I5 and I6 are single-cycle latency instructions. Due to the dependence chain represented by I1, I2 and I3 and the lack of dependences within the block on I4, I5, or I6, the latter three instructions can be moved down in the schedule without affecting the total length of the schedule. Of course, this freedom of movement also depends upon resource constraints imposed by the architecture of the machine. The number and latency of each type of function unit within the machine will determine such constraints.

It is this phenomenon that provides the opportunity for moving instructions down
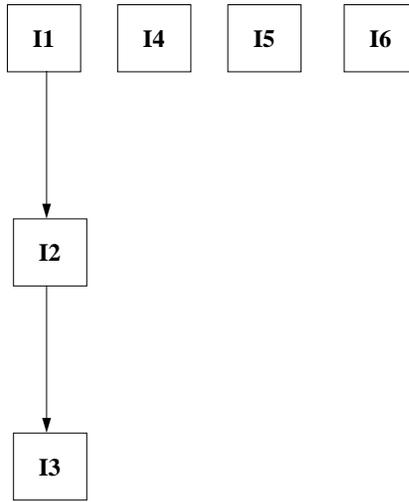
36

Figure 3.1: A simple example DDG showing schedule slack. Assuming single-cycle latencies for I4, I5, and I6, then according to the resource constraints of the machine, these three instructions may be scheduled during any of the cycles during which the dependence chain represented by I1, I2, and I3 is scheduled without affecting the total schedule length of the code represented in the graph.

in the schedule in order to minimize the amount of energy being dissipated in the processor functional units during any given cycle. However, in order for the proposed algorithm to capitalize on the schedule slack that does exist, there needs to be a set of instructions in each region which are not members of the schedule length determining dependence chain and can be moved into those empty slots in existence. Otherwise, there is little chance that energy violations will occur frequently enough to make low-power scheduling necessary.

3.1.2   Region Formation

From the above discussion it is evident that the degree to which the low-power scheduling algorithm can succeed is determined in part by the region formation algorithm employed by the compiler. Small, linear regions such as basic blocks do not provide the opportunities necessary for low-power scheduling to succeed. In fact, as will be demonstrated by the results in the upcoming section, basic blocks generally do not produce schedules which result in significant energy spikes. This is due to their limited size in terms of how many instructions reside in a typical basic block as well as the limited parallelism which exists between the contained instructions, [34]. However, speculation by the scheduler will result in increased parallelism, leading to more opportunities for low-power scheduling to be effective at the risk of overpopulating the block. The impact of speculation on low-power scheduling will be discussed in detail in later in this thesis.

Figure 3.2 shows an example of a typical basic block taken from the *126.gcc* benchmark. In this example, the schedule length of the block is determined by the dependence chain created by instructions *I1*, *I2*, *I4*, and *I5*. This leaves the remaining instruction, *I3*, as the only instruction in the block which has *slack*.

Table 3.1 shows the schedule of the code from Figure 3.2 on a four-issue, universal unit VLIW machine model. Each instruction in this example has a single-cycle latency with the exception of the load instruction (instruction *I2* in Figure 3.2) which has a latency of two cycles. The schedule shows that several empty slots exist as a result

```
I1: r24 = r3 + 2
I2: r7  = LD(r24)
I3: PBRR(btr25)
I4: p26 = CMPP(r7 = 6)
I5: BRCT(btr25) if p26
```

Figure 3.2: Typical basic block code example. This is a sample basic block taken from *126.gcc*. This figure shows that typical basic blocks are limited in size and, as a result, contain limited ILP, [34].

of the dependence chain, but the limited parallelism within the block results in little

chance that an energy threshold violation will occur unless the threshold is sufficiently

low. Thus, from this example, it is clear that low-power scheduling will not be as

effective on regions that are relatively small and linear in nature.

A more robust, non-linear region, called a *treegion*, was introduced in [32] and [33].

A treegion consists of a tree shaped subgraph of a program's *control flow graph* (CFG)

in which all the nodes are basic blocks. Thus, treegions contain multiple control

paths which stem from the *root* of the treegion. This tree shaped topology containing

independent control paths significantly increases the ILP seen by the scheduler and

results in higher performance schedules as compared to basic block and superblock

39

Table 3.1: An example basic block schedule. This table provides the scheduled code
for the basic block given in Figure 3.2 on a four-issue, universal unit VLIW machine.

Unit 0	Unit 1	Unit 2	Unit 3
r24 = r3 + 2	PBRR (btr25)		
r7 = LD (r24)			
p26 = CMPP (r7 = 6)			
BRCT (btr25) if p26			
Total schedule length = 5 cycles Schedule length determined by dependence chain through I1, I2, I4, and I5			

scheduling, [32]. One side effect of this increased performance is increased power
due to higher utilization in the processor's function units. However, due to the non-
linear nature of treegions and the increased availability of *slack* instructions from
the multiple, independent paths therein, treegions provide much more opportunity
for low-power scheduling optimizations. An example, treegion-formed CFG, as taken
from [32], is shown in Figure 3.3.

3.1.3 Energy Threshold Selection

The availability of schedule slack and the existence of a sufficient number of *slack*
instructions within each region are not the only factors which determine the success of
the low-power scheduling algorithm presented. A second factor, namely the per-cycle
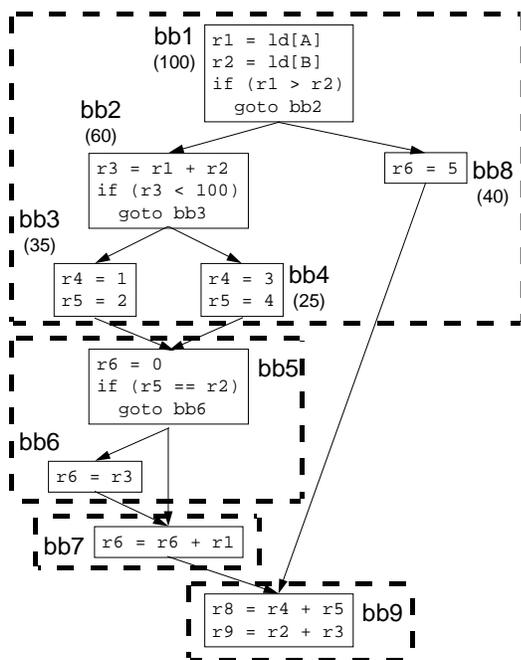
Figure 3.3: An example treegion-formed CFG. All treegions are marked by encompassing dotted lines.

energy dissipation threshold, also plays an integral role in determining the goodness of the algorithm.

Threshold selection not only affects the amount of savings in di/dt achieved by the algorithm, but it also bears directly on the amount of performance degradation seen for each program. More specifically, the energy threshold is indirectly proportional to both di/dt savings as well as performance loss. Therefore, it is imperative that threshold selection be done carefully and with knowledge of how much power must be saved and within what performance boundaries the algorithm must operate. A small threshold value will provide significant savings in di/dt but may also result in

performance degradation which is intolerable for the specific machine or application under consideration. The oposite case is also true. A relatively high threshold value will not impact performance seriously, but, at the same time, may not provide the power savings necessary.

A secondary issue which affects energy threshold selection is the distribution of energy dissipation values given for the various FUs in the processor. A wide range of energy values across the set of FUs is tolerable assuming the distribution is relatively normal. However, in the case where there is a single FU whose average energy dissipation significantly exceeds those for the remaining FUs may force the selection of a threshold value which will provide little to no savings in di/dt at all. This is due to the fact that the threshold value must be greater than or equal to the largest of the average energy dissipation values in the set of FU energy models. This inequality is expressed in Equation 3.1,

$$T_E \geq max_{e \in E}[e] \tag{3.1}$$

where T_E is the energy threshold and E is the set of all FU energy values. Given the situation where such an outlier exists, a secondary algorithm may be employed to handle the scheduling of instructions which must execute on the associated FU.

