

2003

Power considerations for memory-related microarchitecture designs

Zhichun Zhu

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Zhu, Zhichun, "Power considerations for memory-related microarchitecture designs" (2003).

Dissertations, Theses, and Masters Projects. Paper 1539623427.

<https://dx.doi.org/doi:10.21220/s2-az56-s960>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

Power Considerations for Memory-related Microarchitecture Designs

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William & Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

by

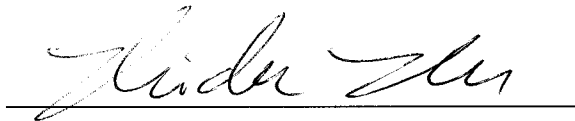
Zhichun Zhu

2003

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

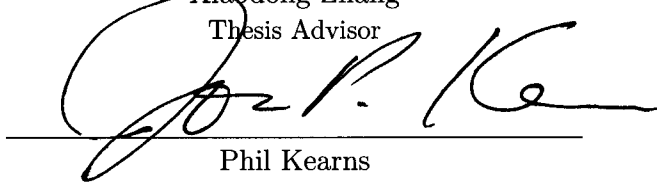


Zhichun Zhu

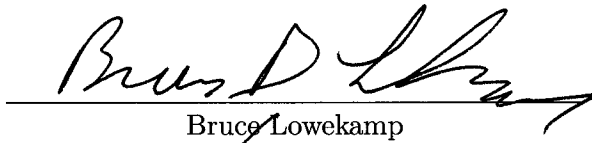
Approved, July 2003



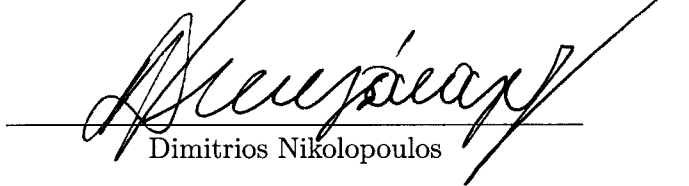
Xiaodong Zhang
Thesis Advisor



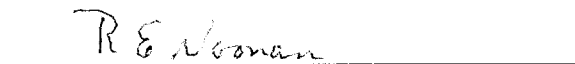
Phil Kearns



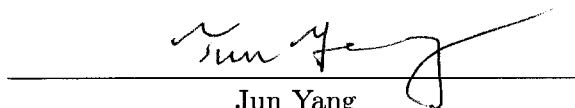
Bruce Lowekamp



Dimitrios Nikolopoulos



Robert Noonan



Jun Yang
University of California, Riverside

Table of Contents

| | |
|---|-------------|
| Acknowledgments | vii |
| List of Tables | viii |
| List of Figures | x |
| Abstract | xiii |
| 1 Introduction | 2 |
| 1.1 Importance of Low-power Architecture Designs | 3 |
| 1.2 Our Work | 4 |
| 2 Background | 8 |
| 2.1 Sources of Power Consumption | 8 |
| 2.2 Low-power Techniques | 9 |
| 2.2.1 Power-saving Techniques at Physical, Circuit and Logic Levels . . . | 10 |
| 2.2.2 Power-saving Techniques at Architectural Level | 11 |
| 2.2.3 Power-saving Techniques at Software Level | 12 |

| | | |
|----------|--|-----------|
| 3 | Evaluation Environment | 15 |
| 3.1 | Evaluation Metrics | 15 |
| 3.1.1 | Performance Metrics | 15 |
| 3.1.2 | Power and Energy Metrics | 16 |
| 3.1.3 | Energy-efficiency Metrics | 17 |
| 3.2 | Performance Evaluation Tools | 18 |
| 3.3 | Energy Consumption Evaluation Tools | 21 |
| 3.4 | Workloads | 22 |
| 4 | Access Mode Predictions for Low-power Cache Design | 25 |
| 4.1 | Significance of Low-power Cache Design | 25 |
| 4.2 | Related Work | 27 |
| 4.3 | Comparisons between Phased and Way-prediction Caches | 28 |
| 4.4 | AMP Caches | 31 |
| 4.4.1 | Strategy | 31 |
| 4.4.2 | Power Consumption | 33 |
| 4.5 | Access Mode Predictors | 34 |
| 4.5.1 | Designs | 34 |
| 4.5.2 | Accuracy | 37 |
| 4.5.3 | Overhead | 38 |
| 4.6 | Multicolumn-based Way-prediction | 39 |
| 4.6.1 | Limitation of MRU-based Way-prediction | 39 |
| 4.6.2 | Multicolumn Cache | 42 |

| | | |
|----------|---|-----------|
| 4.6.3 | Power Considerations for Multicolumn Cache | 45 |
| 4.7 | Experimental Environment | 49 |
| 4.8 | Experimental Results | 50 |
| 4.8.1 | Comparisons of Multicolumn and MRU Caches | 50 |
| 4.8.2 | Energy Reduction of AMP Caches | 53 |
| 4.8.3 | Energy-efficiency of AMP Caches | 59 |
| 4.9 | Summary | 62 |
| 5 | Look-ahead Adaptation Techniques to Reduce Processor Power Consumption | 64 |
| 5.1 | Motivation | 64 |
| 5.2 | Load Indicator Scheme | 68 |
| 5.2.1 | Power Saving Opportunity | 68 |
| 5.2.2 | Load Indicator | 70 |
| 5.3 | Considerations for Load Indicator Scheme | 75 |
| 5.4 | Experimental Environment | 78 |
| 5.4.1 | Power Savings | 80 |
| 5.5 | Effectiveness of Load Indicator Scheme | 82 |
| 5.6 | Combining Local and Look-ahead Optimizations | 85 |
| 5.6.1 | Motivation | 85 |
| 5.6.2 | Load-instruction Indicator | 88 |
| 5.7 | Comparisons between Different Schemes | 90 |
| 5.7.1 | Power Savings | 90 |

| | | |
|----------|---|------------|
| 5.7.2 | Performance Impact | 92 |
| 5.7.3 | Energy Reduction | 93 |
| 5.7.4 | Different Configurations | 94 |
| 5.7.5 | Average Intervals in Each Power Mode | 96 |
| 5.8 | Load-register Indicator Scheme | 97 |
| 5.9 | Effectiveness of Load-register Scheme | 100 |
| 5.9.1 | Power Savings | 100 |
| 5.9.2 | Energy Savings | 102 |
| 5.10 | Related Work | 103 |
| 5.11 | Summary | 105 |
| 6 | Conclusion and Future Work | 107 |
| 6.1 | Conclusion | 107 |
| 6.2 | Future Work | 109 |
| | Bibliography | 112 |

ACKNOWLEDGMENTS

Pursuing this degree is a long journey for me. I have learned a lot during this journey, not only from the study, the research work, but also from many people including my adviser, my professors, and my fellow students. I would like to thank all of them for their help. First of all, I want to thank my adviser, Dr. Xiaodong Zhang, for his mentoring during these five years. He created an excellent research environment for his students by his guidance, hard working, determination, high standard, and vast knowledge.

The Department of Computer Science at the College of William and Mary is a wonderful place for me to pursue my Doctoral degree. I would like to thank all faculty and staff in this department for their help in these years. I would like to acknowledge Dr. Phil Kearns, Dr. Bruce Lowekamp, Dr. Dimitrios Nikolopoulos, and Dr. Robert Noonan for serving the dissertation committee. I would also like to thank Dr. William Bynum, Dr. Evgenia Smirni, and Dr. Andreas Stathopoulos for serving the proposal committee. Especially, I thank Dr. William Bynum for reviewing many of my manuscripts. I want to thank Dr. Evgenia Smirni, Dr. Andreas Stathopoulos, and Dr. Dimitrios Nikolopoulos for their great help and advices in my career development. I would also like to acknowledge Vanessa Godwin for the great help she has offered to me in my graduate study.

I would like to thank the members of the High Performance Computing and Software Lab and other graduate students, Songqing Chen, Xin Chen, Lei Guo, Song Jiang, and Qi Zhang, who have helped to create an environment full of stimulation. I also want to thank Dr. Xing Du for his great help in my research.

I want to thank Dr. Jun Yang from University of California at Riverside, to take time from her busy schedule to serve as the external member of the committee. I would also like to acknowledge Dr. Yiming Hu from University of Cincinnati, for his kind help in my career development.

I would also like to acknowledge the funding agencies that provided funds and equipment that supported my research: Sun Microsystems, for their donations of experimental equipments, and the National Science Foundation and the Air Force Office of Scientific Research, whose grants funded the majority of my graduate research.

I want to thank all my family members. I cannot imagine how I can go through this without their support. My special thanks go to my mother for encouraging me during difficult times, for taking care of my life, and for all she has done for me.

At last, but definitely not at least, I want to thank my husband for his care, his support, his encouragement, his tolerance, and his confidence on me. I am so lucky to have him to be with me for all the good and hard times.

List of Tables

| | | |
|-----|--|----|
| 3.1 | Descriptions of SPEC2000 benchmarks [66]. | 24 |
| 4.1 | Comparisons of sizes and misprediction rates of different access mode predictors. The separate 64 KB instruction cache (iL1) and the data cache (dL1) are 8-way with block size of 64 B. The unified L2 cache (uL2) is 8-way with block size of 128 B. The misprediction rates are the average over all the SPEC2000 programs. | 37 |
| 4.2 | The overall hit rates and first hit rates of multi-column and MRU structures for L1 instruction cache. The cache is 64 KB with block size of 64B. | 52 |
| 4.3 | The overall hit rates and first hit rates of multi-column and MRU structures for L1 data cache. The cache is 64 KB with block size of 64B. | 53 |
| 4.4 | The overall hit rates and first hit rates of multi-column and MRU structures for L2 cache. The cache is 4 MB with block size of 128B. | 54 |
| 4.5 | Access latencies for the way-prediction hit/miss and phased access on 64 KB 4-way L1 and 4 MB 8-way L2 caches. | 60 |

| | | |
|-----|---|-----|
| 5.1 | The distribution of high demanding, low demanding, and idle windows for program <i>swim</i> | 74 |
| 5.2 | Experimental parameters. | 79 |
| 5.3 | Percentage of execution time in low power mode, and percentage of power reduction on the issue logic and execution units by the load indicator scheme under the system with 1 GHz processor. | 82 |
| 5.4 | Parameters for instruction indicator. <i>EC</i> and <i>DC</i> are the conditions for enabling and disabling the low power mode. <i>IIPC</i> and <i>FPIPC</i> are the number of integer instructions issued per cycle and the number of floating point instructions issued per cycle, respectively. | 85 |
| 5.5 | The conditions for enabling and disabling the low power mode under different schemes. <i>LD</i> is the number of outstanding load misses. <i>I_{reg}</i> and <i>F_{reg}</i> are the numbers of free integer and floating-point registers. <i>TE_{XX}</i> and <i>TD_{XX}</i> are thresholds for enabling and disabling the low power mode, respectively. | 99 |
| 5.6 | The conditions for enabling and disabling the low power mode by the load-register indicator scheme. <i>LD</i> is the number of outstanding load misses. <i>I_{reg}</i> and <i>F_{reg}</i> are the numbers of free integer and floating-point registers, respectively. | 103 |

List of Figures

| | | |
|-----|--|----|
| 4.1 | The access patterns for (a) a conventional n-way set-associative cache; (b) a phased n-way set-associative cache; and (c) a way-prediction n-way set-associative cache. | 26 |
| 4.2 | Cache access latencies of the tag path, data path, sequential access, and parallel access on a 16-KB, 4-way set associative cache with block size of 32-byte. The values are estimated using the CACTI model [58]. | 29 |
| 4.3 | The hit rates for SPEC2000 benchmarks on an eight-way 64 KBytes data cache with block size of 64 Bytes and an eight-way 4 MBytes unified L2 cache with block size of 128 Bytes. | 30 |
| 4.4 | The access pattern of n-way AMP cache. | 32 |
| 4.5 | Way-prediction strategy for MRU cache. | 41 |
| 4.6 | Overall cache hit rate and first hit rate of a 64 KB MRU data cache for program 172.mgrid. | 42 |
| 4.7 | Way-prediction strategies for multicolumn cache. | 44 |
| 4.8 | Way-prediction strategy for multicolumn cache without swapping. | 45 |
| 4.9 | Comparisons of the first hit rates among the multicolumn and MRU caches. | 51 |

| | | |
|------|---|----|
| 4.10 | Energy reduction of multicolumn caches compared with MRU caches. . . . | 55 |
| 4.11 | Energy consumption of multicolumn cache, phased cache, and AMP cache. The system has four-way 64 KB instruction and data caches and eight-way 4 MB L2 cache. | 56 |
| 4.12 | Decomposition of energy consumed by instruction, data, and L2 caches. . . | 57 |
| 4.13 | Energy consumed by data and L2 caches. | 58 |
| 4.14 | CPI and E-D product reductions of the AMP cache compared with the mul- ticolumn cache. | 61 |
| 4.15 | CPI and E-D product reductions of the AMP cache compared with the phased cache. | 62 |
| 5.1 | Comparisons of the optimization based on current system status and the look-ahead optimization. | 66 |
| 5.2 | Sampled IPC values and number of outstanding loads during an arbitrarily selected 1024-cycle interval for program <i>swim</i> . “W64” and “W32” correspond to the sample window sizes of 64 cycles and 32 cycles, respectively. | 72 |
| 5.3 | CDF of arrival interval of miss streams. | 76 |
| 5.4 | Distribution of the number of concurrent accesses. | 77 |
| 5.5 | IPC values under the system with 1 GHz processors. | 83 |
| 5.6 | Percentage of total execution time in low power mode under our scheme (load) and instruction indicator (instruction). | 86 |
| 5.7 | Percentage of power reduction. | 91 |
| 5.8 | Normalized IPC values. | 92 |

| | | |
|------|--|-----|
| 5.9 | Percentage of energy reduction. | 93 |
| 5.10 | Power reduction under systems with processor speed scaling from 466 MHz to 1 GHz and 2 GHz. | 95 |
| 5.11 | Average intervals in the low power mode and the normal execution mode under different schemes. | 96 |
| 5.12 | Percentage of power reduction under the load indicator, load-instruction indicator, and load-register indicator schemes on systems with 1 GHz processors. | 101 |
| 5.13 | Percentage of total execution time in the low power mode under the load-register indicator scheme. The load and register portions correspond to the low power execution periods triggered only by the load indicator and the register indicator, respectively. The overlap portion corresponds to those caused by both triggers. | 102 |
| 5.14 | Percentage of energy reduction under the load indicator, load-instruction indicator, and load-register indicator schemes under systems with 1 GHz processors. | 104 |

ABSTRACT

The fast performance improvement of computer systems in the last decade comes with the consistent increase on power consumption. In recent years, power dissipation is becoming a design constraint even for high-performance systems. Higher power dissipation means higher packaging and cooling cost, and lower reliability. This Ph.D. dissertation will investigate several memory-related design and optimization issues of general-purpose computer microarchitectures, aiming at reducing the power consumption without sacrificing the performance. The memory system consumes a large percentage of the system's power. In addition, its behavior affects the processor power consumption significantly. In this dissertation, we propose two schemes to address the power-aware architecture issues related to memory:

- We develop and evaluate low-power techniques for high-associativity caches. By dynamically applying different access modes for cache hits and misses, our proposed cache structure can achieve nearly lowest power consumption with minimal performance penalty.
- We propose and evaluate look-ahead architectural adaptation techniques to reduce power consumption in processor pipelines based on the memory access information. The scheme can significantly reduce the power consumption of memory-intensive applications. Combined with other adaptation techniques, our schemes can effectively reduce the power consumption for both compute- and memory-intensive applications.

The significance, potential impacts, and contributions of this dissertation are:

1. Academia and industry R&D has solely targeted the objective of high performance in both hardware and software designs since the beginning stage of building computer systems. However, the pursuit of high performance without considering energy consumption will inevitably lead to increased power dissipation and thus will eventually limit the development and progress of increasingly demanded mobile, portable, and high-performance computing systems.
2. Since our proposed method adaptively combines the merits of existing low-power cache designs, it approaches the optimum in terms of both retaining performance and saving energy. This low power solution for highly associative caches can be easily deployed with a low cost.
3. Using “a cache miss”, a common program execution event, as a triggering signal to slow down the processor issue rate, our scheme can effectively reduce processor power consumption. This design can be easily and practically deployed in many processor architectures with a low cost.

Power Considerations for Memory-related Microarchitecture Designs

Chapter 1

Introduction

Computer system designs have been totally driven by the dedicated pursuit of high performance for several decades. As a consequence, we have seen a technique trend that well matches the Moore's law: both the clock rate of a processor and the number of transistors inside a chip double for every eighteen months over the last decades. However, this fast improvement on system performance does not come with no cost. As the performance improves exponentially, the chip power consumption also increases significantly. For instance, the power dissipation of the Alpha processors increases from 30 Watts on an early product 21064, to 50 Watts on 21164, then to 90 Watts on 21264. It is estimated that this value will reach 125–150 Watts on the future product 21464 [78]. In recent years, the high power consumption has put a lot of pressure on computer system designs. For example, in order to keep the processor running correctly under this high power consumption (which leads to high temperature), some special cooling and package techniques must be applied.

This dissertation will investigate several memory-related design and optimization issues of general-purpose computer architectures. For general-purpose systems, high performance is still the major concern. Thus, low-power designs for those systems should reduce power consumption without (or only with a slight) performance loss.

1.1 Importance of Low-power Architecture Designs

Traditionally, power consumption is a major concern for portable system designers. On one hand, users' demand on computation power is ever increasing. Today, many portable systems have processing power comparable to desktops or even servers produced just a few years ago. On the other hand, the operation of those systems is limited by the battery lifetime. Users' requirement on light weight and long operation time never ends. Although the advancement on battery technology does increase the battery lifetime, the improvement is only linear, far behind the exponential power increase. Thus, reducing power consumption can directly improve the usability of those systems.

In recent years, power consumption is becoming a critical design constraint even for high-end computer systems. Higher power dissipation means higher packaging and cooling cost, and also means lower reliability. For instance, it is estimated that once the processor dissipates more than 40 Watts, every Watt increase on power consumption costs about \$1 [64] in cooling and package. With the fast increase on power consumption, the air-cooled techniques will reach their limits soon. The urge for some other fancy cooling techniques such as water-cooled techniques seems not feasible due to their cost, noise, and weight. The increasing power consumption of computer systems may also cause environment problems. US EPA estimates that 10% of current electricity usage in US is directly due to desktop computers. In addition, this percentage may increase in the future as ubiquitous computing becomes more and more popular.

Power issues have been attacked at almost all design levels ranging from physical to application. Traditionally, low-power circuit and logic techniques, such as reducing the

supply voltage value and logic optimization, are the major research focuses in this field. These techniques have been effectively reducing the power dissipation of a single transistor or a single device. However, as the number of transistors inside a chip and the clock rate increase exponentially, low-power CMOS and logic designs alone can no longer solve all the power problems. Computer system designs must be **power-aware**. This requires work to be done in architecture, operating systems, and compiler [41, 52]. Architectural approaches has been and will be playing an important role here. For example, the projected power consumption of Pentium III 800 MHz processor scaled from Pentium Pro technique would reach 90 Watts. After careful designs and optimizations at architectural level, the product's power consumption is only 22 Watts [21].