The energy models used in the low-power scheduling studies presented in this chapter were detailed in Table 2.1 in Chapter 2. Based on these models, four energy threshold values were selected for study - $7nJ$, $10nJ$, $12nJ$, and $15nJ$. These values

were selected based on preliminary studies which provided two indications - that the point of diminishing returns in terms of di/dt savings occurs at a point just above $15nJ$ per cycle and that the performance degradation seen by the benchmarks used was too excessive at thresholds below $7nJ$ per cycle.

The following sections detail the low-power scheduling algorithm, its implementation, and the results of the studies performed.

3.2   Low-Power Scheduling Algorithm

The low-power scheduling algorithm presented here is based upon the traditional *list scheduling* algorithm as presented in [30] and [31]. The low-power algorithm varies from list scheduling in that it performs energy dissipation checking before committing an instruction to the schedule. Figure 3.4 provides a pseudo-code representation of the base low-power scheduling algorithm.

The first step in the algorithm, once the ready-list has been established, is to query the MDES for on open FU of the appropriate type. If there are no FUs of the corresponding type, then the operation in question can not be scheduled in the current cycle, and the algorithm moves on to the next operation in the ready-list. Otherwise, the scheduler queries the MDES again to get the energy dissipation value associated with the FU returned by the initial query. This value is added to the current cycle total, which is kept by the scheduler, and the new value is compared to the energy threshold, T_E. If the new value is less than or equal to the threshold value,

```
 1 Region Formation
 2 for each (Region)
 3 {
 4    Generate DDG for Region
 5    low_power_schedule (DDG)
 6    {
 7        get first op in ready list
 8        while all ops in the ready list have not been scheduled
 9        {
10            while the next op is not the last in the ready list
11            {
12                if all dependences are satisfied
13                {
14                    query MDES for an open FU of the proper type
15                    if there is an open FU
16                    {
17                        query the MDES for the energy dissipation value of the FU
18                        add the returned energy value to the current cycle total
19                        if new cycle energy total exceeds cycle energy threshold
20                        {
21                            do not schedule op
22                            continue with next op in the ready list
23                        }
24                        else schedule the instruction in the current cycle
25                    }
26                }
27                continue with the next op in the ready list
28            }
29            increment the cycle counter and reset the op pointer to the
30            beginning of the ready list
31        }
32    }
33 }
```

Figure 3.4: Low-power scheduling algorithm.

then the operation is scheduled and scheduling proceeds with the next operation in
the ready-list. Otherwise, the operation is not scheduled and scheduling continues
with the next operation in the ready-list in order to check for lower energy consuming
operations.

Once all of the instructions in the ready-list have been examined for the current

cycle, the cycle counter is incremented, the operation pointer is reset to the instruction at the head of the ready-list, and scheduling resumes for the next cycle. In this manner, the compiler controls the amount of energy that is dissipated during the execution of each individual cycle creating a smoother di/dt curve across the entire program's execution.

Figures 3.5 and 3.6 demonstrate how the algorithm creates a smoother, more uniformly distributed energy curve for a piece of code. Both figures show the energy dissipated during a 120 cycle span of scheduled code from the *129.compress* benchmark. The first figure, Figure 3.5, provides results in the context of basic block scheduling, and the second figure, Figure 3.6, represents results for treegion scheduling. Clearly, the low-power algorithm effectively eliminates the large energy spikes which exceed the threshold value, which in this case is $7nJ$. It is these spikes that, if executed with large frequency, result in increased average peak power dissipation. Thus, by eliminating the spikes and creating a more uniform energy dissipation distribution, a reduced level of average peak power dissipation can be achieved.

The studies presented in the next two sections correspond to low-power scheduling based on basic blocks and treegions. The sorting algorithms used to sort the instructions in the two region types' DDGs are by *dependence height* and *global weight*, [32], for basic blocks and treegions, respectively. In the initial studies performed in this chapter, no special consideration is given to critical-path instructions during the sorting of the DDGs. The consderation of critical paths and their effect on low-power
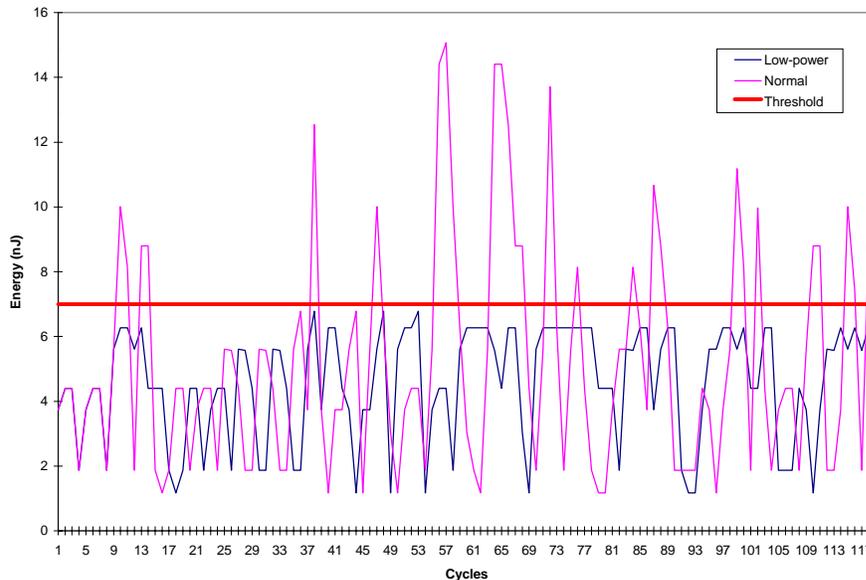
Figure 3.5: Energy distribution example from *129.compress* with basic block scheduling. This energy distribution sample was taken during a 120 cycle span.

scheduling is left as the subject of the next chapter.

3.3 Basic Block Scheduling Studies

In order to clearly demonstrate the dependence of low-power scheduling on the amount

of schedule slack and the number of slack instructions provided by a particular region

type, the first set of studies presented were performed using basic block scheduling.

The graphs in Figures 3.7 and 3.8 show the total savings in di/dt and the slowdown,
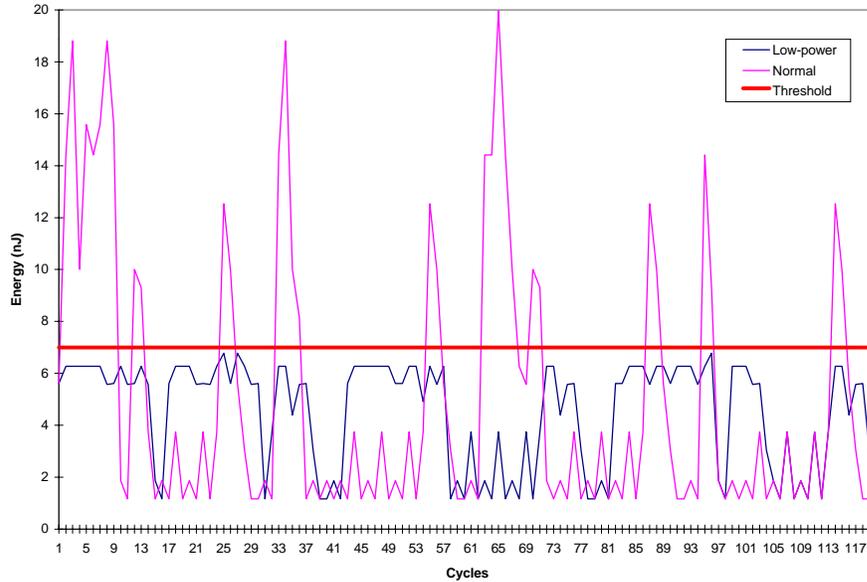
Figure 3.6: Energy distribution example from *129.compress* with treegion scheduling. This energy distribution sample was taken during a 120 cycle span.

respectively, for each benchmark. As mentioned in Chapter 1, the total savings in di/dt and the slowdown numbers for the low-power scheduling experiments are given as percentages of the total energy and total number of execution cycles, respectively, for normal list scheduling of the region type under examination.