1.2 Our Work

As the speed gap between processor and memory continues to widen, the design of memory system becomes increasingly important to the performance of many applications. To achieve high performance, large, highly associative, and multi-level caches are commonly deployed in today's computer systems. To tolerate long memory access latency, modern processors widely exploit complicated issue logic. As a consequence, the memory system contributes a large amount of the system's power consumption. In addition, the behavior of memory system also affects the power consumption of other components significantly. This Ph.D. dissertation will investigate several memory-related design and optimization issues of general-purpose computer architectures, aiming at reducing the power-consumption without sacrificing the execution performance.

We focus on designs and optimizations of processor pipeline and caches, which are not only the critical paths to host program executions, but are also major power consumers in a computer system. For example, it is estimated that the motherboard consumes more than 50% of the notebook power [41, 52]. Even for high-end computer systems, the processor is the single most power-hungry component and contributes about 25% of total system power consumption [70].

As stated above, processor is a particularly important target for power-aware designs due to its high power dissipation within a small die size (e.g. Alpha 21264 consumes 90 Watts within 314 mm^2 area). High-performance processors normally consist on-chip caches to reduce the memory stall time. On-chip caches occupy about 40% of the chip area and belong to the major power consumers in microprocessors. For instance, on-chip caches in Alpha 21264 processors consume 15% of the total power [32]. It is estimated that this portion will further increase to 26% in a future product, the Alpha 21464 processors [78]. For some low-power processors with simple execution logics, this percentage may be even larger. As an example, in a low power embedded processor SA-110, caches consume 43% of its total power [54]. Thus, it is very important to reduce the on-chip cache power consumption.

Our first work targets reducing power consumption of set-associative caches. Set-associative caches are widely used for their ability to reduce cache conflict misses. However, the conventional implementation of set-associative caches is not energy-efficient because multiple blocks in a set are accessed for each cache reference but at most one block will contain the required data. We notice that none of existing techniques are optimized for both cache hits and misses in terms of performance and power consumption. Since cache hit rates are highly application-dependent, those techniques only perform well for a certain

type of applications. To address this limit, we propose an access mode prediction (AMP) cache that dynamically applies different access modes based on the cache hit/miss prediction. Our experimental results indicate that the AMP cache can achieve the lowest energy consumption across a wide range of applications with optimal performance.

To boost performance, modern processors can issue multiple instructions in each cycle and execute instructions in the order different from that in the program. To support those features, the issue logic and execution units are very complicated and contribute a large percentage of processor power consumption. For example, the issue logic and execution units consume about 40% of the total processor power in Alpha 21264 [78]. Thus, dynamic adjusting the processor issue rate can significantly reduce the processor power consumption [6].

Our second work focuses on reducing the processor power consumption based on the memory access information. When a cache load miss falls to the main memory, it is almost certain that the processor will stall for this memory access thus the full processor issue rate is not necessary to maintain the performance. We propose a load indicator scheme that adjusts the processor issue rate according to the existence of main memory accesses. Our experiments show that this simple scheme is very effective in reducing the power consumption for memory-intensive applications. This scheme can be further combined with other indicators based on the average number of instructions issued in each cycle or the number of unoccupied registers to reduce power consumption for both compute- and memory-intensive applications.

The significance, potential impacts, and contributions of this dissertation are:

1. Academia and industry R&D has solely targeted the objective of high performance in both hardware and software designs since the beginning stage of building computer systems. However, the pursuit of high performance without considering energy consumption will inevitably lead to increased power dissipation and thus will eventually limit the development and progress of increasingly demanded mobile, portable, and high-performance systems. This dissertation will address this important issue by focusing on power considerations for memory-related microarchitecture designs.
2. Since our proposed method adaptively combines the merits of existing low power caches, it approaches the optimum in terms of both retaining performance and saving energy. This low power solution for highly associative caches is easy to be deployed with a low cost.
3. Using “a cache miss”, a common program execution event, as a triggering signal to slow down the processor issue rate, our schemes are able to accurately and effectively save unnecessary processor power consumption. This design can be easily and practically deployed in many standard and special processors with a low cost.

The dissertation is organized as follows. In the next chapter, we will present some technical background about low power designs. Then, we will introduce the evaluation environment in Chapter 3. In Chapter 4, we will discuss the low power cache designs based on dynamic power mode switching. In Chapter 5, we will present our work in reducing pipeline power consumption using memory access information. Finally, we will summarize our work and discuss future work in Chapter 6.

Chapter 2

Background

In this chapter, we will briefly discuss the sources of power consumption and introduce some existing work in reducing the power consumption.

2.1 Sources of Power Consumption

In CMOS circuitry, power are consumed by two different types of activities. Dynamic power dissipation is caused by the capacitance charging and discharging as signals transit from 0 to 1 and from 1 to 0. The dynamic power dissipation can be expressed as in [52]:

$$Power_{dynamic} = \frac{1}{2} \cdot C \cdot V^2 \cdot A \cdot f, \quad (2.1)$$

where C is the capacitance of switching nodes, V is the supply voltage, A is the activity factor, and f is the clock frequency.

The capacitance C is a function of wire length and transistor size, which are determined by the underlying circuit technology. The value of C is also roughly proportional to the number of transistors in the circuit. The activity factor A is the average transition probability, which is determined by the circuit implementation and the program behavior. The value of A is between 0 and 1.

Another type of power dissipation is static power consumption. Different from the dynamic power dissipation that only arises due to signal transitions, the static power is consumed in absence of any switching activity. It is caused by leakage current flowing through every transistor that is powered on. The static power consumption can be modeled at the architectural level as in [15]:

$$Power_{static} = N \cdot V \cdot k_{design} \cdot \hat{I}_{leak}, \quad (2.2)$$

where N is the number of transistors, V is the supply voltage, k_{design} is an empirically determined parameter representing the characteristics of a transistor, and \hat{I}_{leak} is a technology-related parameter describing the leakage current per transistor.

Currently, the dynamic power dissipation dominates the total chip power consumption. It accounts for more than 95% of the total power dissipation. The static dissipation only accounts for 2–5% of the total chip power [43]. Thus, like most existing low-power techniques, our work focuses on reducing the dynamic power consumed by the computer systems. As transistors become smaller and faster, the percentage of static power dissipation will increase in the future [15]. There is an increasing number of research projects targeting reducing the static power consumption. Other sources of power consumption also only account for a very small percentage of the total chip power, such as the short-circuit power dissipation due to the finite-slope input signals.

2.2 Low-power Techniques

As stated in Chapter 1, the increasing power consumption has become a severe design constraint as the computer systems become faster and more complicated. Researchers have

proposed low-power techniques at almost all design levels, ranging from physical level to software level, to address this issue [41, 92].

2.2.1 Power-saving Techniques at Physical, Circuit and Logic Levels

Equations (2.1) and (2.2) show that the basic principle of low-power design is to reduce the supply voltage, reduce the capacitance, reduce the switching frequency, reduce the short circuit currents, and reduce the leakage currents. These goals can only be achieved by the technology advancements at physical and circuit levels. Other techniques, such as transistor pin ordering, delay paths balancing, and network restructuring, can produce low-energy gate and functional unit designs. Those logic optimizations can also effectively reduce the circuit power consumption.

Traditionally, the major research focus in low-power design area is on techniques at these design levels. Researchers have proposed many techniques to effectively keep the chip power consumption at a reasonably low level. However, the fast increase on both speed and complexity of computer systems has pushed the power dissipation to a point that the low-power techniques at physical, circuit, and logic levels alone can no longer solve all the problems.

In general, the supply voltage V decreases by 30% for each process generation, and the wire width also reduces by about 30% (which decreasing the capacitance of each transistor) [15]. Both reductions can help keeping the power consumption down. On the other hand, the number of transistors inside the chip doubles for each process generation. In the meantime, the clock frequency of processor also doubles for every 18 months. These two factors will push the chip power consumption up. As a simple estimation, we can see

that the chip capacitance C will increase to $0.7 \times 2 \times C$ in the next process generation, the supply voltage V will decrease to $0.7V$, and the clock rate f will double to $2f$. According to Equation (2.1), the chip dynamic power dissipation will increase by about 40% for each process generation, even after the low-power circuit level techniques have been applied. Thus, the dramatic increases on both complexity and clock frequency of computer systems have already made it necessary to consider power issues at high design levels.

2.2.2 Power-saving Techniques at Architectural Level

In most cases, reducing power consumption and improving performance have conflicting interests. For instance, from Equation (2.1), we can see that slowing down the clock is a straightforward solution for reducing power dissipation. However, the execution time of a program is inversely proportional to the clock rate f . Thus, it is crucial to make trade-offs between performance and power consumption for power-aware designs.

As a natural bridge between software and circuits, the architectural design can effectively utilize the low-level power-saving techniques according to the program's execution behavior and achieve a good balance between performance and power consumption. Since the major target for high-end computer systems is still high performance, the power-aware designs for those systems should *reduce the power consumption without (or only with a slight) performance loss*.

Normally, the design of a general-purpose computer system is optimized to achieve the best average performance for a wide range of representative applications. As a consequence, it is common that part of hardware resources are underutilized for a specific application. Disabling those underutilized resources will not affect the system performance while the

power consumed by those resources can be saved.

A lot of work has been done at the architectural level to reduce the power consumption of the whole computer system, including the processor, cache, memory, and bus. For example, pipeline gating [51] reduces the energy consumption of processors with branch prediction mechanisms by determining when the prediction is likely to be incorrect and preventing wrong-path instructions from entering the pipelines. The filter cache [47] adds a very small buffer between the CPU and L1 cache to filter references to the L1 cache. A hit on the buffer will consume less energy than an access to the L1 cache due to the smaller size of the buffer. Signal encoding with Gray code [67] or invert coding [42] can reduce the energy consumption on the bus by reducing the bit switching frequency. Compression is another type of effective approaches to reducing the energy consumption of processor core, cache, and bus. The compression can be based on significant bytes, zero, or other frequently used values [20, 75, 80].

Researchers have also proposed many architectural schemes to reduce the static power consumption. For instance, the cache decay technique can reduce the cache leakage power by predicting those blocks which will not be used anymore and placing those blocks in a low leakage standby mode [43].

We will discuss architectural low-power techniques related to our work in more details in subsequent chapters.

2.2.3 Power-saving Techniques at Software Level

The techniques for energy reduction at software level can be applied at three different layers: algorithm/application, compiler, and operating systems.

The most effective approach of energy reduction is algorithm optimization. In this aspect, reducing energy consumption is identical to reducing computation complexity [92]. As a simple example, if a 30% reduction of the number of instructions to be executed results in reducing the program's execution time by 30%, this can be directly converted to 30% of energy saving if the power consumption of each instruction is a constant.

Some performance-oriented compilation techniques are also beneficial to energy consumption. For example, loop transformations and data transformations can improve the program locality thus reduce the memory stall time. These techniques can also reduce the energy consumption because they can reduce the number of memory references which are more energy hungry than cache accesses. However, some performance-oriented optimization is not the best from energy consumption point of view. For example, compared with the performance-oriented instruction re-ordering, the energy-oriented one can reduce the energy consumption by 10–15% [41]. So it is necessary to perform optimizations based on different targets.

As the traditional resource manager, operating systems also play an important role in energy saving. Currently, many hardware devices provide multiple power modes where those modes with lower power dissipation have longer response time. Operating systems can take advantage of this feature for power management. Dynamic power management achieves energy-efficient computation by selectively turning off system components when they are idle or partially unexploited [9]. For example, spinning down the disk when its inactive time passes a threshold is a commonly used policy for energy reduction. ACPI (Advanced Configuration and Power Interface) is an open industry specification that defines the manner in which the OS, motherboard hardware, and peripheral devices talk to each other regarding

power usage [40].

Although software level techniques are effective in reducing power dissipation, their effects only restrict to a specific application (algorithm or compiler techniques) or a specific execution environment (OS power management). To deliver a product (processor or computer system) with targeted power consumption, it must still rely on power-aware designs and optimizations at architectural level.

In this dissertation, we will present two memory-related architectural designs to reduce the system power consumption.

Chapter 3

Evaluation Environment

3.1 Evaluation Metrics

3.1.1 Performance Metrics

A direct indicator of system performance is the execution time or delay of a group of representative programs on the target computer system. For the purpose of computer architecture research, two related metrics, *CPI* (cycles per instruction) and its reciprocal *IPC* (instructions per cycle), provide more insightful informations about the system performance. The execution time or delay (D) of a program can be expressed as:

$$D = I \cdot CPI \cdot (1/f), \quad (3.1)$$

$$\text{or } D = I \cdot \frac{1}{IPC \cdot f}, \quad (3.2)$$

where I is the number of instructions executed, and f is the clock rate. D is measured in unit of *Second*.

The value of *CPI* is determined by two factors. The first one is the program's behavior, such as the inherent instruction-level parallelism (ILP) and the program locality. The *CPI* also depends on the underlying architecture, for example, the instruction set, the

processor issue rate, and the cache size. Analyzing the change and breakdown of the *CPI* or *IPC* values can provide many insightful informations on architectural designs, such as the performance bottleneck and the effectiveness of a new scheme.

3.1.2 Power and Energy Metrics

Power consumption of a system is the rate at which energy is drawn over time. It is measured in unit of *Watt*. The power consumption directly determines the system operation temperature. Thus, it is essential to guarantee that the power consumption will never be so high to drive the system reaching a dangerous temperature level [36]. Lower power dissipation will alleviate some burdens on packaging and cooling systems, and make the system more reliable.

Energy consumption is measured in unit of *Joule*. Its relationship with power dissipation is expressed as:

$$E = P \cdot D, \quad (3.3)$$

where E and P are energy and power consumptions, respectively. Lower energy consumption means more useful work can be done for a given energy budget. For portable systems, reducing energy consumption means increasing the battery lifetime.

According to Equation (3.3), it seems to be straightforward that reducing either power consumption P or execution delay D can effectively reduce the energy consumption. However, in most cases, reducing power consumption comes at the cost of degrading performance (i.e. enlarging the delay). For example, slowing down the clock rate can effectively reduce the power dissipation, since $P \propto f$ (Equation 2.1). On the other hand, slowing down the clock rate will also enlarge the execution time, since $D \propto (1/f)$ (Equation 3.2). Thus,

simply slowing down the clock rate cannot save energy. Low-power designs need to also consider their performance impact.

3.1.3 Energy-efficiency Metrics

In some cases, a low-power technique is also beneficial to performance or at least without negative effects on performance. For example, loop optimization not only reduces the program's execution time, but also reduces its energy consumption on memory systems. In these cases, using energy or power consumption metric is sufficient to evaluate the effectiveness of the technique.

However, as we discussed above, in most cases, a design optimized to performance causes unnecessary energy consumption. As an example, aggressively allowing multiple instructions executing on different branch paths can improve the performance of superscalar processors. However, the energy consumed by instructions executed on wrong-paths is wasted. Thus, designers must make trade-offs on performance and energy for their target systems. A new metric is needed to evaluate this trade-off.

The energy-delay product (*EDP*) is widely adopted by researchers to evaluate the energy-efficiency of their designs [12, 31, 39, 47, 48, 67, 92].

$$EDP = E \cdot D. \tag{3.4}$$

In general, lower energy consumption corresponds to longer execution time. The target of energy-efficient designs is to find the optimal design point where the energy-delay product is minimized.

Authors in [13, 52] argue that it is more appropriate to apply different metrics for different classes of systems. For low-power portable systems, the battery life limits the system operations. Users are more tolerant for slow response time on those systems. Thus, energy is the primary consideration there. This makes the power-delay product (PDP), which is identical to energy (E), be a natural metric for such kind of systems. On the other hand, for the highest-performance server-class systems, the program delay is more important than the energy consumption. As a reflection, it may be more suitable to overweight the delay part. ED^2P ($E \cdot D^2$) metrics are more appropriate for the highest-end systems. EDP metrics cannot be applied across diverse classes of systems for comparing system efficiency. However, EDP metrics are still useful for a given class of systems [52].

In our study, we will use the EDP metric to evaluate the power-performance trade-offs, since our analysis is within a single class of systems, the general-purpose systems.

3.2 Performance Evaluation Tools

The most straightforward and convincing way to evaluate the performance of a target system is direct measurement. For example, to promote their products, computer vendors normally report the SPEC results of the products by running the standard benchmarks provided by Standard Performance Evaluation Corporation [66]. In order to provide detailed information on system performance, today's high-performance processors provide hardware counters that can monitor the number of certain events happening during a program's execution. For instance, users can use hardware counters to check the cache miss rate of a given program and identify the performance bottleneck.

Although the direct measurement is accurate, several factors limit its usage in the architecture research. The hardware counters can only monitor a few events at a time. More importantly, the direct measurement can only be performed on existing systems. Thus, it is not suitable for evaluating new ideas with variants of architecture parameters. Simulation and modeling are more suitable on that situation.

Normally, modeling requires a relatively small amount of computation resources and takes a short time. However, modern computer systems are very complicated. For instance, the processor can issue multiple instructions in each cycle; instructions can be executed speculatively; and memory access latency is not a constant. All these factors make it very difficult if not impossible to develop an accurate and attractable model for real architectural study.

Simulation has been extensively used for the computer architecture research because of the high cost on building hardware prototypes and also because of its accuracy in estimating performance of complicated computer systems. SimpleScalar tool set [14] is the most commonly used simulator for evaluating new ideas for uniprocessor systems in recent years. It is an execution-driven simulator that generates simulation results on the fly and produces comprehensive statistics for the program execution. The most advanced simulator in the tool set, *sim-outorder*, emulates a superscalar processor with five-stage pipelines¹ and multi-level memory systems. It supports several advanced techniques such as multi-issue, out-of-order execution, speculative execution, and non-blocking load/store. Currently, SimpleScalar supports two instruction sets. The *PISA* version supports the SimpleScalar Instruction Set Architecture which is a close derivative of the MIPS instruction

¹The five stages of the pipeline are fetch, dispatch, execution, writeback, and commit.

set. The *Alpha* version supports Alpha ISA and can directly run binary code compiled on real Alpha systems.

We use the Alpha version of *sim-outorder* simulator in our experiments and insert some new functions to the simulator for our study. For example, we have implemented the multicolumn-based way-prediction on the cache access function for our low-power cache study.

SimpleScalar only emulates the behavior of the processor. Compared with those simulators (e.g. SimOS [60]) that emulate the behavior of the whole system, including the processor, memory, disks, and operating systems, SimpleScalar cannot capture the program's execution behavior due to context switches or system calls. However, compared with those whole-system simulators, it provides more detailed and flexible simulations on the processor, which is the target in our study. In addition, the benchmark we have used in our study, SPEC 2000 [66], mainly consists scientific applications, whose execution behavior is much less dependent on operating systems than commercial workloads. In order to characterize the execution behavior of commercial workloads, we need to use the whole-system simulators [84, 27]. However, since the major concern of this dissertation focuses on microarchitecture, using a detailed processor simulator is more suitable. In addition, our experimental results will not be favored due to those behaviors that cannot be captured by the simulator. For example, the variants of cache miss rate would be larger if context switches are performed. This will make our low-power cache designs save more energy than that under the single process environment.