By examing Figure 3.7 and considering the previous discussion on basic block characteristics, it is evident that basic block scheduling does not provide sufficient schedule slack and slack instructions to obtain significant savings in di/dt across the
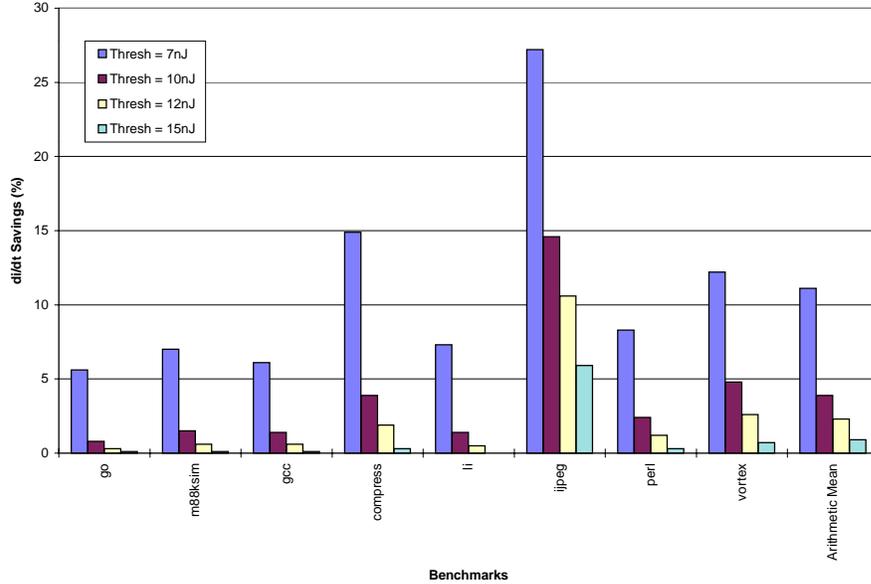
Figure 3.7: Total di/dt savings for basic block scheduling.

entire range of threshold values. The averages for di/dt savings and slowdown for each threshold are listed in Table 3.2. Slowdown results correspond closely to the results for di/dt savings. In addition to the effects of schedule slack on the results, an examiniation of the graphs also provides insight into how threshold selection affects the outcome of low-power scheduling.

The exception to the above generalizations with respect to di/dt savings and slowdown can be seen in the results for *132.ijpeg*. *132.ijpeg* seems to experience significant savings in di/dt as well as much larger slowdowns than the other benchmarks. This
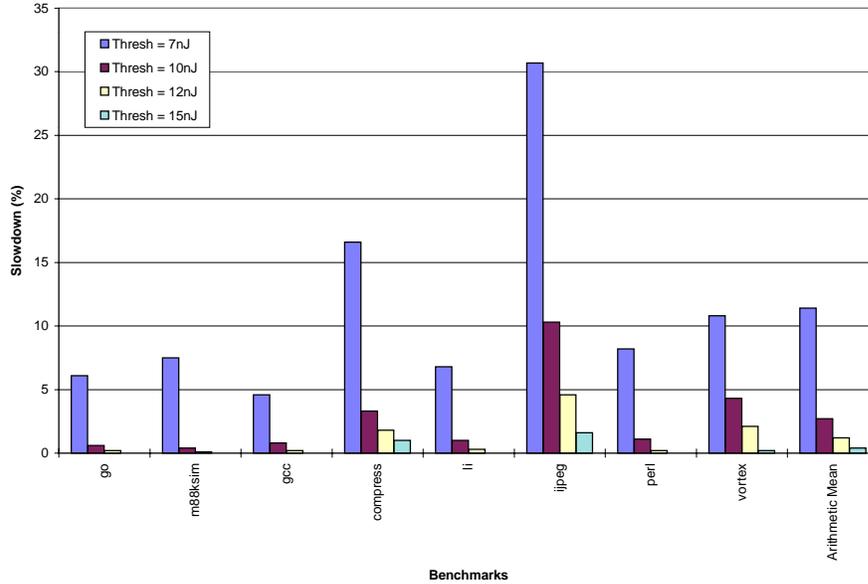
48

Figure 3.8: Total slowdown for basic block scheduling.

Table 3.2: Averages for di/dt savings and slowdown for basic block scheduling.

Threshold Value	Average di/dt Savings	Average Slowdown
$7nJ$	11.1%	11.4%
$10nJ$	3.9%	2.7%
$12nJ$	2.3%	1.2%
$15nJ$	0.9%	0.4%

can be explained by taking a close look at the code characterisitcs within the benchmark. *132.ijpeg* is an image compression/decompression program which manipulates images residing in main memory. Thus, the code basically spends its time in three

different loops - loading images form memory, image processing, and storing processed images back into memory. The key loops in terms of low-power scheduling are the image loading and storing loops because loads and stores dissipate the most energy in the machine model under consideration (refer to Table 2.1). Thus, the extra savings in di/dt are seen as a result of the repeated execution of high-energy loads and stores. The increased slowdown, in this case, is directly related. Energy violations due to loads and stores attempting to execute in parallel naturally force the delay of some instructions which would otherwise be scheduled in the current cycle during normal scheduling. Since, in the case of $132.ijpeg$, these delays involve non-unit latency instructions, the performance implications of these delays are amplified.

Figures 3.9 and 3.10 provide a different visualization of the results for basic block scheduling. These graphs show the distribution of di/dt savings and slowdown across the range of threshold values studied. The data here gives a better indication of how threshold selection affects both of these metrics. The most interesting portion on both of these graphs is the segment between the $10nJ$ and the $7nJ$ data points. Here a significant increase in the slope of both curves is seen. In the context of basic blocks, this behavior is what is expected. With respect to slowdown this behavior should be natural for any region. This generalization is not necessarily true with respect to savings in di/dt, but in the context of basic blocks it is. This is, again, due to the limited ILP in basic blocks. As previously mentioned, the limited parallelism between basic block instructions will prevent energy violations from occuring with any signifi-

cance when threshold values are large with respect to the processor's energy models. However, even in the context of the limited ILP in basic blocks, small threshold values will result in a fairly significant amount of energy violations. These energy violations will enhance the savings in di/dt with respect to that of higher threshold values.
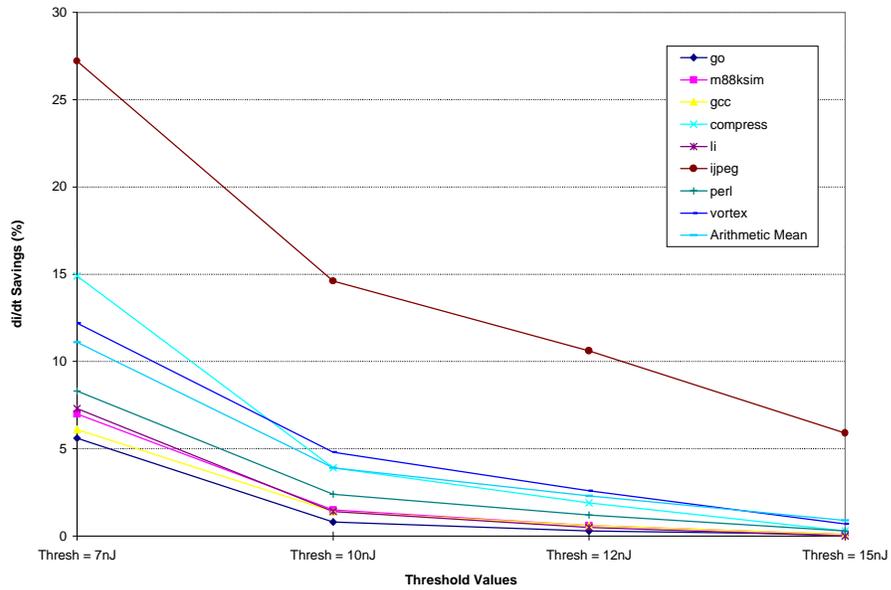

Figure 3.9: Distribution of di/dt savings for basic block scheduling across threhsolds.
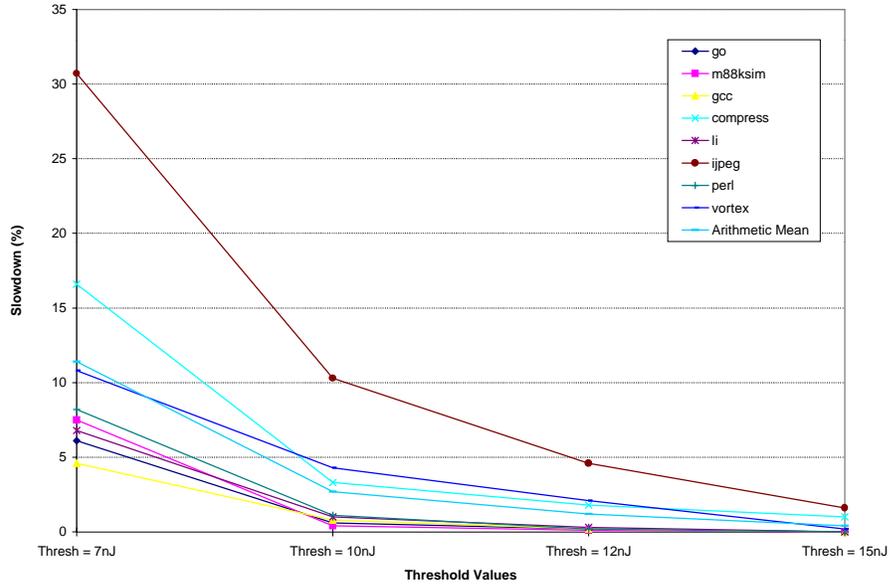
Figure 3.10: Distribution of slowdown for basic block scheduling across thresholds.

3.4 Treegion Scheduling Studies

The results from the previous section on basic block scheduling motivate the studies presented in this section. Low-power scheduling must be given the opportunity to move instructions around in order to achieve significant reductions in di/dt while still producing high-performance schedules. Basic blocks, due to their limited size and ILP, do not provide this opporunity. Treegions, on the other hand, provide large regions with instructions spanning multiple, independent control paths resulting in

significantly increased opportunities for low-power scheduling to move instructions down in the schedule without seriously affecting performance.

Figures 3.11 and 3.12 provide the di/dt savings and slowdown results, respectively, for treegion scheduling, and Table 3.3 provides a summary of the average results for both metrics across the four thresholds studied. From the results shown in Figure 3.11, a significant increase in the amount of di/dt savings available in treegions over that which is available in basic blocks can be seen immediately. This is primarily due to the increased ILP seen by the scheduler as a result of the topology of the regions being scheduled. This increase in ILP translates into a potential increase in energy spikes during execution exposing more opportunities for low-power scheduling to smooth out the dynamic di/dt curve for the region in question. This increased opportunity over basic block scheduling is demostrated clearly in Figures 3.5 and 3.6. From these graphs it can be seen that the increased ILP in treegions results in larger energy spikes during normal scheduling allowing for increased savings when low-power scheduling is enforced.

Table 3.3: Average for di/dt savings and slowdown for treegion scheduling.

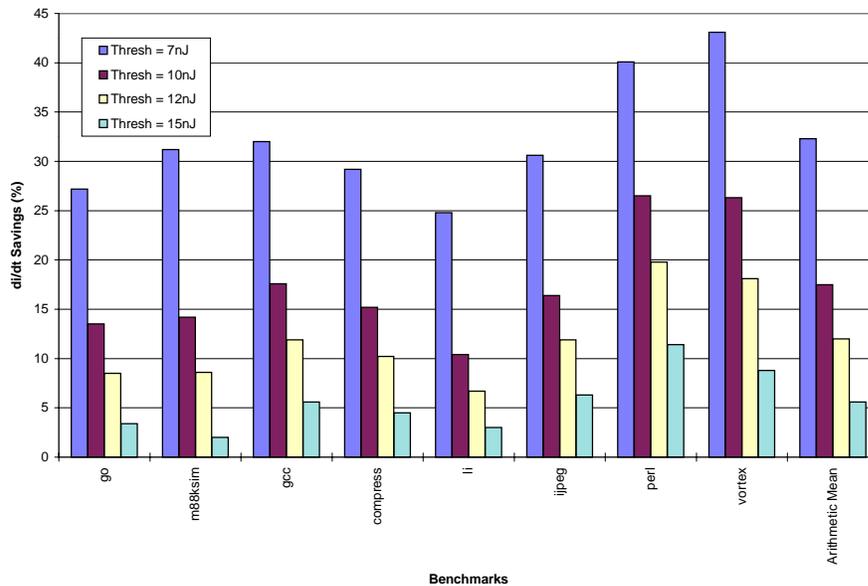| Threshold Value	Average di/dt Savings	Average Slowdown
$7nJ$	32.3%	19.6%
$10nJ$	17.5%	6.9%
$12nJ$	12.0%	2.7%
$15nJ$	5.6%	0.6%

Figure 3.11: Total di/dt savings for treegion scheduling.

The slowdowns presented in Figure 3.12 are somewhat surprising at first glance, especially with respect to the smallest two threshold values. However, these results can be explained with a detailed analysis of how the base implementation of low-power scheduling handles critical paths in treegions. The most obvious issue affecting slowdown is the disregard for the critical path through any treegion inherent in the low-power scheduling algorithm. The implementation discussed to this point simply proceeds down the ready-list, scheduling as many instructions as possible in each cycle based on the energy threshold constraints. In this manner, critical-path instructions
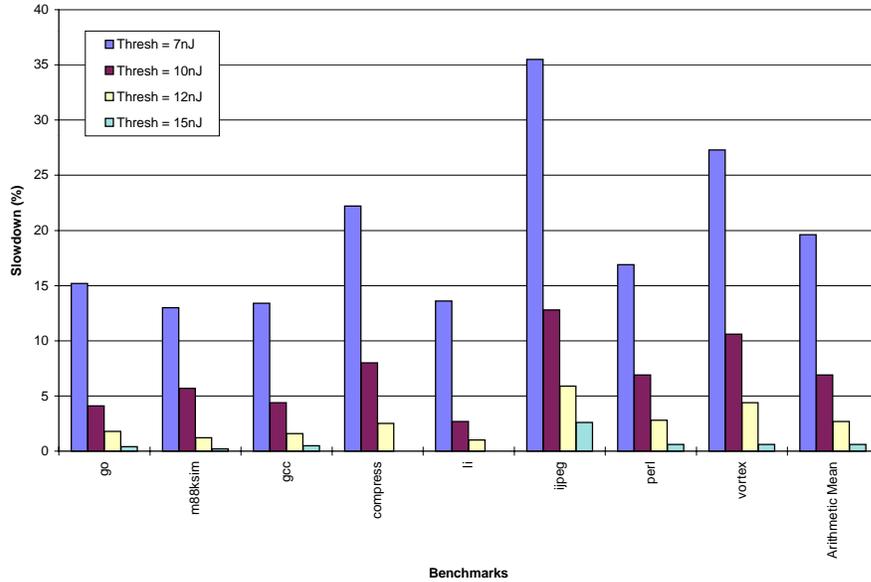
Figure 3.12: Total slowdown for treegion scheduling.

are just as likely to be delayed as any other instructions in the treegion. Ideally, the scheduler would have prior knowledge as to which instructions in each region have slack. In such a case, the scheduler could limit the movement of instructions down in the schedule to those instructions which have slack and will not result in the unnecessary delay of critical-path instructions. Obviously, the current situation is an undesirable one, but one that can be remedied given a framework which can locate and make visible to the scheduler the critical paths in the code.