3.3 Energy Consumption Evaluation Tools

Energy consumption can be analyzed by either simulation-based or probabilistic-based techniques [41]. Simulation-based techniques accurately monitor transitions at each cycle and use lower level analysis results to construct higher level models. In contrast, probabilistic-based techniques view signals as random transitions between 0 and 1 with certain statistical characteristics. Normally, simulation-based techniques are more accurate than probabilistic-based ones but take much more computation time.

Until recently, power estimation tools are only available at low design levels, such as Hspice [5] and PowerMill [68]. Although these circuit level simulations can accurately estimate dynamic and static power dissipation within a few percentage, they are not very useful for making architectural decisions [12]. Their simulation is too slow. Thus, they are only practical for circuits with relatively small number of transistors. More importantly, these tools need complete circuit designs as input which are not available at architectural level design stage.

Researchers have been making efforts to provide energy estimation tools at the architecture level. As a regular structure, cache has been a good candidate for energy analysis [67, 42, 46, 58]. For example, authors in [42] propose an analytical energy dissipation model for conventional set-associative caches and use it to evaluate several low power techniques. CACTI [79] is an analytical model for estimating the access and cycle time of direct-mapped and set-associative caches. It takes the cache size, block size, associativity, and process parameters as input. It has been extended to CACTI 2.0 [58] that includes both timing and power models for on-chip caches. We use CACTI 2.0 to estimate the power

consumption of different implementations of set-associative caches in our study.

Only until recently, energy consumption simulations for the integrated system are publicly available [12, 74]. Wattch [12] integrates parameterized power models of common structures in modern superscalar processors into the SimpleScalar architectural simulator [14]. The cycle-level performance simulator generates cycle-by-cycle hardware access counts for each basic module and inputs these counts to the parameterized power models to estimate the power dissipation of the whole processor and each component. SimplePower [74] also builds its infrastructure on the SimpleScalar simulator. However, it uses a combination of analytical and transition-sensitive models for energy estimation. For regular structures such as caches, the analytical models are applied. For other components such as functional units whose energy consumptions are dependent on the switching activity and capacitance, the transition-sensitive models are applied. The simulator contains switch capacitance tables for functional units for each input transition obtained from lower level simulations. Table lookups are done based on the input bits and output bits for the estimated components. Wattch provides fully parameterizable power model for the processor, while SimplePower only provides parameterized model for caches. The currently publicly available version of SimplePower only models energy consumption for an in-order 5-stage pipelined data path with perfect caches running only integer instructions.

3.4 Workloads

SPEC CPU2000 is an industry-standardized benchmark suite that contains compute intensive benchmarks exercising a wide range of hardware [66]. It is extensively used to evaluate

the performance of the processor, memory, and compiler.

The SPEC CPU2000 suite consists of two groups of benchmarks, the integer one (CINT2000) and the floating point one (CFP2000). Table 3.1 describes all the programs briefly [66].

We use SPEC CPU2000 as workload in our study. The reference input data files are used in the experiments. We use the precompiled Alpha version of SPEC2000 binaries [77] and run it under SimpleScalar simulator. To eliminate the start-up effects, we fast-forward the first four billion instructions, then collect detailed statistics on the next one billion instructions.

| Category | Program | Description | |
|--------------|-------------|---|----------------------------------|
| CINT2000 | 164.gzip | Compression | |
| | 175.vpr | FPGA Circuit Placement and Routing | |
| | 176.gcc | C Programming Language Compiler | |
| | 181.mcf | Combinatorial Optimization | |
| | 186.crafty | Game Playing: Chess | |
| | 197.parser | Word Processing | |
| | 252.eon | Computer Visualization | |
| | 253.perlbnk | PERL Programming Language | |
| | 254.gap | Group Theory, Interpreter | |
| | 255.vortex | Object-oriented Database | |
| | 256.bzip2 | Compression | |
| | 300.twolf | Place and Route Simulator | |
| | CFP2000 | 168.wupwise | Physics / Quantum Chromodynamics |
| | | 171.swim | Shallow Water Modeling |
| 172.mgrid | | Multi-grid Solver: 3D Potential Field | |
| 173.applu | | Parabolic / Elliptic Partial Differential Equations | |
| 177.mesa | | 3-D Graphics Library | |
| 178.galgel | | Computational Fluid Dynamics | |
| 179.art | | Image Recognition / Neural Networks | |
| 183.earth | | Seismic Wave Propagation Simulation | |
| 187.facerec | | Image Processing: Face Recognition | |
| 188.amp | | Computational Chemistry | |
| 189.lucas | | Number Theory / Primality Testing | |
| 191.fma3d | | Finite-element Crash Simulation | |
| 200.sixtrack | | High Energy Nuclear Physics Accelerator Design | |
| 301.apsi | | Meteorology: Pollutant Distribution | |

Table 3.1: Descriptions of SPEC2000 benchmarks [66].

Chapter 4

Access Mode Predictions for Low-power Cache Design

4.1 Significance of Low-power Cache Design

As we have stated in Chapter 1, the successful pursuit of high performance on computer systems has produced a negative byproduct of high power dissipation. Circuit-level techniques alone can no longer keep the power dissipation under the reasonable level. Increasing efforts have been made on reducing power dissipation via architectural approaches [13]. One research focus is on reducing the power consumed by on-chip caches, which are among the major power consumers in microprocessors.

Today's computer systems normally consist multi-level memory hierarchy. The performance of caches is critical to many applications' performance. To achieve high performance, the size of caches is getting larger, and the associativity of caches is also increasing. In addition, many processors embrace both level one and level two caches on chip. As a consequence, on-chip caches occupy about 40% of chip area and consume a large portion of chip power. For instance, on-chip caches in Alpha 21264 processors consume 15% of the total

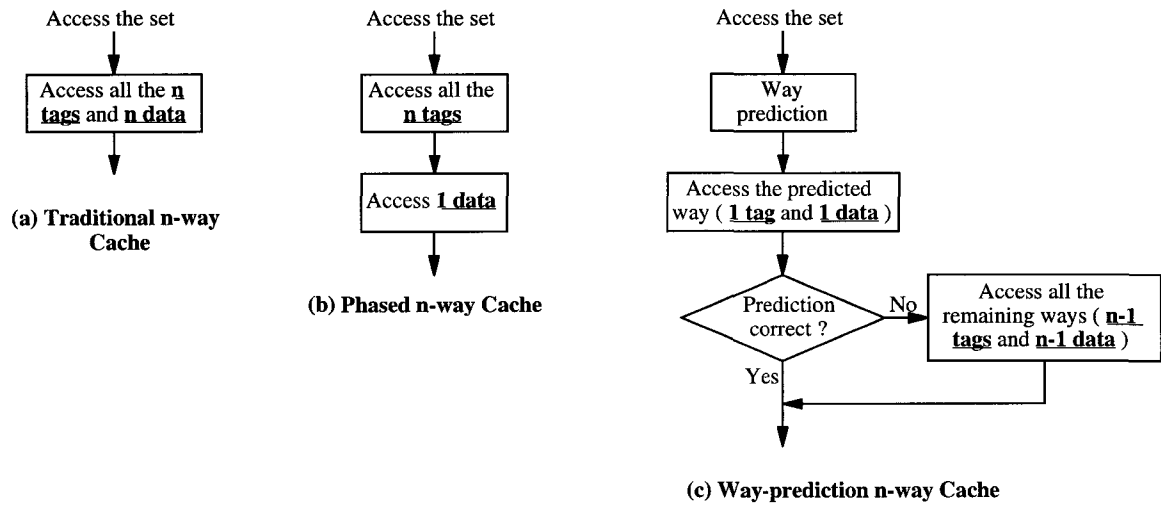


Figure 4.1: The access patterns for (a) a conventional n -way set-associative cache; (b) a phased n -way set-associative cache; and (c) a way-prediction n -way set-associative cache.

power [32]. It is estimated that this portion will further increase to 26% in a future product, the Alpha 21464 processors [78]. This percentage can be even larger in some low power processors. For example, caches consume 43% of the total power of embedded processor SA-110 [54].

Set-associative cache is commonly used in modern computer systems for its ability to reduce cache conflict misses. However, a conventional set-associative cache implementation is not power-efficient by its nature. As Figure 4.1-(a) shows, a conventional n -way set-associative cache probes all the n blocks (both tag and data portions) in a set for each cache reference. However, at most block will contain the required data. The percentage of wasted energy will increase as the cache associativity n increases. We have already seen high-associativity caches in some commercial processors. For example, Intel Pentium 4 processor exploits four-way L1 caches and an eight-way L2 cache. Thus, it is very important

to reduce the power consumption of set-associative caches.

4.2 Related Work

An effective approach to reducing the power consumption of a set-associative cache is to reduce the number of memory cells involved for an access. One scheme is to divide each data RAM into several sub-banks, and to only activate the words at the required offset from all cache ways [67]. Another alternative is to selectively disable a subset of cache ways during execution periods with modest cache activity [3]. Other examples of techniques to reduce cache energy consumption include the one used in SA-100 processor which divides each of the 32-way instruction and data caches into 16 fully-associative caches and enables only one-eighth of the cache for each access [54]. The filter cache [47] adds a very small buffer between the CPU and L1 cache to filter references to the L1 cache. A hit on the buffer will consume less energy than an access to the cache due to the much smaller size of the buffer. The tag comparison elimination technique keeps links within the instruction cache and allows instruction fetch to bypass the tag array [50, 85]. This only works for instruction caches whose access patterns are regular and predictable.

Two representative techniques in reducing the power consumption of set-associative caches are the phased cache and way-prediction cache [39]. The phased cache [11, 34] first compares all the tags with the accessing address, then probes the desired data way only. The way-prediction technique first speculatively selects a way to access before making a normal cache access. Combining these two techniques, the predictive phased cache [37] first probes all the tag sub-arrays and the predicted data sub-array. This approach reduces the

energy consumption of non-first hits (cache hits whose locations are mispredicted) in the way-prediction cache at the price of increasing energy consumption for both first hits (cache hits whose locations are correctly predicted) and cache misses.

4.3 Comparisons between Phased and Way-prediction Caches

Figure 4.1-(b) and (c) illustrate the access patterns for phased and way-prediction n -way set-associative caches, respectively. Compared with the conventional implementation, the phased cache only probes *one* data sub-array instead of n data sub-arrays¹. However, the sequential accesses of tag and data will increase the cache access latency. As shown in Figure 4.2, the tag path without output takes about 1.15 ns, and the data path alone without output takes about 0.90 ns on a 16-KB, 4-way set associative cache with block size of 32-byte². Under the conventional implementation, the tag and the data are accessed in parallel. Each cache access takes about 1.32 ns. In contrast, if the tag and the data are accessed one by one, the access latency will be almost doubled to 2.22 ns. Thus, although the phased implementation can reduce the power consumption of set-associative caches, it will increase the cache access latency and harm the overall system performance.

The way-prediction cache first accesses the tag and data sub-arrays of the predicted way. For instance, the MRU-based way-prediction cache always accesses the most recently accessed block in a set first. If the prediction is not correct, it then probes the rest of tag and data sub-arrays simultaneously. An access in phased cache consumes more energy and

¹Each way comprises a tag sub-array and a data sub-array.

²The values are estimated using the CACTI model [58].

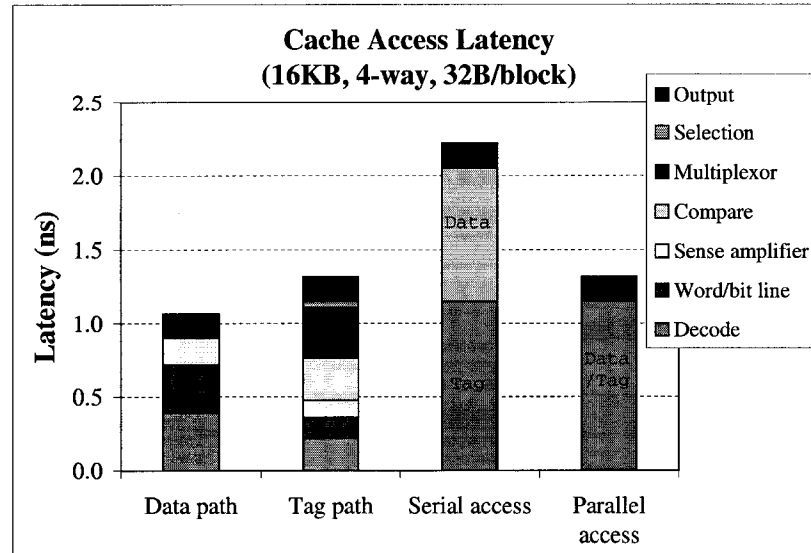


Figure 4.2: Cache access latencies of the tag path, data path, sequential access, and parallel access on a 16-KB, 4-way set associative cache with block size of 32-byte. The values are estimated using the CACTI model [58].

has longer latency than a correctly predicted access in way-prediction cache, but consumes less energy than a mispredicted access. When the prediction accuracy is high, the way-prediction cache is more energy-efficient than the phased cache [39].

The way-prediction hit rate is bounded by the cache hit rate which is highly application-dependent. Figure 4.3 shows the hit rates of all the SPEC2000 benchmark programs on a 64 KBytes data cache and a 4 MBytes L2 cache³. Most applications have L1 data cache hit rates as high as 95% with a few exceptions, such as *art* whose hit rate is only 65%. The distribution of L2 cache hit rates is even more diverse than that of L1 cache hit rates. Fourteen of the twenty-six programs have L2 cache hit rates higher than 95%, while ten

³We will discuss the experimental environment in details at Section 4.7.

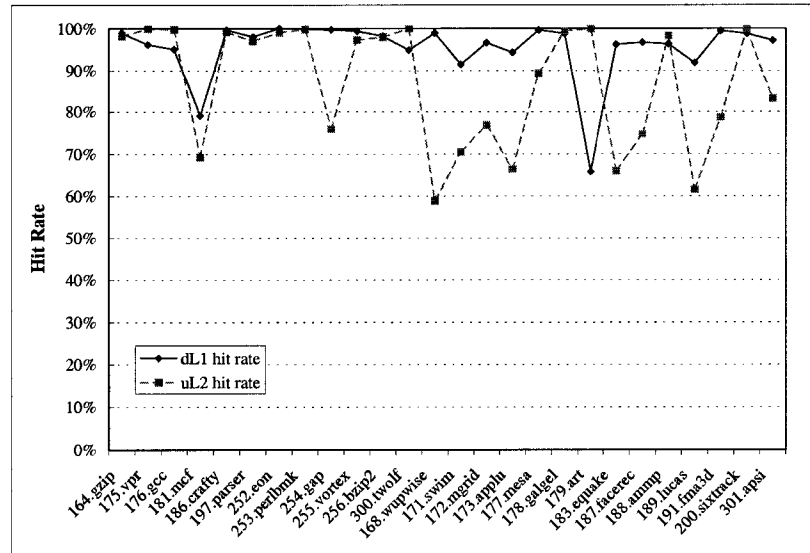


Figure 4.3: The hit rates for SPEC2000 benchmarks on an eight-way 64 KBytes data cache with block size of 64 Bytes and an eight-way 4 MBytes unified L2 cache with block size of 128 Bytes.

programs have L2 cache hit rates below 80%.

The way-prediction cache will consume less energy on applications with good locality, while the phased cache will consume less energy on applications with poor locality. Thus, neither way-prediction cache nor phased cache can perform consistently well in power saving across all these applications.

4.4 AMP Caches

4.4.1 Strategy

The energy consumption of a cache access can be minimized if a cache hit is handled by the way-prediction mode, while a cache miss is handled by the phased mode. Motivated by this relationship between a cache access mode and its energy consumption, and by the application-dependent cache hit patterns, we propose to use an *access mode prediction* technique based on cache hit/miss prediction for low power cache design [89]. It combines the energy-efficient merits of both phased and way-prediction cache structures. In the case of predicted cache hits, the way-prediction scheme is applied to determine the desired way and to probe that way only. In the case of predicted misses, the phased scheme is applied to access all the tags first, then probe the required way only. We call this *AMP cache*.

Figure 4.4 presents the access sequence controlled by access mode predictions. Upon a reference, the access mode predictor makes a decision based on the cache access history. If the predictor indicates that the way-prediction mode should be applied, the following access sequence will be handled by the way-prediction scheme. Otherwise, it will be handled by the phased access scheme.

The two most important components in the AMP cache design are the access mode predictor and the way-predictor.

- *Access Mode Predictor*

A good access mode predictor is essential to select a suitable access technique for each cache reference. Since we target at low power cache design, the access mode predictor should only add little overhead on both latency and energy consumption.

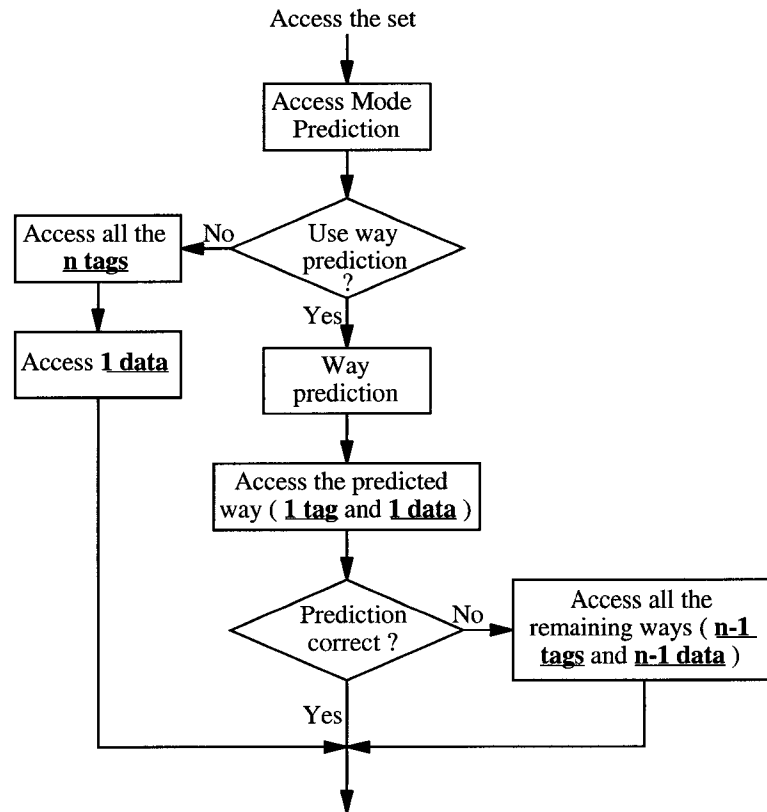


Figure 4.4: The access pattern of n-way AMP cache.

Motivated by existing branch prediction techniques, we derive a simple predictor that uses a global access history register and a global pattern history table obtaining high prediction accuracy.

- *Way-predictor*

A good way-predictor can minimize the energy consumption of cache hits. Previous studies have shown that MRU-based way-prediction can effectively reduce the energy consumption of set-associative caches. However, we find that as the cache associativity increases, the effectiveness of MRU-based way-prediction in energy saving continues

to decrease. To address this problem, we study several way-prediction policies and find that the multicolumn-based way-prediction is a nearly optimal technique. It is highly effective for energy reduction of high associativity caches. We also propose a power-efficient variant of multicolumn cache to eliminate the swapping operation in the original design.