In a similar fashion to the previous section, Figures 3.13 and 3.14 provide the
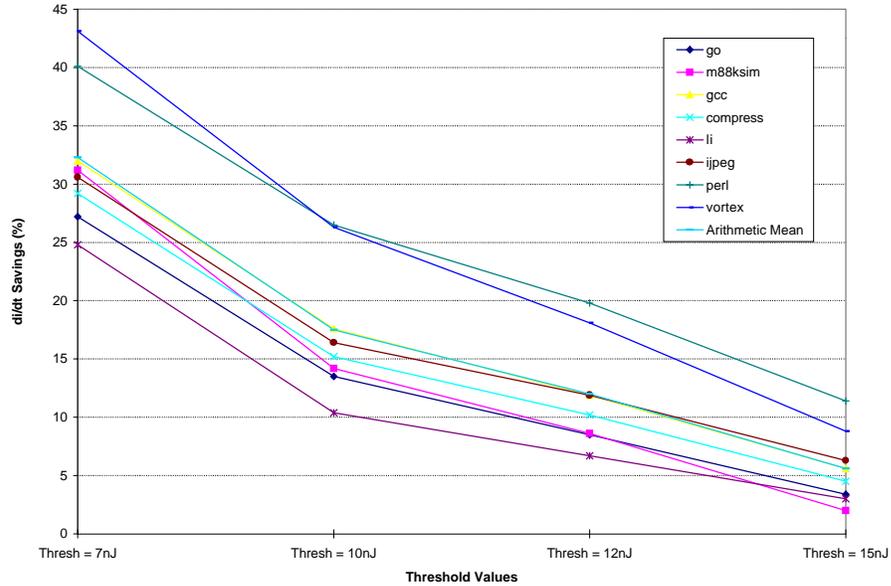
Figure 3.13: Distribution of di/dt savings for treegion scheduling across thresholds.

distributions of the savings in di/dt and slowdown across the range of threshold values studied. Of particular interest in these graphs is the notably less abrupt change in slope between the $10nJ$ and $7nJ$ data points for the di/dt savings plot. This is indicative of the increased opportunities treegions provide for savings in di/dt at the higher threshold values.

In summarizing the results from this section, one major issue was noted which needs to be addressed in order to achieve the maximum benefits from low-power, treegion scheduling. An accurate framework for identifying critical paths in treegions
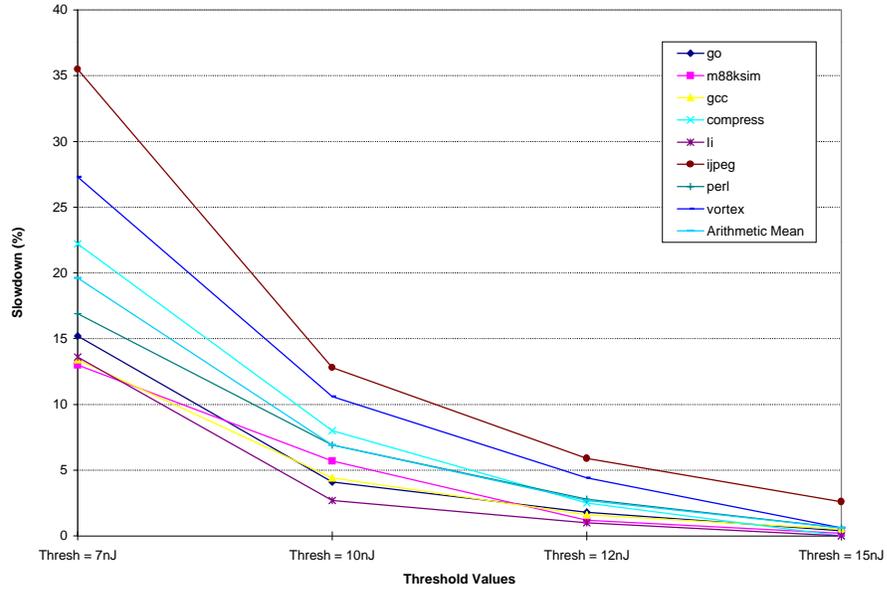
Figure 3.14: Distribution of slowdown for treegion scheduling across thresholds.

is necessary to ensure that only those instructions that have slack and do not reside along the treegion's critical path are moved down to accomodate energy constraints. The following chapter deals with this issue and proposes a technique for finding and marking the critical paths in treegions. In addition, a more complex technique is proposed and discussed.

3.5 Low-Power Scheduling Comparison - Treegions vs Basic Blocks

The previous two sections provided the initial results from the low-power scheduling studies performed on basic blocks and treegions, respectively. In this section, the results will be combined and compared so as to solidify the claim that large, multi-path regions must be employed in order for the proposed low-power scheduling algorithm to be effective.
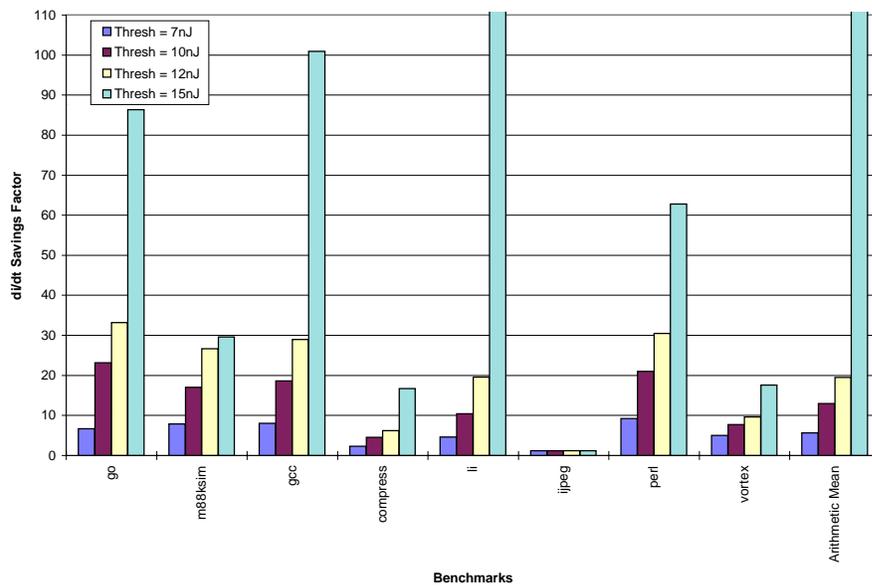

Figure 3.15: Increase in di/dt savings for low-power treegion scheduling over low-power basic block scheduling.

The results reflecting di/dt savings from the previous two sections clearly show that treegions provide more opportunity for increased energy savings as a result of their non-linear shape, increased ILP, and increased schedule slack over that of basic blocks. However, the magnitude of this increase is not fully conveyed in the previous graphs. Figure 3.15 presents this information in a much clearer fashion. The y-axis in the graph represents the factor by which the di/dt savings increases for treegion scheduling over basic block scheduling for each benchmark and each threshold value. From the graph it can be seen that low-power scheduling typically provides energy savings which are orders of magnitude larger when scheduling in the context of treegions as opposed to basic blocks. Note that the $15nJ$ bars for *130.li* and the *arithmetic mean* extend past the top of the graph. The values for these bars are 6008 and 790.4, respectively.

Likewise, the slowdown results given in the previous section for low-power treegion scheduling do not convey the full picture. When comparing Figures 3.12 and 3.8, it might be interpreted that slowdown for treegion scheduling is worse than that of basic block scheduling. It is important to note, however, the inherent performance improvements that treegions provide over basic blocks, [32]. In addition, it should also be noted that Figure 3.12 shows slowdown with respect to normal treegion scheduling and not with respect to normal basic block scheduling or low-power basic block scheduling. As a proper performance comparison, Figure 3.16 shows the speedup provided by low-power treegion scheduling over low-power basic block scheduling.