4.4.2 Power Consumption

In this section, we will use a simplified timing and energy model to quantify the observation that we have discussed above. The latency and energy consumption of different types of caches used in our experiments are estimated based on the CACTI timing and power consumption model [58].

Let E_{tag} and E_{data} be the energy consumed by a tag sub-array and a data sub-array upon a reference, respectively. For a correctly predicted hit in the way-prediction cache, the energy consumed is $E_{tag} + E_{data}$, compared with $n \cdot E_{tag} + E_{data}$ in the phased cache, where n is the associativity of the cache. On the other hand, a miss in the way-prediction cache will consume $(n + 1) \cdot E_{tag} + (n + 1) \cdot E_{data}$, in comparison with $(n + 1) \cdot E_{tag} + E_{data}$ in the phased cache. Regarding the access latency, a correctly predicted hit in the way-prediction cache takes one time unit, compared with two time units in the phased cache. Our objective is to pursue the lowest possible energy consumption and latency for both cache hits and misses.

Let H , M , and W_b be the numbers of cache hits, misses, and write-backs, respectively.

A phased cache, a way-prediction cache, and an AMP cache consume energy as follows:

$$E_{phased} = H \cdot (n \cdot E_{tag} + E_{data}) + M \cdot ((n + 1) \cdot E_{tag} + E_{data}) + W_b \cdot E_{data}, \quad (4.1)$$

$$E_{way_prediction} = H \cdot (E_{tag} + E_{data}) + M \cdot ((n + 1) \cdot E_{tag} + E_{data}) + M_R \cdot n \cdot E_{data} + W_b \cdot E_{data} + \Delta_{w_mispred}, \quad (4.2)$$

$$E_{AMP} = H(E_{tag} + E_{data}) + M((n + 1)E_{tag} + E_{data}) + W_b \cdot E_{data} + \Delta_{A_mispred}, \quad (4.3)$$

where $\Delta_{w_mispred}$ and $\Delta_{A_mispred}$ are the energy overheads of mispredictions in the way-prediction cache and the AMP cache, respectively. M_R is the number of read misses.

For an AMP cache with a perfect access mode prediction and a perfect way-prediction, its energy consumption is the lower bound of energy consumption for the set-associative cache. The misprediction overhead is determined by both the accuracy of access mode predictor and that of way-predictor. We will discuss these two predictors in following sections.

4.5 Access Mode Predictors

4.5.1 Designs

Authors in [82] propose to use cache hit/miss prediction for improving load instruction scheduling. In order to reduce the memory bandwidth requirement, authors in [72] use miss prediction to dynamically mark which load instruction is cacheable/non-allocatable. We want to apply cache hit/miss prediction to choose way-prediction or phased access for energy savings.

This is different from most work in predicting cache behavior which targets prefetching or reducing cache miss rate. For example, the next cache line and set predictor is used for instruction fetch prediction and branch prediction [19]. Prediction caches use a history of recent cache misses to predict whether a replaced cache line will be accessed in the near future thus should be put into the victim cache to reduce the overall cache miss rate [10].

The access mode predictor shares a similar objective with the branch predictor. Thus, we derive the access mode predictor from existing branch predictors [65, 81, 55, 53]. The intuition behind the access mode predictor is that cache misses are clustered and program behavior is repetitive. Technically, nearly all branch prediction techniques may be adapted. However, since we target at reducing cache power consumption, the access mode predictor must be simple. Thus we only present variants with low resource requirements here.

Each cache (instruction, data, or L2) has its own access mode predictor, which uses its reference address or hit/miss history as the index to the prediction table. We have studied following prediction variants.

- *Saturating counter.*

This prediction has the simplest implementation. It is based on the two-bit saturating up/down counter [65]. Each cache has its own prediction table. The number of entries in the table equals the number of cache sets. Each entry is a two-bit saturating counter. When a cache reference arrives, its set index determines which counter should be accessed. If the most significant bit of the counter is “1”, the way-prediction scheme will be applied. Otherwise, the phased scheme will be applied. The counter is incremented for each way-prediction hit and is decremented for each way-prediction

miss or cache miss.

- *Two-level adaptive predictor.*

Another alternative is derived from the two-level adaptive branch predictor [81]. We implement both *GAg* and *PAg* versions. In the implementation derived from *GAg* version, a global k -bit access history register records the results of the most recent k accesses. If the access is a way-prediction hit, a “1” is recorded; otherwise, a “0” is recorded. The global access history register is the index to a global pattern history table which contains 2^k entries. Each entry is a two-bit saturating counter. In the implementation based on *PAg* version, each set has its own access history register. A single pattern history table is indexed by all the access history registers. In our experiments, the number of entries in the global pattern history table for both *GAg* and *PAg* predictors is set to the number of sets in the corresponding cache.

- *(M,N) correlation predictor.*

This is based on the scheme proposed in [55]. An M -bit shift register stores the hit/miss history for the most recent M accesses. Each set has 2^M entries which are indexed by the M -bit register. Each entry is an N -bit counter. We apply (2,2) predictor in the experiments.

- *gshare predictor.*

The *gshare* predictor is originally proposed in [53]. The global pattern history table is indexed by the exclusive OR of the global access history with the set index of current reference. The number of entries in the table is equal to the number of sets in the cache in our experiments.

4.5.2 Accuracy

| Access Mode Predictor | Size (bits) | | | Misprediction Rate (%) | | |
|-----------------------|-------------|------|------|------------------------|------|-------|
| | iL1 | dL1 | uL2 | iL1 | dL1 | uL2 |
| Saturating Counter | 256 | 256 | 8K | 0.11 | 5.68 | 14.44 |
| GAg | ~256 | ~256 | ~8K | 0.12 | 4.97 | 5.51 |
| PAg | ~1K | ~1K | ~56K | 0.13 | 3.94 | 3.83 |
| (2, 2) | ~1K | ~1K | ~32K | 0.11 | 5.27 | 13.43 |
| gshare | ~256 | ~256 | ~8K | 0.12 | 6.01 | 15.57 |

Table 4.1: Comparisons of sizes and misprediction rates of different access mode predictors. The separate 64 KB instruction cache (iL1) and the data cache (dL1) are 8-way with block size of 64 B. The unified L2 cache (uL2) is 8-way with block size of 128 B. The misprediction rates are the average over all the SPEC2000 programs.

Table 4.1 compares the sizes and misprediction rates of five different policies that are discussed above on the system with 8-way 64 KB instruction/data caches and an 8-way 4 MB L2 cache. In our experiments, for *saturating counter*, a 128-entry 2-bit (32-byte) table is used for instruction and data caches, and a 4096-entry 2-bit (1-KB) table is used for L2 caches. For *GAg* and *gshare*, both the instruction cache and the data cache have a 32-byte global pattern history table and a 7-bit global access history register; the L2 cache has a 1 KB global pattern history table and a 12-bit global access history register. For *PAg*, both instruction and data caches have a 32-byte global pattern history table and a 112-byte local access history register table; the L2 cache has a 1-KB global pattern history table and a 6144-byte local access history register table. For *(2,2)*, both instruction and data caches have a 128-byte table and a 2-bit register; the L2 cache has a 4-KB table and a 2-bit register.

The values of misprediction rates presented are the average over all the twenty-six SPEC2000 programs. The two-level adaptive predictor *PAg* obtains the lowest misprediction rates which are below 4% on instruction, data, and L2 caches. However, it also

occupies more additional area than other predictors. The *GAg* predictor gains the second lowest misprediction rates which are below 6% on both L1 and L2 caches and requires the smallest additional area.

4.5.3 Overhead

We decide to select the *GAg* predictor in the remaining of our experiments for several reasons. The prediction accuracy of *GAg* is only slightly lower than that of *PAg*. However, the *GAg* predictor requires much less additional space for recording access history than the *PAg* predictor. For *GAg* predictor, the 8-way 64 KB instruction and data caches only need a 32-byte global pattern history table and a 7-bit global access history register each; the 4 MB L2 cache only requires a 1 KB global pattern history table and a 12-bit global access history register. Regarding the area, the overhead of *GAg* predictors for instruction and data caches is only 0.05%, and is only 0.02% for L2 cache.

More importantly, unlike other predictors, the *GAg* predictor does not require the address of the next cache reference to perform the mode prediction. Thus, the access mode can be selected before the next cache reference arrives. Since the access time of the small prediction table is less than the access time of the cache, there is no timing overhead introduced by the *GAg* predictor. Regarding the energy consumption, our simulation results indicate that the overall overhead on instruction, data, and L2 caches by using mode predictors is under 0.8% for all the programs.

4.6 Multicolumn-based Way-prediction

In the previous section, we discuss the selection of access mode predictors. The effectiveness of the AMP cache on energy reduction depends not only on the accuracy of access mode predictor, but also on the underlying way-prediction technique. In this section, we will introduce a way-prediction technique based on multicolumn caches [83].

Way-prediction caches were originally proposed for reducing the average access latency of set associative caches [22, 2, 44, 1, 18, 83]. Some prediction strategies are used to predict the way containing the desired data and read from that way first. Hash-rehash [2], MRU [22, 44], column-associative [1], predictive sequential associative [18], and multicolumn [83] caches are a few such examples.

Way-prediction techniques can also be beneficial for power saving in set-associative caches [39]. If the way-prediction is correct, which is called a *first-hit* in this paper, only the desired way is probed. In this case, the power consumption is close to that in a direct-mapped cache. However, if the prediction is wrong, the way-prediction cache will consume more energy than a conventional implementation because some additional operations are performed to maintain the prediction mechanism. Thus, the accuracy of the prediction technique, measured by the first-hit rate, is crucial to the reduction of power consumption.

4.6.1 Limitation of MRU-based Way-prediction

Authors in [39] propose to use the MRU way-prediction technique to reduce the power consumption for set-associative caches. Authors in [46] show that the hash-rehash, the column-associative, and the MRU way-prediction techniques effectively reduce the power

dissipation of two-way set-associative caches. Previous studies have shown that an MRU-based way-prediction cache is more power-efficient compared with other techniques [39, 46].

The MRU cache maintains an MRU list that marks the location of the most recently used (or accessed) block for each set. For any reference mapping into the set, the search always begins from the MRU location. This search order is effective for both level one and level two caches because of the temporal locality in reference streams [44]. For low-associativity caches, such as two-way or four-way caches, the MRU technique works well in predicting the desired way. However, as the cache associativity further increases, the MRU structure may potentially decrease the first-hit rate.

Figure 4.5 shows how an MRU cache searches the locations for two coming references on a four-way associative cache which has four sets. For clarity, only the tag and set portions of a reference are presented here. Thus the first reference (000101) maps to set “01” with tag “0001”.

Upon the first reference 000101, the MRU cache begins its search from way 1 according to the MRU information of set 1 (Figure 4.5). The predicted way does not contain the desired data, then the MRU cache probes all the remaining ways as the second attempt. In this example, the first reference is a cache miss. Thus, the least recently used block at way 2 is replaced and the corresponding MRU information is updated to point to way 2. When the second reference to the same set, 101101, arrives, the search still begins from the MRU location, which is way 2 now. This is not a first hit. The MRU cache then probes all the remaining ways and finds the desired block at way 3. The MRU information is updated again to reflect the fact that the MRU location is way 3 now. In this example, neither of the two references is a first hit on the MRU cache.

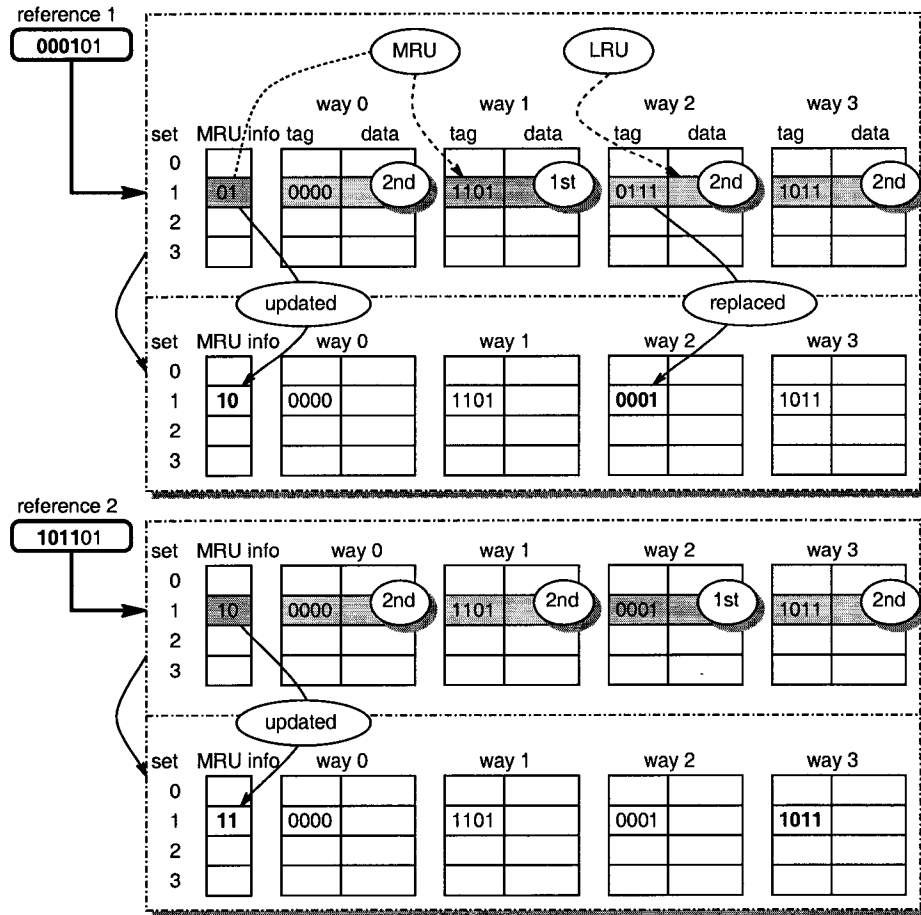


Figure 4.5: Way-prediction strategy for MRU cache.

In an MRU cache, the number of search entries is equivalent to the number of sets. As the cache associativity doubles for a fixed cache size and a fixed block size, the number of search entries in the MRU cache is halved. This means the number of entries available for way-prediction is reduced by half. The interference on the MRU information will be worsen. Normally, this causes the way-prediction accuracy to decrease. As shown in Figure 4.6, our experiments indicate that for the SPECint2000 program *172.mgrid*, as the associativity of a

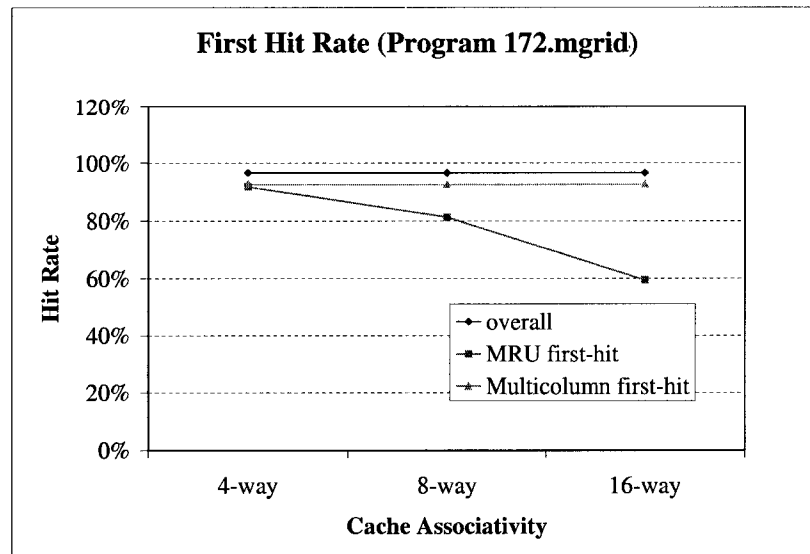


Figure 4.6: Overall cache hit rate and first hit rate of a 64 KB MRU data cache for program 172.mgrid.

64 KB L1 MRU data cache increases from four to eight, then to sixteen, the first hit rate of the cache drops from 91.8% to 81.3%, then to 59.4%, respectively. This trend is consistent with the results presented in [22]. In addition, the first hit rate of high-associativity MRU cache falls far behind the overall cache hit rate. This indicates that the MRU-based way-prediction is not optimal in power-saving.

4.6.2 Multicolumn Cache

The multicolumn cache [83] has addressed the limitation of MRU cache discussed above. The authors observe that the importance of a cache block in a set may be different to different references mapping into the same set. To consider the differences, The authors define a

concept of *major location* as the location on which a reference can be directly mapped⁴, and use the major location as the guidance for the access sequence. The multicolumn cache maps a block as in a direct-mapped cache but replaces a block as in a set-associative cache.

As the associativity increases, the first-hit rate does not change much in the multicolumn cache. The number of search entries in the multicolumn cache equals the product of the number of sets and the cache associativity. Thus, as the associativity increases for a fixed cache size and a fixed block size, the number of search entries in the multicolumn cache remains the same. Using the same program *172.mgrid* as example, as the associativity of a 64 KB L1 multicolumn data cache increases from four to eight, then to sixteen, the first-hit rate keeps the same (92.7%, 92.7%, and 92.7%, respectively, as shown in Figure 4.6). In addition, the first hit rate of multicolumn cache is very close to the overall cache hit rate. This indicates that the multicolumn is a nearly optimal way-prediction policy, since the cache hit rate is the upper bound of first hit rate that can be achieved by any way-prediction policy.

Figure 4.7 shows the search sequence in a multicolumn cache for the same reference stream in Section 4.6.1. For the first reference 000101, its tag is “0001”. For an n -way set associative cache, the major location of a reference is determined by the low order $\log n$ bits of its tag. Thus the major location of the first reference is way 1. The search begins here. The predicted way does not contain the desired block, then all the remaining ways are probed as the second attempt. Since this reference is a cache miss, the LRU block at way 2 is replaced. However, the new block (000101) and the block already residing at way 1

⁴For an n -way set associative cache, the major location of a reference is determined by the low order $\log n$ bits of its tag.

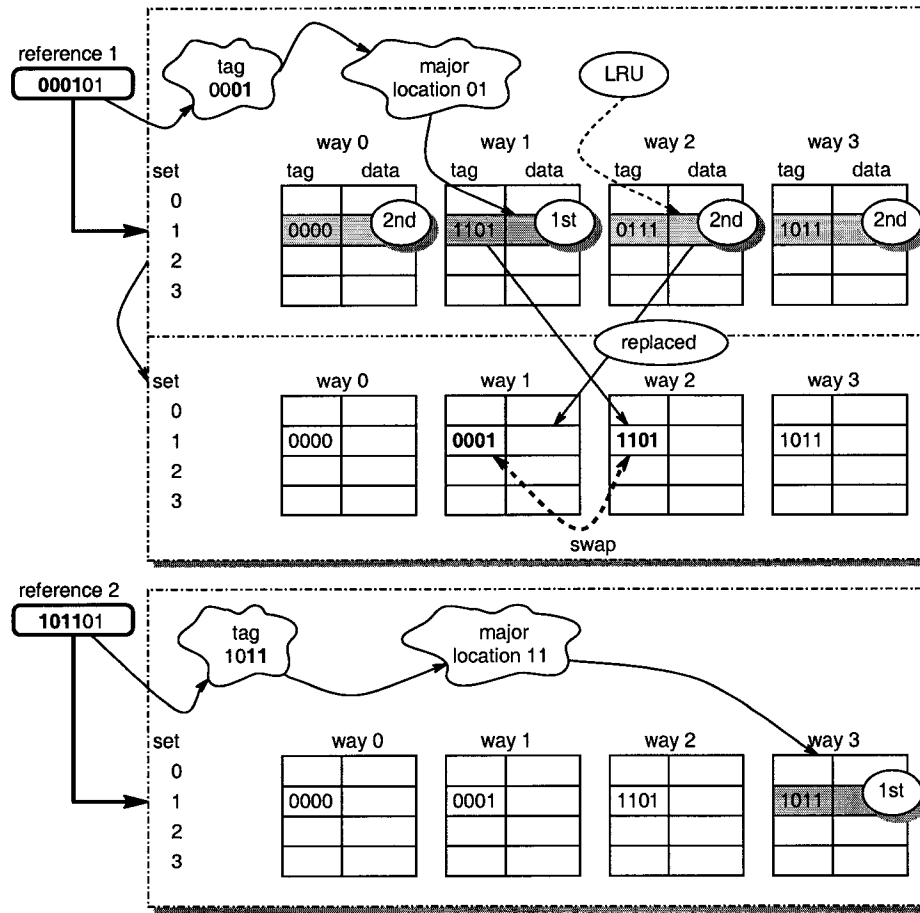


Figure 4.7: Way-prediction strategies for multicolumn cache.

(110101) have the same major location “01”. The design of multicolumn cache ensures that the MRU block is always at its major location. In this example, the new block (000101) is placed at its major location way 1, and the block originally at way 1 (110101) is swapped to way 2. For the second reference (101101), its major location is way 3. Thus the tag and data portion of way 3 are probed as the first attempt. This is a first hit. In comparison, the second reference is a non-first hit in the MRU implementation due to the interference

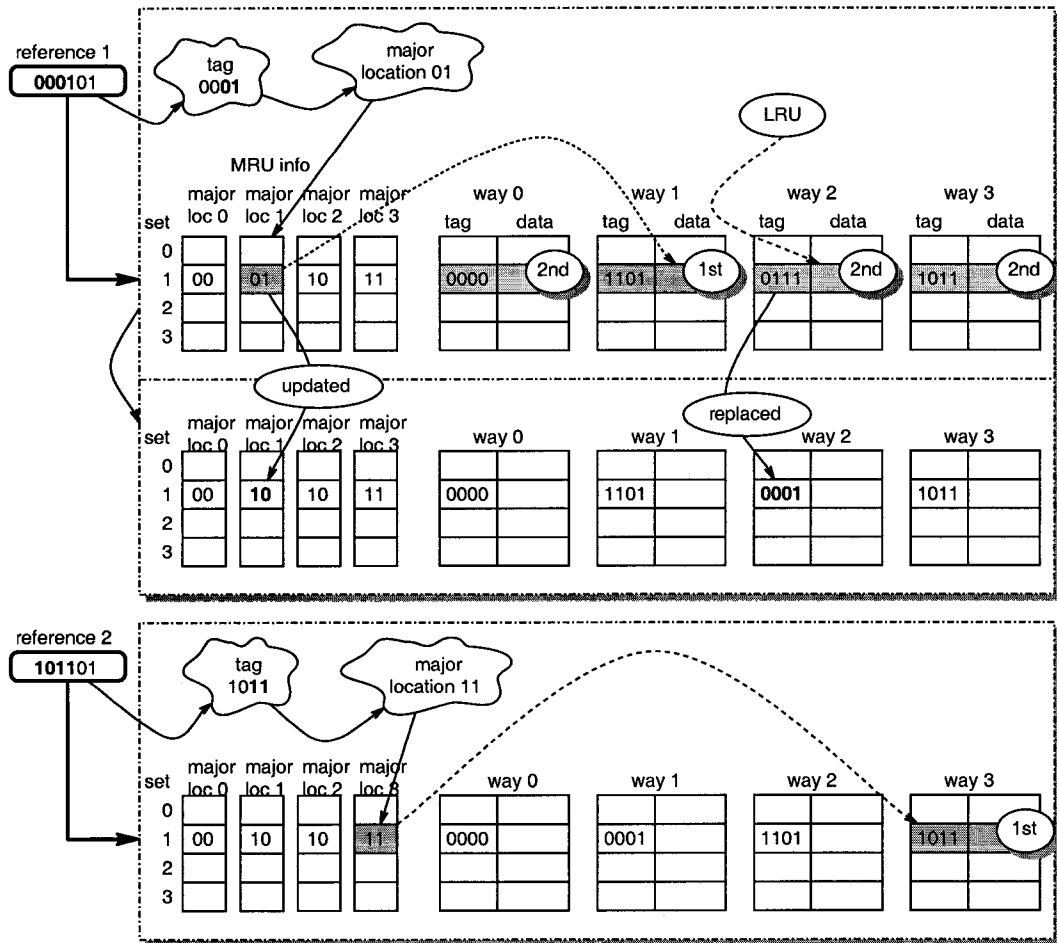


Figure 4.8: Way-prediction strategy for multicolumn cache without swapping.

on MRU information.

4.6.3 Power Considerations for Multicolumn Cache

In the original design of multicolumn caches, a swapping mechanism is applied to ensure that the MRU block always resides at its major location after a reference (though other data blocks may replace it later). From power consumption point of view, swapping is an

expensive operation. Two cache ways are involved in each swapping operation. The power consumption approximately equals the cost of two accesses to a single way.

In order to eliminate the swapping operation, we propose a power-efficient variation for the multicolumn cache. An index entry is maintained for each major location in a set to record its MRU information.

Figure 4.8 shows the implementations of multicolumn caches without swapping. We still use the same access sequence used in Section 4.6.1 as the example. For the first reference (000101), its tag is “0001”, and its major location is “01”. The MRU information of major location 1 at set 1 is retrieved, which points to way 1. Thus, the search begins from here. The predicted way does not contain the desired block, then all the remaining ways are probed as the second attempt. This reference is a cache miss, the LRU block at way 2 is replaced. However, the new block (000101) and the block already residing at way 1 (110101) have the same major location 1. Since the new block (000101) is the most recently accessed block whose major location is 1, the MRU information of major location 1 needs to be updated to reflect this change. In the implementation without swapping, the MRU information of major location 1 is updated to point to way 2 where the new block resides. By recording the MRU information for each major location, the MRU block does not need to always reside at its major location. Thus, the energy-consuming swapping operations are avoided. For the second reference (101101), its major location is 3 and the corresponding MRU information points to way 3. The tag and data portions of way 3 are probed as the first attempt. This is a first hit. No swapping is performed and the MRU information keeps unchanged.

There are several performance and energy consumption trade-offs in the implementations

of multicolumn caches. For the original design, the search always begins from the major location of a reference, which is determined by a simple bit selection on the reference address. Thus, the decision of a search order will not lengthen the cache access time, and will not consume additional energy. However, to maintain this way-prediction mechanism, the swapping operation may be performed. This swapping operation is energy-consuming, and may also delay subsequent coming references.

On the other hand, the implementation without swapping avoids the energy-consuming operation at the price of additional cache area for recording MRU information, of additional energy consumption for retrieving MRU information, and of possible increase on cache access time. Just like the MRU cache, the multicolumn cache without swapping needs to fetch the MRU information first. Since the reference address is used to index the MRU information table, the retrieval of MRU information may lengthen the cache access time. However, the arguments for MRU cache can also be applied here. If the reference address could be available earlier, the cache access could begin at an earlier pipeline stage [18]. In addition, the overhead of fetching MRU information can be tolerated for L2 caches [44].

Regarding the area, the overhead of recording the MRU information is trivial. Compared with the MRU cache which has one MRU entry for each set, the multicolumn cache without swapping has a larger MRU table where each block has an MRU entry. However, the MRU table in the multicolumn cache still only accounts for a very small portion of the data array. The area overhead is $\log(\text{associativity})/(\text{block_size} \cdot 8)$. The cache block size in a modern computer system is normally at least 32 bytes. Thus, the area overhead is very small. For example, considering a sixteen-way set-associative cache with block size of 64 bytes, each set needs 64 bits to record the MRU information. The MRU table only accounts for less

than 1% of the data array.

One more dimension of the MRU table in the multicolumn cache also makes the access more complicated. It is necessary to first identify which one of the n MRU entries in a set to be fetched. This is determined by the major location of the reference. Since the determination is from a bit selection, the latency of fetching the MRU information in the multicolumn cache is comparable to that in the MRU cache. Furthermore, the bit selection also makes it possible to only probe the desired entry. As the result, the power consumption for indexing the MRU table in the multicolumn cache is also comparable to that in the MRU cache. The energy consumption overhead of accessing such a small MRU table is also trivial.

We evaluate the frequency of swapping operations in eight-way multicolumn caches under the default system configuration in our experiments. For all the SPEC2000 benchmarks, on average, only 0.2% of references to the instruction cache involve swapping operations, while 3.8% and 12.0% of references to the data and L2 caches involve swapping operations, respectively. Since for L1 instruction cache, the possibility of a reference involving a swapping operation is very low, it is more feasible to directly use the major location as the search guidance, in company with swapping operations for this latency-sensitive cache. On the other hand, the possibility of a reference to L2 cache involving a swapping operation is as high as 12.0%, which means about 20% additional energy will be consumed by the swapping operation. In this case, it seems more feasible to apply the multicolumn implementation without swapping and overlap the fetch of MRU information with the accesses to L1 caches.

In summary, the original multicolumn cache design has no overhead on cache access latency. However, it requires energy-consuming swapping operations to maintain the way-

prediction mechanism. The multicolumn implementation without swapping eliminates the swapping operations at the cost of trivial increase on cache area and access latency. In our experiments presented in the following sections, we will utilize the multicolumn cache with swapping for instruction and data caches which are latency-sensitive and involve relatively a small percentage of swapping operations, and apply the implementation without swapping for L2 cache which is not so latency-sensitive but involves a large percentage of swapping operations.

4.7 Experimental Environment

CACTI [79] is a timing model for on-chip caches. It has been extended to CACTI 2.0 [58] that includes both timing and power models. We use CACTI 2.0 to estimate the power consumption of different implementations of set-associative caches in our study. For caches with access mode predictions, we estimate the energy consumption of first-hits, non-first-hits, misses, and write-backs in both way-prediction and phased accessing modes, respectively. We use the SimpleScalar tool [14] to collect the cache access and program execution statistics. We modified the cache simulator in order to emulate the behavior of different way-prediction and mode prediction approaches.

SPEC CPU2000 is a comprehensive benchmark suite that contains compute intensive benchmarks exercising a wide range of hardware. We use the precompiled Alpha version of SPEC2000 binaries [77]. The reference input data files are used in the experiments. We fast-forward the first four billion instructions, then collect detailed statistics on the next one billion instructions.

In our execution-driven simulation, the memory hierarchy is the main focus. The simulated 1 GHz 8-issue processor has separate 64 KB instruction and data caches with block size of 64 Bytes, and a unified 4 MB L2 cache with block size of 128 Bytes. This configuration is similar to those used in high-end workstations such as Alpha 21264, and Ultra Sparc III. The associativity of L1 and L2 caches ranges from four to sixteen. The caches have two sub-banks in each data RAM [67]. We assume that 0.18 μm technology is used.

4.8 Experimental Results

4.8.1 Comparisons of Multicolumn and MRU Caches

As we have stated in previous sections, the first-hit rate is crucial for power reduction of way-prediction caches. In this section, we will first compare the first-hit rate of the multicolumn-based way-prediction with that of the MRU-based way-prediction for set-associative caches.

Figure 4.9 shows the difference of first-hit rates between the multicolumn and MRU caches as the cache associativity increases. For clarity, this figure only presents the average first-hit rates of all the SPEC2000 programs and the first-hit rates of two representative programs *vpr* (an integer program) and *facerec* (a floating-point program). The overall hit rates and the first-hit rates of MRU and multicolumn instruction, data, and L2 caches on each program are presented in Table 4.2 to Table 4.4, respectively.

The first-hit rates of MRU caches decrease as the cache associativity increases. In contrast, the first-hit rates of multicolumn caches keep almost unchanged. For example, as the cache associativity increases to sixteen, the first-hit rate of the MRU L2 cache is only 40% of that of the multicolumn L2 cache for program *vpr*. In addition, the first-hit rates of

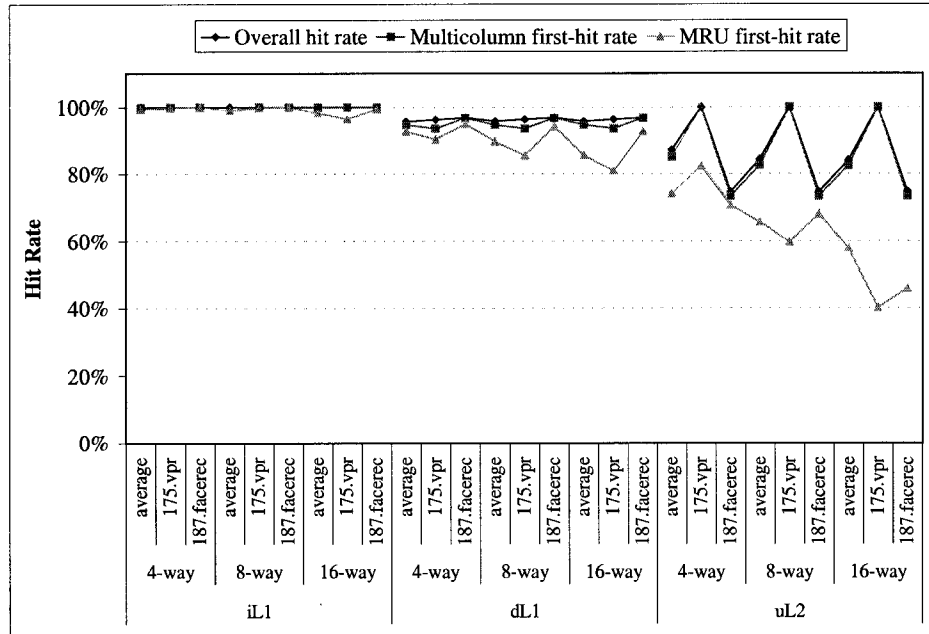


Figure 4.9: Comparisons of the first hit rates among the multicolumn and MRU caches.

multicolumn caches are very close to the overall cache hit rates, which are the upper bounds of first-hit rates, at different cache levels and for different cache organizations. The average first-hit rates of multicolumn caches are above 98% of the average overall cache hit rates. On the other hand, the first-hit rates of MRU caches have noticeable differences compared with the overall cache hit rates. For the sixteen-way L2 cache, the average first-hit rate of MRU-based way-prediction is only 69% of the average overall hit rate.

In terms of first-hit rates, the multicolumn-based way-prediction is a nearly optimal way-prediction technique. However, high first-hit rate is not our final target. Figure 4.10 illustrates the effectiveness of multicolumn-based way-prediction in reducing cache energy

| program | four-way | | | eight-way | | | sixteen-way | | |
|--------------|----------|---------|--------|-----------|---------|--------|-------------|---------|--------|
| | overall | mul-col | MRU | overall | mul-col | MRU | overall | mul-col | MRU |
| 164.gzip | 1.0000 | 1.0000 | 0.9955 | 1.0000 | 1.0000 | 0.9955 | 1.0000 | 1.0000 | 0.9955 |
| 175.vpr | 1.0000 | 1.0000 | 0.9975 | 1.0000 | 1.0000 | 0.9975 | 1.0000 | 1.0000 | 0.9637 |
| 176.gcc | 0.9996 | 0.9961 | 0.9898 | 0.9998 | 0.9962 | 0.9851 | 0.9999 | 0.9962 | 0.9824 |
| 181.mcf | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 186.crafty | 1.0000 | 0.9997 | 0.9750 | 1.0000 | 0.9997 | 0.9564 | 1.0000 | 0.9997 | 0.9404 |
| 197.parser | 1.0000 | 1.0000 | 0.9995 | 1.0000 | 1.0000 | 0.9939 | 1.0000 | 1.0000 | 0.9914 |
| 252.eon | 1.0000 | 0.9979 | 0.9897 | 1.0000 | 0.9979 | 0.9802 | 1.0000 | 0.9979 | 0.9639 |
| 253.perlbnk | 0.9993 | 0.9969 | 0.9895 | 0.9998 | 0.9969 | 0.9774 | 1.0000 | 0.9970 | 0.9706 |
| 254.gap | 1.0000 | 0.9988 | 0.9943 | 1.0000 | 0.9988 | 0.9867 | 1.0000 | 0.9988 | 0.9795 |
| 255.vortex | 0.9961 | 0.9897 | 0.9694 | 0.9981 | 0.9904 | 0.9615 | 0.9986 | 0.9906 | 0.9474 |
| 256.bzip2 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9889 |
| 300.twolf | 1.0000 | 0.9990 | 0.9909 | 1.0000 | 0.9990 | 0.9869 | 1.0000 | 0.9990 | 0.9800 |
| 168.wupwise | 1.0000 | 1.0000 | 0.9981 | 1.0000 | 1.0000 | 0.9963 | 1.0000 | 1.0000 | 0.9837 |
| 171.swim | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9999 |
| 172.mgrid | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9903 |
| 173.applu | 1.0000 | 1.0000 | 0.9987 | 1.0000 | 1.0000 | 0.9942 | 1.0000 | 1.0000 | 0.9934 |
| 177.mesa | 1.0000 | 0.9990 | 0.9894 | 1.0000 | 0.9990 | 0.9785 | 1.0000 | 0.9990 | 0.9577 |
| 178.galgel | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 179.art | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 183.equake | 1.0000 | 0.9962 | 0.9860 | 1.0000 | 0.9963 | 0.9642 | 1.0000 | 0.9963 | 0.9531 |
| 187.facerec | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9992 | 1.0000 | 1.0000 | 0.9937 |
| 188.ammp | 1.0000 | 0.9995 | 0.9992 | 1.0000 | 0.9995 | 0.9983 | 1.0000 | 0.9995 | 0.9974 |
| 189.lucas | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| 191.fma3d | 0.9996 | 0.9970 | 0.9913 | 1.0000 | 0.9971 | 0.9873 | 1.0000 | 0.9971 | 0.9802 |
| 200.sixtrack | 1.0000 | 0.9991 | 0.9991 | 1.0000 | 0.9991 | 0.9987 | 1.0000 | 0.9991 | 0.9845 |
| 301.apsi | 1.0000 | 0.9995 | 0.9911 | 1.0000 | 0.9995 | 0.9863 | 1.0000 | 0.9995 | 0.9834 |
| Average | 0.9998 | 0.9988 | 0.9940 | 0.9999 | 0.9988 | 0.9894 | 0.9999 | 0.9988 | 0.9816 |