59
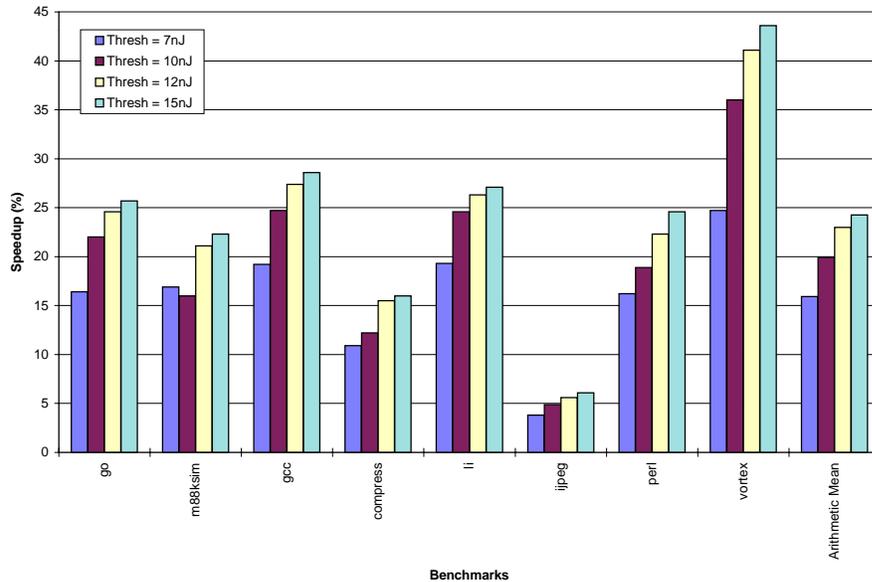
Figure 3.16: Speedup for low-power treegion scheduling over low-power basic block scheduling.

The above figures concisely demonstrate the dependence of the effectiveness of low-power scheduling on the region formation algorithm employed by the compiler. In addition, they also demostrate the effectiveness of treegions with respect to the provision of schedule slack allowing significant energy savings while maintaining reasonably high levels of performance.

60

Chapter 4

Critical-Path-First Scheduling

The results from the studies on low-power, treegion scheduling in Chapter 3 demonstrate that the base implementation of low-power scheduling is fundamentally flawed in that it disregards the region's critical path when making decisions on which instructions to delay upon incurring an energy violation. This chapter deals with this issue first by proposing a mechnism, called *critical-path-first scheduling*, which identifies critical paths in treegions based on flow dependence height and then makes the scheduler aware of these paths by giving them highest priority in the ready-list. Later in the chapter, an additional, and possibly more accurate, mechanism for critical-path identification is discussed.

4.1   Critical-Path-First Implementation and Initial Results

The critical-path-first approach uses flow dependence information to determine the existence of critical paths in treegions. This is accomplished by analyzing a treegion's DDG, which is implemented as a *directed acyclic graph* (DAG), to identify the existing flow dependences. The critical-path through a treegion is the path which

has the highest flow dependence height. Once the critical-path through a treegion has been identified, the DAG is resorted with the critical-path nodes having highest priority[1]. The scheduler uses the resorted DAG, and performs low-power scheduling according to the algorithm given in Figure 3.4. By having the critical-path instructions at the head of the ready-list, the scheduler checks those instructions first when scheduling, which ensures that critical-path instructions are not unnecessarily delayed when energy violations occur during scheduling. Figures 4.1 and 4.2 show the savings in di/dt and slowdown, respectively, for the critical-path-first implementation. In addition, Table 4.1 provides the averages for savings in di/dt and slowdown for each threshold value.

Table 4.1: Averages for di/dt savings and slowdown for critical-path-first treegion scheduling.

Threshold Value	Average di/dt Savings	Average Slowdown
$7nJ$	32.7%	25.0%
$10nJ$	17.9%	9.3%
$12nJ$	12.2%	4.0%
$15nJ$	5.9%	1.5%

The slowdown results given in Figure 4.2 and Table 4.1 are quite surprising in that they show that the slowdowns for the critical-path-first experiments have increased

---

[1]It should be noted that all instructions off the critical path remain sorted according to the global weight heuristic defined in [32] as stated in the previous chapter.
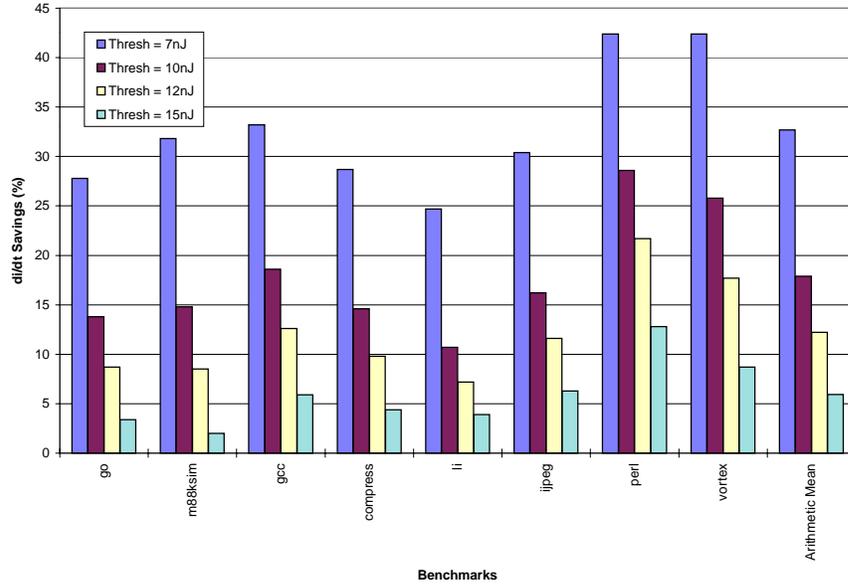
Figure 4.1: Total di/dt savings for critical-path-first scheduling.

over those for the base implementation experiments with treegions. This result is bothersome because the primary intent of the critical-path-first approach is to ensure that slowdown is kept to an absolute minimum by giving highest priority to critical-path instructions. However, although this result is counter-intuitive, it can be explained by examining how speculation affects the two implementations.
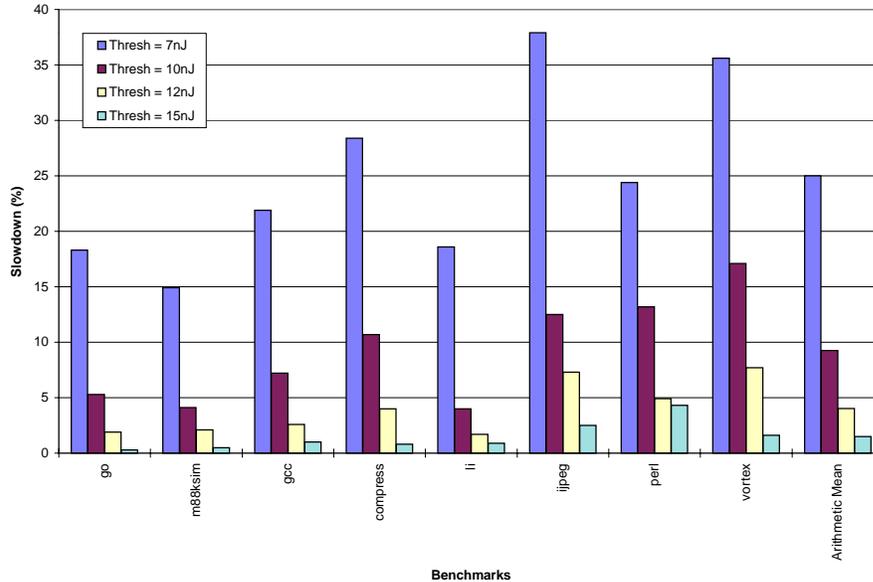
Figure 4.2: Total slowdown for critical-path-first scheduling.