Table 4.2: The overall hit rates and first hit rates of multi-column and MRU structures for L1 instruction cache. The cache is 64 KB with block size of 64B.

consumption. This figure shows the reduction of overall energy consumed by instruction cache, data cache, and L2 cache under the multicolumn-based way-prediction compared with the MRU-based way-prediction. Compared with the MRU-based way-prediction, the multicolumn-based technique can reduce the overall energy consumption of four-way instruction, data, and L2 caches by 0.1% to 39.6% (6.8% on average). As cache associativity increases to eight, the energy reduction is 0.4% to 50.3% (16.6% on average). For sixteen-way caches, the energy reduction of multicolumn caches is even more promising, which

| program | four-way | | | eight-way | | | sixteen-way | | |
|--------------|----------|---------|--------|-----------|---------|--------|-------------|---------|--------|
| | overall | mul-col | MRU | overall | mul-col | MRU | overall | mul-col | MRU |
| 164.zip | 0.9896 | 0.9840 | 0.9692 | 0.9898 | 0.9839 | 0.9574 | 0.9900 | 0.9839 | 0.9351 |
| 175.vpr | 0.9617 | 0.9343 | 0.9027 | 0.9621 | 0.9343 | 0.8539 | 0.9624 | 0.9343 | 0.8083 |
| 176.gcc | 0.9512 | 0.9500 | 0.9453 | 0.9512 | 0.9500 | 0.9406 | 0.9513 | 0.9500 | 0.9326 |
| 181.mcf | 0.8087 | 0.8039 | 0.7952 | 0.8084 | 0.8035 | 0.7841 | 0.8079 | 0.8032 | 0.7488 |
| 186.crafty | 0.9942 | 0.9677 | 0.9252 | 0.9967 | 0.9683 | 0.8473 | 0.9973 | 0.9684 | 0.7851 |
| 197.parser | 0.9803 | 0.9720 | 0.9455 | 0.9809 | 0.9719 | 0.9170 | 0.9811 | 0.9720 | 0.8794 |
| 252.eon | 0.9998 | 0.9945 | 0.9733 | 0.9999 | 0.9945 | 0.9559 | 0.9999 | 0.9945 | 0.9172 |
| 253.perlbnk | 0.9977 | 0.9922 | 0.9742 | 0.9985 | 0.9925 | 0.9616 | 0.9989 | 0.9925 | 0.9403 |
| 254.gap | 0.9973 | 0.9964 | 0.9921 | 0.9973 | 0.9964 | 0.9874 | 0.9973 | 0.9964 | 0.9773 |
| 255.vortex | 0.9929 | 0.9820 | 0.9623 | 0.9939 | 0.9821 | 0.8928 | 0.9948 | 0.9822 | 0.8734 |
| 256.bzip2 | 0.9821 | 0.9772 | 0.9443 | 0.9824 | 0.9772 | 0.9331 | 0.9826 | 0.9772 | 0.9159 |
| 300.twolf | 0.9490 | 0.9411 | 0.9123 | 0.9490 | 0.9409 | 0.8920 | 0.9494 | 0.9410 | 0.8610 |
| 168.wupwise | 0.9893 | 0.9746 | 0.9675 | 0.9893 | 0.9746 | 0.9611 | 0.9893 | 0.9746 | 0.9513 |
| 171.swim | 0.9175 | 0.9145 | 0.9134 | 0.9149 | 0.9145 | 0.9119 | 0.9149 | 0.9145 | 0.8869 |
| 172.mgrid | 0.9665 | 0.9270 | 0.9179 | 0.9665 | 0.9270 | 0.8131 | 0.9665 | 0.9270 | 0.5935 |
| 173.applu | 0.9431 | 0.9351 | 0.9185 | 0.9432 | 0.9352 | 0.8958 | 0.9432 | 0.9352 | 0.8651 |
| 177.mesa | 0.9962 | 0.9954 | 0.9799 | 0.9962 | 0.9954 | 0.9405 | 0.9963 | 0.9954 | 0.9050 |
| 178.galgel | 0.9884 | 0.9706 | 0.8783 | 0.9884 | 0.9706 | 0.8056 | 0.9884 | 0.9706 | 0.6926 |
| 179.art | 0.6599 | 0.6565 | 0.6457 | 0.6599 | 0.6565 | 0.6323 | 0.6599 | 0.6565 | 0.6072 |
| 183.quake | 0.9999 | 0.9999 | 0.9893 | 0.9999 | 0.9998 | 0.9753 | 0.9999 | 0.9998 | 0.9154 |
| 187.facerec | 0.9672 | 0.9657 | 0.9476 | 0.9672 | 0.9657 | 0.9411 | 0.9672 | 0.9657 | 0.9260 |
| 188.ampp | 0.9371 | 0.9295 | 0.9115 | 0.9379 | 0.9296 | 0.8989 | 0.9384 | 0.9297 | 0.8726 |
| 189.lucas | 0.9188 | 0.9145 | 0.9034 | 0.9188 | 0.9145 | 0.8882 | 0.9188 | 0.9145 | 0.8585 |
| 191.fma3d | 1.0000 | 0.9996 | 0.9925 | 1.0000 | 0.9996 | 0.9726 | 1.0000 | 0.9996 | 0.9518 |
| 200.sixtrack | 0.9880 | 0.9774 | 0.9371 | 0.9882 | 0.9775 | 0.8805 | 0.9882 | 0.9775 | 0.8080 |
| 301.apsi | 0.9719 | 0.9337 | 0.8899 | 0.9720 | 0.9337 | 0.8626 | 0.9720 | 0.9337 | 0.8257 |
| Average | 0.9557 | 0.9457 | 0.9244 | 0.9559 | 0.9458 | 0.8963 | 0.9560 | 0.9458 | 0.8552 |

Table 4.3: The overall hit rates and first hit rates of multi-column and MRU structures for L1 data cache. The cache is 64 KB with block size of 64B.

ranges from 5.1% to 67.3% (40.0% on average).

4.8.2 Energy Reduction of AMP Caches

Figure 4.11 compares the energy consumption of multicolumn cache, phased cache, and the AMP cache. We have measured the energy consumed by four-way instruction, data, and eight-way L2 caches. All the values are normalized to the energy consumed by the multicolumn implementation. The results show that the relative energy consumption of

| program | four-way | | | eight-way | | | sixteen-way | | |
|--------------|----------|---------|--------|-----------|---------|--------|-------------|---------|--------|
| | overall | mul-col | MRU | overall | mul-col | MRU | overall | mul-col | MRU |
| 164.gzip | 0.9831 | 0.9816 | 0.9773 | 0.9827 | 0.9813 | 0.9713 | 0.9825 | 0.9811 | 0.9374 |
| 175.vpr | 0.9994 | 0.9994 | 0.8233 | 0.9994 | 0.9994 | 0.5972 | 0.9994 | 0.9994 | 0.4011 |
| 176.gcc | 0.9981 | 0.9969 | 0.8416 | 0.9981 | 0.9969 | 0.7289 | 0.9981 | 0.9969 | 0.6378 |
| 181.mcf | 0.7076 | 0.6879 | 0.5329 | 0.6926 | 0.6720 | 0.5068 | 0.6858 | 0.6653 | 0.4764 |
| 186.crafty | 0.9956 | 0.9909 | 0.9358 | 0.9922 | 0.9895 | 0.7615 | 0.9908 | 0.9903 | 0.5612 |
| 197.parser | 0.9700 | 0.9569 | 0.8405 | 0.9702 | 0.9561 | 0.7516 | 0.9701 | 0.9556 | 0.6512 |
| 252.eon | 0.9941 | 0.9940 | 0.9332 | 0.9905 | 0.9904 | 0.9844 | 0.9880 | 0.9879 | 0.9874 |
| 253.perlbnk | 0.9988 | 0.9941 | 0.9628 | 0.9977 | 0.9977 | 0.9126 | 0.9959 | 0.9958 | 0.8960 |
| 254.gap | 0.7590 | 0.7545 | 0.7401 | 0.7589 | 0.7545 | 0.7221 | 0.7590 | 0.7546 | 0.6896 |
| 255.vortex | 0.9812 | 0.9572 | 0.8869 | 0.9729 | 0.9448 | 0.7706 | 0.9669 | 0.9346 | 0.6591 |
| 256.bzip2 | 0.9806 | 0.9647 | 0.8126 | 0.9803 | 0.9637 | 0.6521 | 0.9791 | 0.9623 | 0.4663 |
| 300.twolf | 0.9995 | 0.9995 | 0.8252 | 0.9995 | 0.9995 | 0.6364 | 0.9995 | 0.9995 | 0.4880 |
| 168.wupwise | 0.6617 | 0.6484 | 0.5480 | 0.6460 | 0.6398 | 0.5213 | 0.6433 | 0.6382 | 0.4990 |
| 171.swim | 0.6983 | 0.6808 | 0.6802 | 0.7048 | 0.6878 | 0.6862 | 0.7048 | 0.6878 | 0.6859 |
| 172.mgrid | 0.7686 | 0.6956 | 0.6883 | 0.7678 | 0.6957 | 0.6826 | 0.7663 | 0.6957 | 0.5229 |
| 173.applu | 0.6645 | 0.6574 | 0.6484 | 0.6644 | 0.6567 | 0.6463 | 0.6644 | 0.6567 | 0.6307 |
| 177.mesa | 0.8909 | 0.8676 | 0.8365 | 0.8936 | 0.8686 | 0.7916 | 0.8931 | 0.8683 | 0.7132 |
| 178.galgel | 0.9930 | 0.9916 | 0.5660 | 0.9939 | 0.9919 | 0.5491 | 0.9939 | 0.9919 | 0.4934 |
| 179.art | 0.9998 | 0.9998 | 0.5669 | 0.9998 | 0.9998 | 0.4646 | 0.9998 | 0.9998 | 0.4273 |
| 183.equake | 0.7344 | 0.7348 | 0.7340 | 0.7343 | 0.7342 | 0.7237 | 0.7342 | 0.7342 | 0.7052 |
| 187.facerec | 0.7460 | 0.7328 | 0.7065 | 0.7460 | 0.7328 | 0.6810 | 0.7464 | 0.7328 | 0.4594 |
| 188.amp | 0.7407 | 0.5977 | 0.4367 | 0.6188 | 0.5444 | 0.3665 | 0.5972 | 0.5337 | 0.2916 |
| 189.lucas | 0.5550 | 0.4207 | 0.2955 | 0.6155 | 0.4810 | 0.3557 | 0.6118 | 0.5051 | 0.3855 |
| 191.fma3d | 0.9994 | 0.9994 | 0.9994 | 0.3659 | 0.3648 | 0.3659 | 0.3574 | 0.3591 | 0.3556 |
| 200.sixtrack | 0.9982 | 0.9949 | 0.7412 | 0.9982 | 0.9952 | 0.6656 | 0.9982 | 0.9951 | 0.5141 |
| 301.apsi | 0.8321 | 0.8292 | 0.6986 | 0.8310 | 0.8284 | 0.5947 | 0.8309 | 0.8283 | 0.5002 |
| Average | 0.8711 | 0.8511 | 0.7407 | 0.8429 | 0.8256 | 0.6573 | 0.8406 | 0.8250 | 0.5783 |

Table 4.4: The overall hit rates and first hit rates of multi-column and MRU structures for L2 cache. The cache is 4 MB with block size of 128B.

the multicolumn and phased caches is application-dependent. Among all the twenty-six programs, phased caches consume less energy than multicolumn caches for six programs. Compared with multicolumn caches, phased caches may consume up to 37% more energy or up to 40% less energy, depending on the program behavior. On average, phased caches consume 16.7% more energy than multicolumn caches.

For all the programs, the AMP caches consume less energy than multicolumn caches. The energy reduction is 8.6% on average and up to 46.2%. This indicates that the AMP

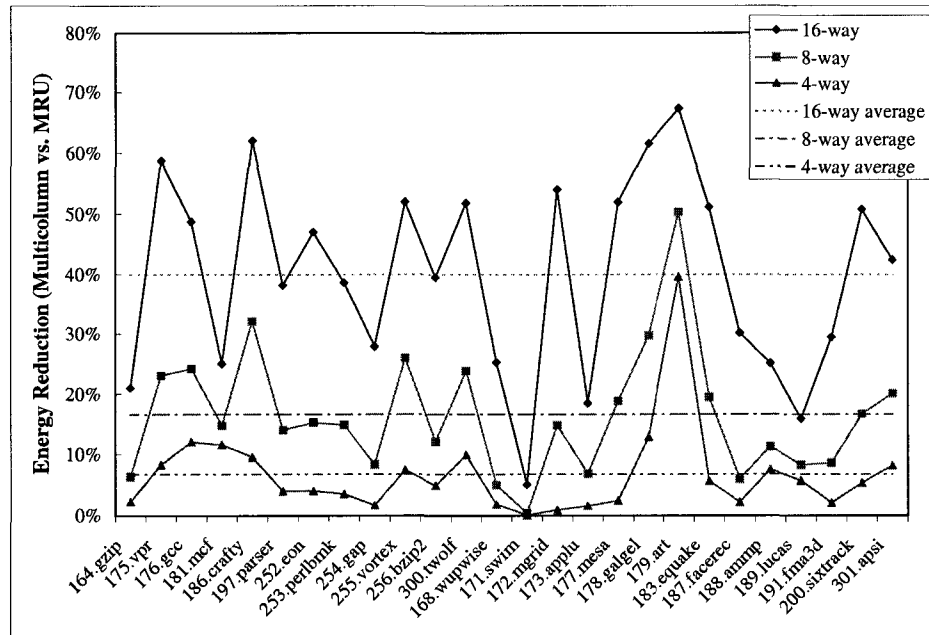


Figure 4.10: Energy reduction of multicolumn caches compared with MRU caches.

caches can effectively save energy on way-prediction misses and cache misses. Compared with phased caches, the AMP caches consume less energy for all the programs but one (*mcf*). This exception is due to the very high miss rates on data and L2 caches and the irregular access patterns of the program. For *mcf*, the AMP caches consume 31.1% more energy than phased caches. For other programs, the access mode prediction technique can reduce the energy consumption of phased caches by 10.6% to 27.1%. For all the programs, the average energy reduction of AMP caches is 20.0%, compared with phased caches. This indicates that the location of many references can be correctly predicted by the way-prediction policy and significant amount of energy can be saved.

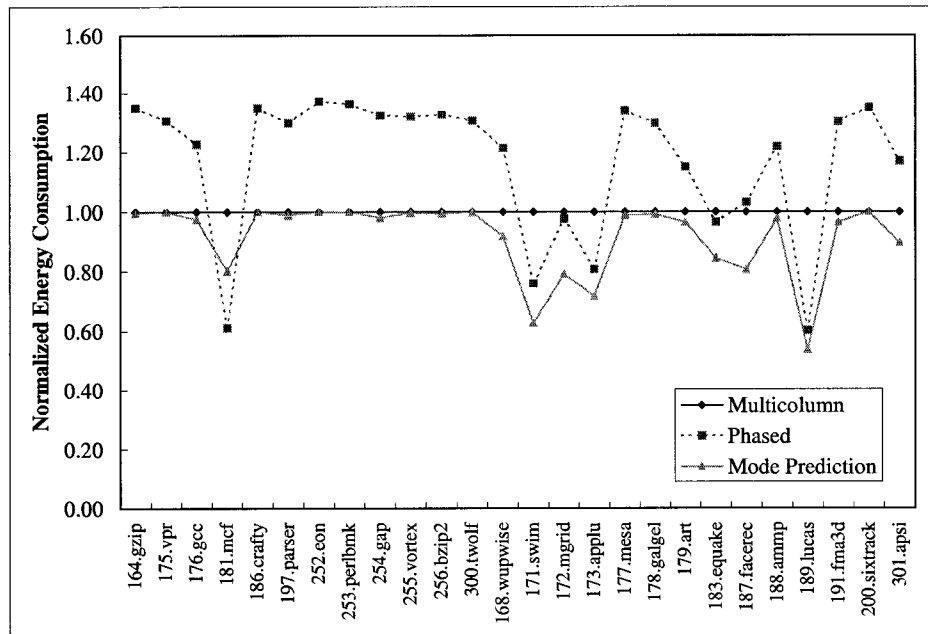


Figure 4.11: Energy consumption of multicolumn cache, phased cache, and AMP cache. The system has four-way 64 KB instruction and data caches and eight-way 4 MB L2 cache.

To show more details about energy reduction of caches with access mode predictions, we use six programs as examples and show the decomposition of energy consumed by instruction, data, and L2 caches in Figure 4.12. All of the values are normalized to the overall energy consumption of multicolumn caches. Looking into the decomposition of energy consumed by instruction, data, and L2 caches, we find that phased instruction caches consistently consume more energy than both multicolumn and AMP instruction caches. This is due to the high first-hit rates of multicolumn and AMP instruction caches. Compared with these two caches, the phased cache consumes more energy for first-hits. The relative energy

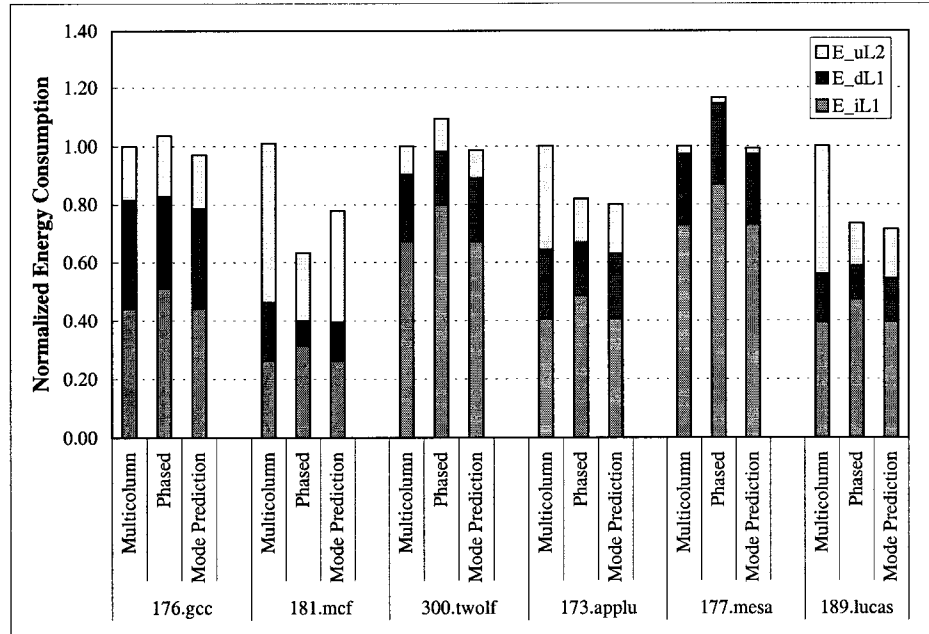


Figure 4.12: Decomposition of energy consumed by instruction, data, and L2 caches.

consumptions on data and L2 caches are application dependent. For example, *lucas* has high L2 cache miss rates. Since the phased cache has lower energy consumption for misses than the multicolumn cache, the L2 phased cache consumes much less energy than the L2 multicolumn cache. The cache with access mode predictions always intends to consume the lowest possible energy for both hits and misses. For program *lucas*, its energy consumption on instruction cache is comparable to that of multicolumn cache, and its energy consumption on L2 cache is comparable to that of phased cache. Our study show that the cache with access mode predictions is the most energy-efficient one.

The percentage of energy consumed by instruction phased caches is application-

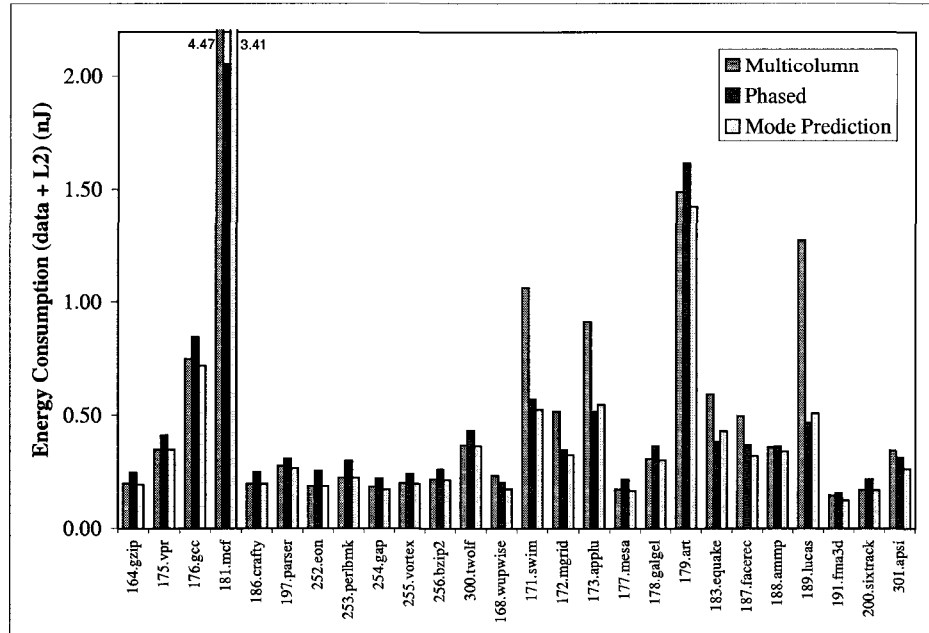


Figure 4.13: Energy consumed by data and L2 caches.

dependent. It ranges from 26.4% (for *art*) to 82.0% (for *fma3d*). Since the phased instruction cache always consumes more energy than the multicolumn and AMP instruction caches, to make a more fair comparison, we assume the instruction caches always have the same AMP structure.

Figure 4.13 compares the energy consumption per instruction by only data and L2 caches for the three different cache implementations. For seventeen of the twenty-six programs, the multicolumn data and L2 caches consume less energy than the phased data and L2 caches by up to 26.6%. For the other nine programs, the phased implementation consumes less energy than the multicolumn implementation by up to 63.4%. On average, their energy

consumption is comparable to each other. The phased implementation consumes 0.3% more energy than the multicolumn implementation on average. The AMP data and L2 caches consume less energy than the multicolumn data and L2 caches for all the programs by up to 60.2% and 14.4% on average. For twenty-two programs, the AMP caches consume less energy than the phased implementation by 6.1% to 26.7%. For the other four programs, the phased implementation consumes less energy than the AMP caches by 5.5% to 39.7%. On average, the AMP data and L2 caches consume 10.7% less energy than the phased data and L2 caches.

In summary, the multicolumn cache consumes less energy for applications with high cache hit rates, while the phased cache consumes less energy for applications with low hit rates. Since the hit rate is highly application-dependent especially at data cache and L2 cache, neither the multicolumn cache nor the phased cache works well for a wide range of applications in terms of energy reduction. In contrast, the AMP cache always intends to consume the lowest possible energy for both cache hits and misses. It consumes less energy than both the multicolumn and the phased caches for a wide range of applications.

4.8.3 Energy-efficiency of AMP Caches

Regarding the access latency, a correctly predicted hit in the multicolumn cache or in the AMP cache has shorter latency than an access in the phased cache. Table 4.5 presents the access latency for a way-prediction hit, a way-prediction miss, and a phased access by using the CACTI model. According to these obtained values, we set the cache access latency by processor cycle in our experiments as follows. For instruction and data caches, a way-prediction hit takes one cycle, a way-prediction miss or a phased access takes two cycles.

| | iL1/dL1 | | L2 | |
|---------------------|---------|----------|----------|-----------|
| Way-prediction Hit | 0.90 ns | 1 cycle | 5.85 ns | 6 cycles |
| Way-prediction Miss | 1.61 ns | 2 cycles | 11.39 ns | 12 cycles |
| Phased Access | 1.88 ns | 2 cycles | 10.77 ns | 12 cycles |

Table 4.5: Access latencies for the way-prediction hit/miss and phased access on 64 KB 4-way L1 and 4 MB 8-way L2 caches.

For L2 cache, a way-prediction hit takes six cycles, a way-prediction miss or a phased access takes twelve cycles.

There is a concern that the AMP cache may not efficiently handle access sequences with varied latencies. However, for the way-prediction caches such as MRU cache, it is also necessary to handle accesses with varied latencies. Thus, the AMP cache does not introduce additional complexity to deal with different access latencies compared with the way-prediction caches. For the same reason, the mixture of phased accesses with way-prediction hits does not introduce additional complexity in solving the conflicts on accessing data array, compared with the mixture of way-prediction misses with way-prediction hits.

Figure 4.14 and Figure 4.15 compare the performance and energy-delay product of the AMP cache with those of the multicolumn cache and the phased cache, respectively.

Compared with the multicolumn cache, the AMP cache may mispredict the access modes of some way-prediction hits and perform phased accesses. This only slightly increases the average cache access latency and degrades the overall performance. Among the twenty-six programs, the AMP cache gets the same CPI for seven programs compared with the multicolumn cache. The maximum CPI increase is 1%. The average CPI increase is only 0.1%. This indicates that only a very small percentage of way-prediction hits are mispredicted as phased accesses by the access mode predictor. The AMP cache obtains almost the same

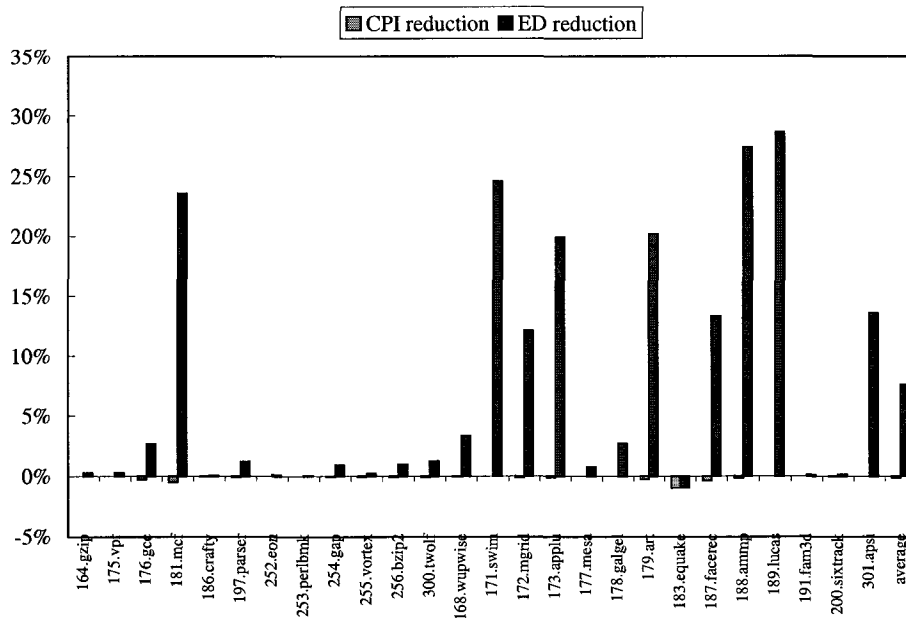


Figure 4.14: CPI and E-D product reductions of the AMP cache compared with the multicolumn cache.

performance as the multicolumn cache.

Way-prediction hits have shorter latencies than those of phased accesses. Thus, the average access latency in the AMP cache is shorter than that in the phased cache. Compared with the phased cache, the CPI reduction of the AMP cache ranges from 0.4% to 14.9%, and is 6.1% on average.

The AMP cache reduces the energy-delay product by 8.5% on average (up to 46.2%) compared with the multicolumn cache, and by 24.8% on average (up to 35.3%) compared with the phased cache.

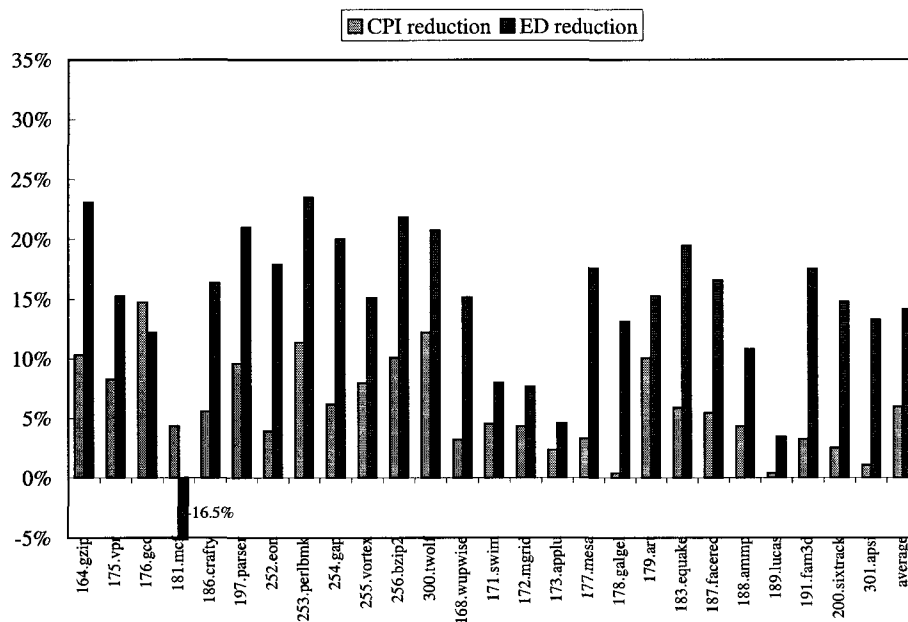


Figure 4.15: CPI and E-D product reductions of the AMP cache compared with the phased cache.

4.9 Summary

The way-prediction cache and phased cache are two existing techniques in reducing the power consumption of set-associative caches. However, neither minimizes energy consumption for both cache hits and misses. Their effectiveness in reducing energy consumption is highly application-dependent. In order to further reduce the energy consumption of set-associative caches, we propose a cache structure with access mode predictions (AMP cache) which combines the merits of way-prediction and phased accessing together. Our study shows the following results:

- With a simple access mode prediction based on cache hit/miss prediction, the AMP cache can effectively reduce the energy consumption for a wide range of applications under systems with moderate to high associativity caches. For example, the AMP cache reduces energy consumed by four-way L1 and eight-way L2 caches by 9% and 20% on average (up to 46% and 27%) compared with the multicolumn and phased implementations, respectively.
- The AMP cache is an energy-efficient design. It can reduce the energy-delay product of a system with four-way L1 and eight-way L2 caches by 9% and 25% on average (up to 46% and 35%) compared with the multicolumn and phased implementations, respectively.
- The multicolumn-based way-prediction technique is a nearly optimal scheme, which can correctly predict the locations of 98% of cache hits on average. For four-way, eight-way, and sixteen-way caches, multicolumn-based way-prediction can reduce the energy consumption by 7%, 17%, and 40% on average, respectively, compared with MRU-based way-prediction. In addition, the AMP cache can also utilize other way-prediction techniques.
- The AMP cache can exploit the same mechanism used in the way-prediction cache to handle the varied latencies from different access modes and way-prediction hits and misses.
- The additional overhead of access mode predictor and multicolumn-based way-prediction is negligible measured by cache area, access latency, and energy consumption.

Chapter 5

Look-ahead Adaptation Techniques to Reduce Processor Power Consumption

5.1 Motivation

The pursuing of high performance on general purpose processors has been increasing the speed as well as the complexity of processors. As a byproduct, the last decade has seen a dramatic increase of processor power consumption [78]. To address this issue, researchers have targeted reducing processor's power dissipation with minimum performance impacts. One effective approach at architectural level, called *architecture adaption*, is to adaptively activate and deactivate hardware resources in accord with the dynamic changes of a running program's execution behavior [17, 6, 29, 56, 61]. This is based on an observation that the program execution behavior varies significantly among different programs and among different execution phases of a single program [76].

There are two key factors in architecture adaptation [61]: (1) when to trigger the adap-

tation, i.e., under what conditions to activate/deactivate resources; and (2) what adaptation techniques to apply. Our work focuses on the first issue.

In order to reduce processor power consumption while retaining performance, most existing solutions try to meet the current program requirement with a minimum number of active resources. However, this type of approaches shares a common limitation: the adaptation is triggered after the change of demand has been detected. Without the foreknowledge of future demand variants, resources are kept active when the current demand is high, even if the demand is going to drop. Figure 5.1-(a) shows an example of power-saving optimization based on a current system status. The monitored value is compared with a threshold at the end of each sampling window (the dots). At time A, the scheme will put the processor into a normal execution mode based on the current knowledge that the measured value is higher than the threshold. However, it does not foresee that a slack exists between time B and D, where the processor is almost idle.

Ideally, the deactivation of hardware resources should start earlier than the drop point to maximize the power saving, subject to maintaining the same performance. As illustrated in Figure 5.1-(b), part of the work is delayed to time C. However, since the same amount of work is finished before time D, the overall performance remains the same although the processor stays at the low power mode for a longer time. Thus, the optimization based on current system status loses some power-saving opportunities compared with the look-ahead optimization.

In this study, we show that some hardware events can accurately predict future demand degradation thus hardware resources can be deactivated confidently in advance even if the current demand is high. Specifically, this study utilizes the events of main memory accesses

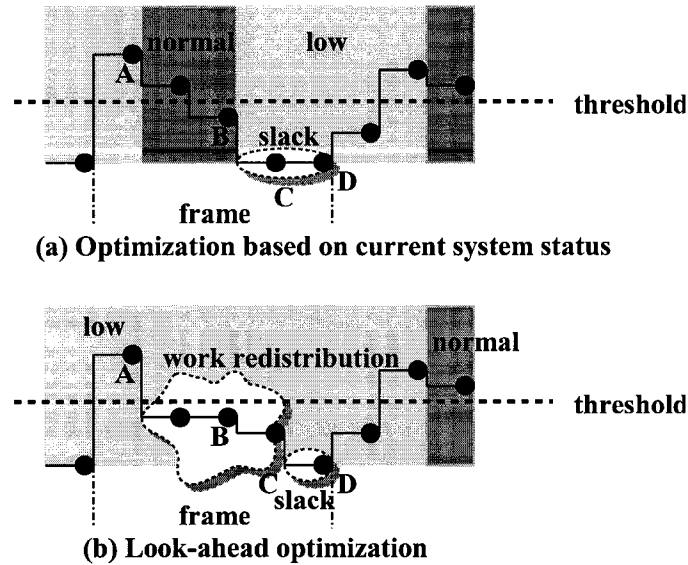


Figure 5.1: Comparisons of the optimization based on current system status and the look-ahead optimization.

to trigger architecture adaptation.

As the speed gap between processors and memory continues to widen, the memory access latency consistently increases compared with processor cycle time. Once an L2 cache load miss to main memory happens, it is almost certain that the processor will stall for this cache miss (although the processor may perform some useful work for subsequent instructions before stalling). For example, considering a moderate system configuration, a 4-way issue processor with a 128-entry instruction window runs at 2 GHz clock rate, and the memory access latency is 100 ns. Once a load miss happens, the load cannot be resolved within 200 cycles, while the instruction window will become full in 32 cycles at the full issue rate and this will force the processor to stall. Thus, when a load miss happens, maintaining the full processor issue width is wasteful even if the current program demand is high. In addition, this resource degradation period will enlarge as the speed gap between the processor and

memory continues to widen.

We propose a new scheme, called *load indicator*, that triggers the processor issue rate adaption with the existence of main memory accesses. In particular, the scheme reduces the issue width when a load miss occurs, and resumes the full issue rate when all outstanding loads finish. Previous studies have shown that adjusting the processor issue rate is an effective adaption technique for power-saving [6, 61]. With an accurate prediction on future demand degradation, the load indicator scheme redistributes the CPU work over a relatively long period of time to maximize the power saving while achieving the same performance.

Our experiments show that, for memory-intensive applications, this scheme can save more power with a performance loss comparable to that of the pipeline balancing technique [6] which periodically adjusts the processor issue rate based on the average issued IPC (instructions per cycle). For seven memory-intensive applications from SPEC2000 benchmark suites, our scheme can reduce the power consumption on the issue logic and execution units by 24% and 11%, respectively. The total processor power consumption is reduced by 5.4% on average with a performance loss of 0.5%, compared with a 4.2% average power saving with a performance loss of 0.7% by the pipeline balancing technique.

The load indicator scheme foresees the degradation on resource demands caused by memory accesses. Thus it only has effects on memory-intensive applications. We further propose two variants of the scheme to cover a wider range of applications. The first variant, called *load-instruction indicator* scheme, utilizes information of both load misses and the IPC changes (in the absence of load misses) to adjust the processor issue rate. The second variant, called *load-register indicator* scheme, reduces the processor issue rate when cache load misses happen or the number of free registers drops below a threshold, and resumes

the full issue rate when there are no outstanding load misses and the number of free registers increases above another threshold. Both variants captures power saving opportunities caused by program behavior changes and hardware constraints besides main memory accesses. They effectively save energy for applications with a wide range of memory stall portions.

5.2 Load Indicator Scheme

As indicated in [76], the execution behavior of programs varies significantly among different programs and among different execution phases of the same program. Thus, architectural adaptations based on program requirement variants are effective on reducing processor power consumption with a negligible performance loss [17, 6, 29, 56, 61].

5.2.1 Power Saving Opportunity

The primary target of modern general purpose processors is high performance. Modern processors can execute multiple instructions in each cycle to boost performance. Out-of-order execution is supported by most state-of-art general purpose processors to improve performance. These techniques improve processor performance significantly. As a simple example, for a system with 16 KB instruction/data caches (1 cycle access time) and 256 KB L2 cache (6 cycle access time), as the processor issue width increases from four to eight, the performance of SPEC2000 program *171.swim* improves by 25%; on the other hand, if the processor only supports in-order execution, the performance will degrade by 45%.

Although these techniques do improve the performance effectively, they also increase the energy consumption significantly. In order to execute multiple instructions concurrently, the

processor needs to consist multiple functional units. Compared with the single-issue processors, the duplicate of functional units also means multiplication on energy consumption of datapath. In order to support out-of-order execution, the issue logic of processor is much more complicated than that only supports in-order execution. For an n -way issue in-order execution processor, the issue logic only needs to check the dependency and data availability of the first n instructions in the instruction window. In comparison, for an n -way issue out-of-order execution processor, the issue logic needs to check all the m instructions in the instruction window and select n ready instructions whose data dependences are solved and the corresponding functional units are available. In general, $m \gg n$, the issue logic picks n ready instructions out of m entries. We can see that the issue logic of out-of-order execution processor is much more complicate than that of in-order execution processor. The increasing complexity of issue logic also corresponds to the increase on energy consumption.