4.2 Analysis of Speculation Effects on Critical-Path-First Approach

The current implementation of the LEGO instruction scheduler aggressively performs speculation at every possible opportunity. While this provides a significant performance boost for normal list scheduling, it is a potential problem for a constraint-driven algorithm such as low-power scheduling. The constraint-driven nature of low-power scheduling naturally limits the amount of speculation which may be performed by the scheduler due to the delaying of instructions to meet energy requirements. However,

64

it was found that the prioritization of critical-path instructions in the critical-path-first approach led to increased speculation over the base low-power implementation. Figure 4.3 shows the increase in the number of speculated instructions in the root basic blocks of treegions for the critical-path-first studies over the base implementation studies. An interesting result in Figure 4.3 are the numbers given for *124.m88ksim*. These numbers, in combination with those given in Figure 4.4, give the indication that overspeculation is not necessarily the only reason for the increased slowdown seen by the critical-path-first approach. However, it is important to note that Figures 4.3 and 4.4 reflect only the increases in root basic blocks. A more encompassing statistical analysis of the total number of speculated instructions across all blocks in *124.m88ksim* shows that there is a total overall increase in the number of speculated instructions for critical-path-first scheduling over the base implementation. Experimental results show that the increase in the total number of speculated instructions in all blocks of *124.m88ksim* ranges from 84.4% for the $7nJ$ threshold to 3.3% for the $15nJ$ threshold.

The overspeculation of instructions into higher, more heavily weighted blocks in a treegion overpopulates the empty slots within those blocks, preventing the movement of slack instructions into empty slots to accomodate the constraints of low-power scheduling. In addition, this overpopulation may lead to an increase in the number of energy violations, resulting in further slowdown due to instruction delays necessary to meet energy requirements. Figure 4.4 shows the increase in schedule length for the
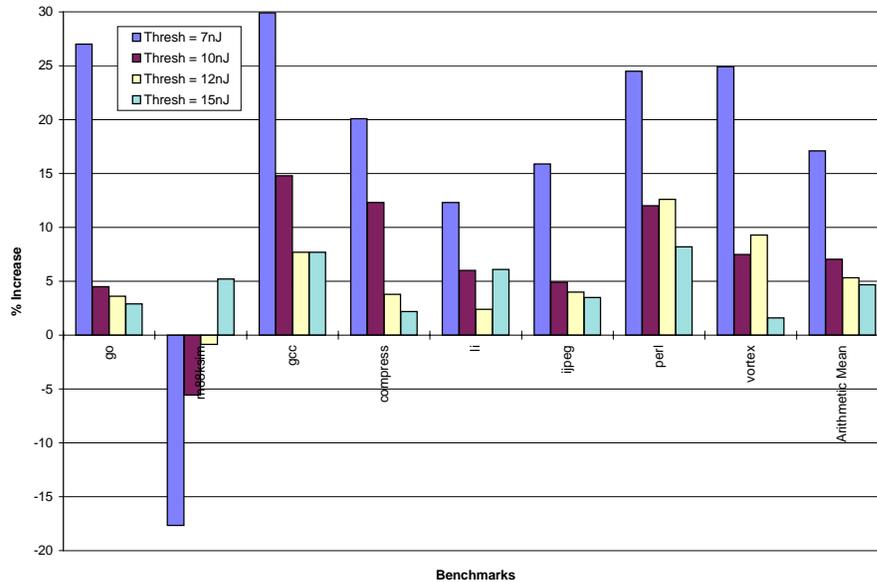
Figure 4.3: Increase in speculated instructions in treegion root blocks for critical-path-first scheduling over base implementation.

root basic blocks in all treegions for each benchmark.

4.3 Alternative Mechanism for Prioritizing Critical Paths

One fundamental problem with the critical-path-first approach presented in the previous section is that it allows for the identification and prioritization of potentially multiple critical paths in a single treegion. This is a result of the use of flow dependence information to identify critical paths. The problem that arises is that critical

66
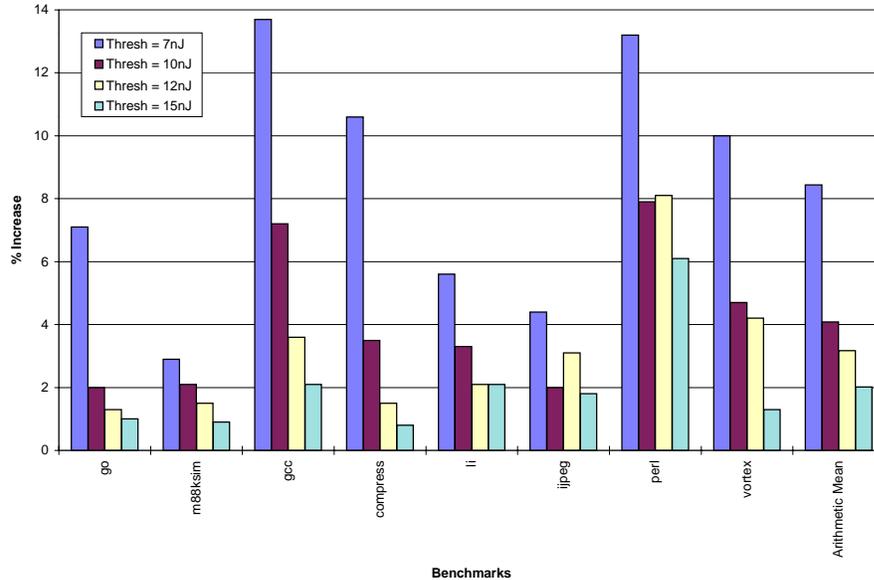
Figure 4.4: Increase in schedule length of treegion root blocks due to overspeculation.

paths which have little to no weight in terms of execution are given equally high priority in the ready-list as those instructions which reside along the most heavily weighted critical path. Ideally, only the most heavily weighted critical path would be given priority in the ready-list. However, without accurate profile information or resource constraint information, it is difficult to statically single out the single most heavily weighted critical path through any treegion.

Ideally, the dominant critical path would be identifiable through resource constraints as well as flow dependences. However, a resource constrained critical path is

not easily identifiable during a single scheduling pass. However, it might be possible to obtain such information by implementing a two-pass scheduler which operates as follows: 1) schedule the original code using basic blocks, 2) register allocate to establish full resource allocations, 3) form treegions based on the register allocated code from the first scheduling pass and determine the rource-constrained critical path, 4) perform second-pass scheduling according to the critical-path-first approach using the critical path from step 3. In this manner, more realistic critical paths can be determined in terms of the resource constraints of the machine as well as the original flow dependences in the code. However, even implementing this approach does not guarantee the existence of a single dominant critical path at the end of step 3. It is quite possible that equally long and constrained paths may still exist in a single treegion. In such a case, accurate profile information may aid in determining the single, most heavily weighted of those paths.

Indeed, this is just one of many optimizations to the algorithms presented in this and the previous chapter. There is much work to be done on enhancing these algorithms as well as improving the accuracy and applicability of the energy models used by the scheduler. Such improvements and advancements are discussed in more detail in the following chapter.

Chapter 5

Conclusions and Future Work

5.1   Conclusions

Power dissipation is a growing concern for manufacturers and designers of high-performance microprocessors. As clock speeds and transistor densities continue to rise, reliability and cooling are becoming major issues due to the corresponding increase in heat being dissipated. This thesis has presented a novel scheduling algorithm for statically scheduled VLIW architectures which eliminates current spikes resulting from the parallel execution of instructions on high-energy function units during program execution by limiting the amount of energy which can be dissipated in the processor core during a single cycle. By eliminating such spikes, a more uniform distribution of energy dissipation is created allowing for reduced peak power dissipation from the processor core.

The fundamental concepts and issues regarding low-power scheduling as presented in Chapters 2 and 3 were the process of modeling the processor's function units in terms of their energy dissipation, the proper selection of the per-cycle energy threshold

to obtain the desired energy savings and to control the performance degradation of the implementation, and the provision of schedule slack by the type of region used during compilation and scheduling. The results presented in Chapters 3 and 4 show that the low-power scheduling algorithm provides significant savings in di/dt in the execution core at the cost of varying levels of slowdown in execution time. The degree of energy savings and slowdown incurred are totally controlled by the user and can be easily manipulated by careful threshold selection.

5.2 Future Work

Throughout this thesis, several issues regarding needed future work have arisen. The first avenue of future work which needs to be further developed in order to obtain more accurate schedules in terms of their energy dissipation is the development and application of more detailed energy models. Specifically, energy characteristics need to be supplied to the scheduler at a finer grain. Energy models should be applied at the instruction level as opposed to the function unit level in order to obtain more accuracy. For example, instead of having a single average energy dissipation value associated with an integer ALU, each ALU operation type should have its own energy dissipation value represented in the MDES. This is necessary because energy dissipation characteristics vary greatly from circuit to circuit, and a multiplier in a typical ALU will not have energy characteristics in the same range as an adder. The accuracy of the energy models could be taken a step further to include operand switching

information, but this may hinder the speed of the overall scheduling process unless highly accurate profile information is available.

Second, more work needs to be done towards balancing the effects of speculation in the context of low-power scheduling. As was demonstrated in Chapter 4, over-speculation can result in increased slowdown which otherwise might be avoided by properly limiting speculation. The slowdown results given in Figure 4.2 do not even convey the full picture. These results represent speculative code increase before register allocation. In the context of register allocation, overspeculation will also result in excess spill and fill code due to the potential overlapping of live ranges. This effect is especially evident in machines which contain limited register resources.

A final avenue for potential future work regarding low-power scheduling is more optimal selection of instructions to delay in the context of energy threshold violations. It should be possible to analyze the DDG of each treegion in order to determine which instructions in the graph have slack and exactly how long that slack is. This could be accomplished through simple def-use analysis of the graph. This information could then be used to more effectively guide the instruction scheduler in deciding which instructions to delay and by how much those instructions can be delayed before certain performance impact is incurred.

In addition to future work regarding low-power scheduling, additional work in the general area of low-power architecture and code optization is called for. Of immediate interest is the concept of dynamic monitoring and control of power dissipation for both

superscalar and VLIW architectures. Techniques such as throttling the instruction fetch and issue pipeline stages and dynamic rescheduling for low power are the types of ideas which need further investigation. In addition, more work needs to done in the area of high-level power anaylsis and estimation. Architects and designers need to know as early as possible whether a specific hardware scheme is pheasible, not only in terms of area and delay, but now power as well. Indeed, the low-power architecture community has a great deal of work to do and a great deal to learn about this young field of research.

Bibliography

[1] T. Burd, *CPU Info Center, http://infopad.eecs.berkeley.edu/CIC/*.

[2] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. Reading, MA: Addison-Wesley, 1993.

[3] M. Horowitz, T. Indermaur and R. Gonzalez, "Low-power digital design," *Proc. Int'l Symp. on Low Power Electronics*, vol. 8, pp. 8–11, Oct. 1994.

[4] A. Chandrakasan, et al, "Low-Power CMOS digital design," in *IEEE Journal of Solid State Circuits*, vol. 27, pp. 473–484.

[5] M. K. Gowan, L. L. Biro and D. B. Jackson, "Power considerations in the design of the Alpha 21264 microprocessor," in *Proc. 35th Ann. Design Automation Conference*, (San Francisco), June 1998.

[6] D. W. Bailey and B. J. Benschneider, "Clocking design and analysis for a 600-MHz Alpha microprocessor," in *IEEE Journal of Solid State Circuits*, vol. 33, no. 11, pp. 1627–1633, Nov. 1998.

[7] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," in *IEEE Journal of Solid State Circuits*, vol. 31, no. 9, pp. 1277–1284, Sept. 1996.

[8] C. L. Su and A. M. Despain, "Cache designs for energy effieciency," in *Proc. 28th Hawaii Int'l Conference on System Sciences*, vol. 1, pp. 306–315, 1995.

[9] J. Kin, M. Gupta and W. H. Mangione-Smith, "The filter cache: An energy efficient memory structure," in *Proc. 30th Int'l Symp. on Microarchitecture*, (Raleigh, NC), Dec. 1997.

[10] S. Manne, A. Klauser, D. C. Grunwald, and F. Somenzi, "Low power TLB design for high performance microprocessors," in Technical Report, Department of Electrical and Computer Engineering, University of Colorado at Boulder, Boulder, CO, 1997.

[11] G. Albera and R. I. Bahar, "Power and performance tradeoffs using various cache configurations," in *Proc. Power Driven Microarchitecture Workshop* in conjunction with *25th Ann. Int'l Symp. on Computer Architecture*, (Barcelona, Spain), June 1998.

[12] K. Ghose and M. B. Kamble, "Energy efficient cache organizations for superscalar processors," in *Proc. Power Driven Microarchitecture Workshop* in conjunction with *25th Ann. Int'l Symp. on Computer Architecture*, (Barcelona, Spain), June 1998.

[13] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: Speculation control for energy reduction," in *Proc. 25th Ann. Int'l Symp. on Computer Architecture*, (Barcelona, Spain), June 1998.

[14] C. L. Su, C. Y. Tsui, and A. M. Despain, "Low power architecture and compilation techniques for high-performance processors," in *Proc. IEEE COMPCON*, pp. 489–498, 1994.

[15] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," presented at *1994 Symp. on Low-Power Electronics*, 1994.

[16] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," in *IEEE Trans. on VLSI Systems*, pp. 437–445, 1994.

[17] M. T. C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and low-power scheduling techniques for embedded DSP software," presented at *1995 Int'l Symp. on System Synthesis*, 1995.

[18] V. Kathail, M. Schlansker, and B. R. Rau, "HPL PlayDoh architecture specification: version 1.0," Tech. Rep. HPL-93-80, Hewlett-Packard Laboratories, Technical Publications Department, 1501 Page Mill Road, Palo Alto, CA 94304, Feb. 1994.

[19] F. Najm, "A survey of power estimation techniques in VLSI circuits", in *IEEE Trans. on VLSI Systems*, vol. 2, no. 4, pp. 446–455, Dec. 1994.

[20] M. A. Crit, "Estimating dynamic power consumption of CMOS circuits", in *IEEE Int'l Conference on Computer-Aided Design*, pp. 534–537, Nov. 1987.

[21] F. Najm, R. Burch, P. Yang, and I. Hajj, "CREST - a current estimator for CMOS circuits", in *IEEE Int'l Conference on Computer-Aided Design*, pp. 204–207, Nov. 1988.

[22] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits", in *29th Ann. Design Automation Conference*, pp. 253–259, June 1992.

[23] J. Satyanarayana and K. K. Parhi, "HEAT: Hierarchical Energy Analysis Tool", in *Proc. 33rd Ann. Design Automation Conference*, (Las Vegas), June 1996.

[24] R. Burch, F. Najm, P. Yang, and T. Trick, "McPower: A Monte Carlo approach to power estimation", in *IEEE/ACM Int'l Conference on Computer-Aided Design*, pp. 90–97, Nov 1992.

[25] X. Xakellis and F. Najm, "Statistical estimation of the switching activity in digital circuits", in *31st Ann. Design Automation Conference*, pp. 728–733, 1994.

[26] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering", in *33rd Ann. Design Automation Conference*, pp. 702–707, June 1996.

[27] M. Reilly, Compaq Alpha Products Group, Personal communication, May 1998.

[28] J. C. Gyllenhaal, *A Machine Description Language for Compilation.* MS Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1994.

[29] J. C. Gyllenhaal, W. W. Hwu, and B. R. Rau, "HMDES version 2.0 specification," in Technical Report IMPACT-96-3, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1996.

[30] M. Johnson, *Superscalar Microprocessor Design.* Englewood, CA: PTR Prentice Hall, Inc., 1991.

[31] S. M. Muchnick, *Advanced Compiler Design and Implementation.* San Francisco: Morgan Kaufmann Publishers, 1997.

[32] W. A. Havanki, *Treegion Scheduling for VLIW Processors.* MS Thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1997.

[33] W. A. Havanki, S. Banerjia, and T. M. Conte, "Treegion scheduling for wide-issue processors," in *Proc. 4th Int'l. Symp. on High Performance Computer Architecture,*(Las Vegas), Feb., 1998.

[34] N. P. Jouppi and D. W. Wall, "Available instruction-level parallelism for super-scalar and superpipelined machines," in *Technical Report 89.7*, Western Research Labs, Digital Equipment Corporation, Palo Alto, CA, July, 1989.