As a consequence, the issue logic and execution units consume about 40% of the total processor power consumption. Previous studies have shown that adjusting the processor issue rate is an effective power-saving technique, which can effectively reduce the power consumed by the issue logic and execution units [6, 61]. Authors in [6] propose a technique, called *pipeline balancing*, that dynamically adjusts processor issue width for each sampling window based on the average issued IPC measured in the previous window. For convenience of discussions, we will call this technique as *instruction indicator* in following sections. Authors in [61] propose a technique that uses the mean functional unit utilization and the number of structural hazard over the last period to trigger the issue rate adaptation for the next period.

Those techniques share a common limitation. They capture the change of program

behavior after the change has happened. More importantly, they cannot foresee the resource requirement degradation. Thus, they only perform power optimization based on the current knowledge of program behavior. As shown in Figure 5.1, this property loses some opportunities in power saving.

If a scheme can accurately predict a future program demand degradation, it is safe to only satisfy a part of the currently demanded work. The main idea of our look-ahead optimization is to redistribute work over a relatively long period of time in order to generate more power saving opportunities while finishing the same amount of work over that period. In principle, this is similar to many power-saving schemes for multimedia applications. For multimedia applications, as long as the demanded work of a “frame” finishes within a deadline, the performance is considered to be the same regardless the work real completion time. Thus, redistributing the work within a frame using dynamic voltage scaling is effective in saving power and retaining performance [38, 61].

However, general-purpose applications do not have such a concept of “frame” as multimedia applications do. Thus, the question here is how to find time periods within which work can be redistributed to save energy while the same amount of work can be finished within the periods.

5.2.2 Load Indicator

There are several possible solutions to identify the performance degradation in advance, such as by providing hints from the application, compiler, or operation systems. For example, authors in [73] propose to use a compiler-driven static IPC prediction at the loop-level to guide the fetch throttling for energy saving. Our study explores a hardware-based indicator

with a simple implementation, catching current technology trend.

As the speed gap between processors and memory continues to widen, the memory access latency increases consistently relative to the processor cycle time. Researchers have proposed many techniques to reduce the average memory access latency, for example, deploying advanced DRAM technologies [25, 88], exploiting the DRAM row buffer locality [86, 28, 87], and reordering current memory accesses [59, 90]. However, even after applying those latency-reduction techniques, for modern multi-issue, multi-GHz processors, once a cache load miss to main memory happens, it is almost certain that the processor will stall due to the long main memory access time. Thus, maintaining a partial processor issue width is enough to retain the performance from the time when the load miss happens to the time when the missing data are returned. The existence of load misses sends a signal of future reduction of program requirements on computation resources.

Figure 5.2 shows the sampled IPC values of program *swim* and the number of outstanding cache load misses in a representative 1024-cycle interval during the program's execution. For clarity, we only present the floating point IPC values in the figure. The integer IPC values have a similar pattern. The system configuration will be presented in details in Section 5.4. From the figure, we can see that normally multiple cache load misses happen together. From the starting time when the first cache load miss occurs to the ending time when all the load misses return, the program execution behavior forms a "frame": an active period followed by an idle period. This result shows that the information of load misses can be used to accurately predict the future performance degradation.

This figure also presents the measured IPC values under different sampling window widths. Larger window size smooths out spurs and avoids thrashing between different

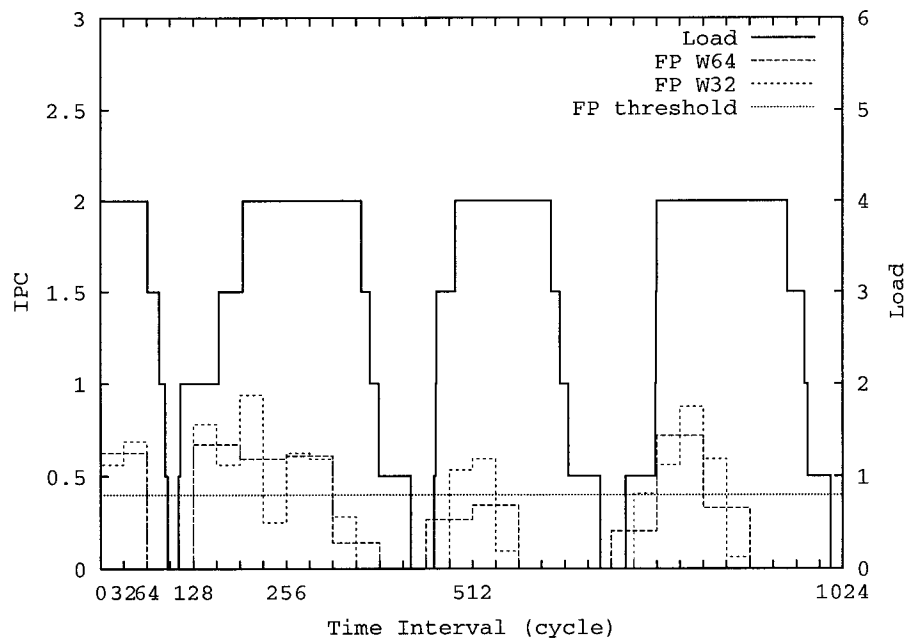


Figure 5.2: Sampled IPC values and number of outstanding loads during an arbitrarily selected 1024-cycle interval for program *swim*. “W64” and “W32” correspond to the sample window sizes of 64 cycles and 32 cycles, respectively.

power modes (e.g. from cycle 192 to cycle 256). However, larger window size may also fail to capture the change of program behavior (e.g. from cycle 480 to cycle 544). Thus, the sampling window size must be tuned carefully.

We propose a scheme, called *load indicator*, that saves power by reducing the issue width when one or more load misses occur, and retains performance by resuming the full issue rate when there are no outstanding load misses¹. In the implementation, a register is used to record the number of outstanding L2 cache load misses. When a load miss happens, if the processor is in the normal execution mode (i.e., with the full issue rate), the processor will transit to the low power mode. In the low power mode, the processor issue rate and the

¹We only consider cache load misses since normally write misses can be well handled by write buffers and will not directly cause the processor stall.

number of active functional units are reduced by half. When a load miss is returned or a speculated load miss is squashed², the register value is decreased by one. When the register value drops to zero, the processor will return to the normal execution mode immediately.

```
if (a cache load miss happens)
    miss register value + 1;
    if (processor is in normal mode)
        processor transfers to the low power mode;

if (a cache load miss resolves)
    miss register value - 1;
    if (miss register value = 0 AND processor is in low power mode)
        processor returns to the normal mode;
```

The load indicator can identify the execution periods when a program does not require the full processing power. Several techniques can be applied during those periods to reduce the processor power consumption. For instance, if the processor has a dual speed pipeline structure [57, 62], instructions can be issued to the slow pipeline when cache load misses happen. In this study, we apply the technique of reducing the processor issue rate because this is a very effective technique and can be applied to most architectures.

The complexity of the load indicator scheme is low, and the overhead is trivial. Only one register is added to record the number of outstanding loads. The control logic is also very simple because it only checks the register value then makes the adaptation. There

²The scheme does not distinguish between true and speculative load misses for simplicity. Our experiments show that this only has small impact on power consumption and performance.

| Scheme | High demanding window | | | Low demanding window | | | Idle window |
|------------------|-----------------------|---------|---------|----------------------|---------|---------|-------------|
| | Percent | IPC_I | IPC_F | Percent | IPC_I | IPC_F | Percent |
| Normal execution | 60.4% | 0.9852 | 0.7620 | 17.7% | 0.2980 | 0.1945 | 21.9% |
| Load indicator | 55.9% | 0.9741 | 0.7644 | 23.8% | 0.4070 | 0.2660 | 20.3% |

Table 5.1: The distribution of high demanding, low demanding, and idle windows for program *swim*.

is some additional logic to adjust the processor issue rate. However, compared with clock gating at each component on a cycle-by-cycle basis, the adjustment of issue rate is at a coarser level. The additional power consumed by the scheme is negligible.

Table 5.1 presents the redistribution of work after applying the load indicator scheme using program *swim* as the example. Under normal execution, at 60.4% of sampling windows, either the issued integer IPC value IPC_I or the issued floating point IPC value IPC_F is higher than its corresponding threshold. On the other hand, the processor issues almost no instructions at 21.9% of sampling windows. This indicates that a large space is left to shift some of the work from the high demanding windows to the low demanding or idle windows. After applying the load indicator scheme, the percentage of sampling windows with high issued IPC values drops to 55.9%, while the percentage of sampling windows with low IPC values increases to 23.8%, and the percentage of idle windows drops to 20.3%. The average IPC values in low demanding windows also increase. This indicates that more work is done in low IPC windows, and the processor gets more chances to run in the low power mode.

5.3 Considerations for Load Indicator Scheme

A concern on the load indicator scheme is that reducing the processor issue rate when a main memory access happens will slow down finding the next main memory access and might degrade the performance. Another concern is that the existence of main memory accesses might make the processor issue rate switching too frequently. However, these two concerns do not affect the effectiveness of the load indicator scheme, because normally main memory accesses are clustered together.

In [90], we study the burstiness of main memory accesses. The system configuration is as follows. The processor is 4-way issue running at 2 GHz speed. The instruction and data caches are 4-way 64 KB with 2-cycle access latency, and the L2 cache is 4-way 1 MB with 8-cycle access latency. The memory system is a 2-channel Direct Rambus DRAM system with 60 ns access latency.

We first measure the arrival intervals of cache misses so as to gauge the burstiness in the miss stream. If a large fraction of accesses arrive in short intervals, that indicates the miss stream is highly bursty. Because the DRAM access delay will slowdown the instruction execution and the speed of the processor generating cache misses, we configure a perfect DRAM system that has the latency of L2 cache hit and an infinite bandwidth. Figure 5.3 shows the CDF (Cumulative Distribution Function) of the arrival interval of memory accesses of fifteen selected SPEC2000 programs. The top and the bottom figures contain the floating point and integer programs, respectively.

All of the applications have a dense distribution on short intervals. For example, *mcf* has 22% distribution at the zero cycle interval, which means that 22% of misses follow some

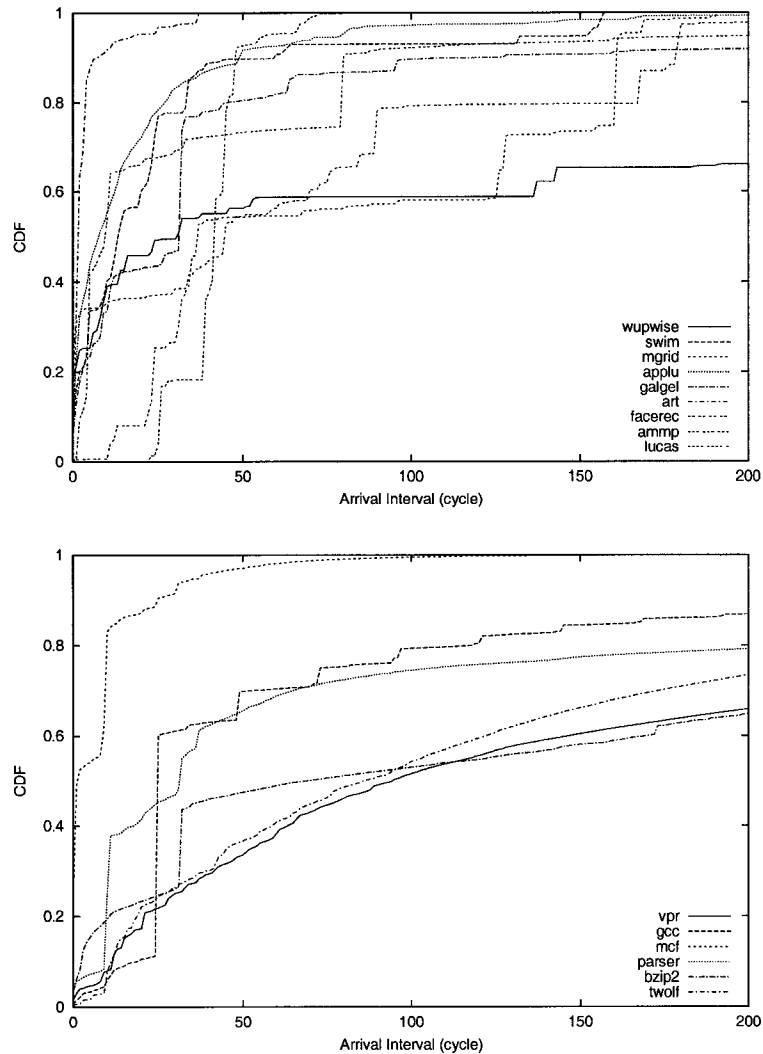


Figure 5.3: CDF of arrival interval of miss streams.

other misses at the same cycle. For *swim*, 10% of misses follow some other misses at the same cycle. For *mcf* and *swim*, the average arrival intervals are nine cycles and 25 cycles, respectively. Among the fifteen selected programs, eight of them have more than 25% of L2 misses clustered within seven cycle intervals, and four of them have more than 40% of L2 misses clustered within seven cycle intervals.

Figure 5.4 further presents the distribution of the number of concurrent accesses in the

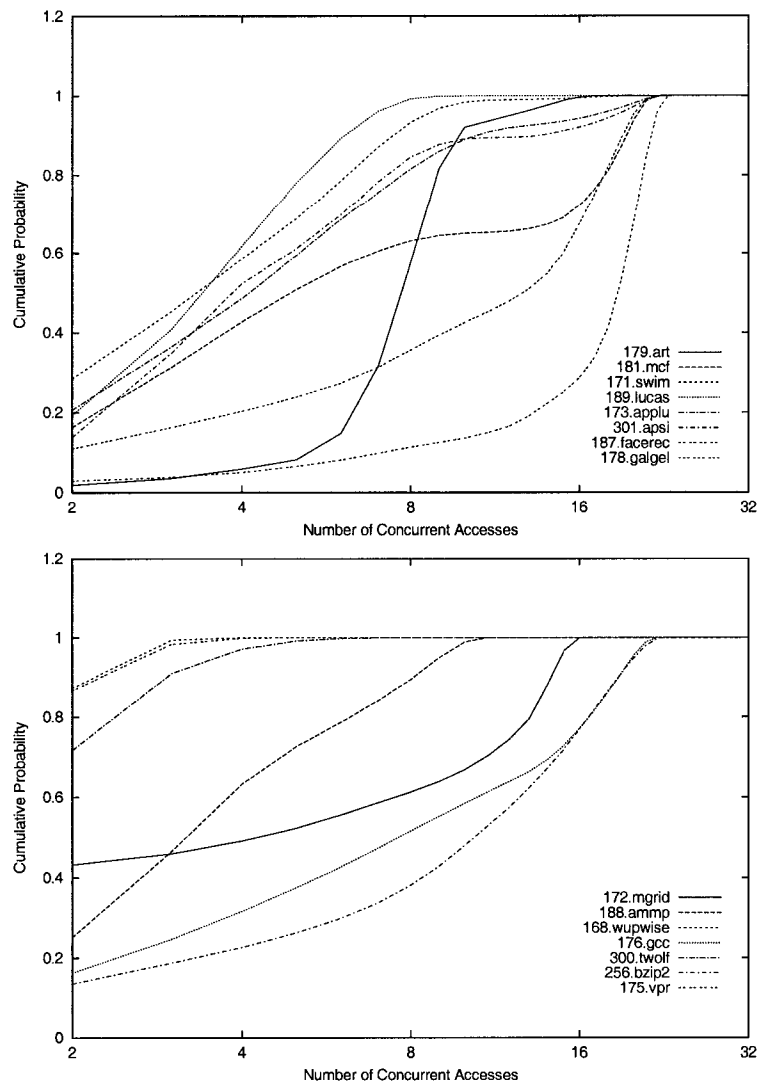


Figure 5.4: Distribution of the number of concurrent accesses.

bursty phase, when two or more outstanding memory references happen together. The top figure contains programs with the fraction of bursty phase higher than 40%; and the bottom one contains programs with the lower bursty phase fraction. Most programs have high burstiness in the bursty phase. In general, programs with a higher fraction of bursty phase tend to have higher probability on large number of concurrent accesses. For all the programs presented in the top figure, more than 40% of bursty references are grouped with

at least three other references. Even for some programs with a small bursty phase fraction, the burstiness inside the bursty phase is still high. For example, program *bzip2* only spends 6% of its execution time in the bursty phase, however, more than 60% of concurrent accesses are clustered as groups with at least eight references.

These indicate that normally multiple memory accesses happen in very short intervals. There are two reasons for high burstiness. First, when a miss happens, it is highly possible that the current working set does not reside in the cache, thus more misses are expected to happen in the near future. Second, wide-issue processors execute multiple instructions per cycle, thus make cache misses be clustered.

The clustering of memory accesses makes the load indicator scheme effective in capturing power saving opportunities while still maintaining the performance. Even when the processor issue rate reduces, the next memory access can still be found before previous memory accesses finish. In addition, the clustering of memory accesses also makes the processor issue rate switch not too frequently. We will discuss the power mode switch frequency in more details in Section 5.7.5.

5.4 Experimental Environment

We use *sim-alpha* as the simulator, which has been validated against a 466 MHz Alpha 21264 processor [26]. The Alpha 21264 processor is a supercalar processor that can issue up to four integer instructions and two floating-point instructions every cycle. Its pipeline contains seven basic stages. The on-chip instruction cache is 64 KB, 2-way set-associative with block size of 64 bytes, and contains a line predictor and a way predictor. The on-chip

| | |
|----------------------|--|
| Processor speed | 466 MHz/1 GHz/2 GHz |
| Processor issue rate | 6-way |
| Functional units | 4 int ALU, 4 int Mult 1 fp ALU, 1 fp Mult |
| Integer queue | 20 entries |
| Floating-point queue | 15 entries |
| Reorder buffer | 80 entries |
| Load queue size | 32 |
| Store queue size | 32 |
| Register file | 80-entry integer 72-entry floating point |
| Instruction cache | 2-way 64 KB, 64 B block 1/1/2 cycle hit latency |
| Data cache | 2-way 64 KB, 64 B block 3/3/4 cycle hit latency |
| L2 cache | 1-way 2 MB, 64 B block 7/12/14 cycle hit latency |
| DRAM latency | 2(RAS), 4(CAS), 2(precharge) 2(controller) bus cycles |
| Bus bandwidth | 1.2/1.6/2.1 GB/s |

Table 5.2: Experimental parameters.

data cache is 64 KB, 2-way set-associative with 64-byte blocks, and is backed up with an 8-entry victim buffer [45, 24]. The off-chip L2 cache is 2 MB direct-mapped with 64-byte blocks. The memory bus is 128-bit wide and operates at 75 MHz speed [26]. Considering the current processor speed, we also scale the processor speed to 1 GHz and 2 GHz in our experiments. The cache access latency, DRAM access latency, and bus bandwidth are scaled accordingly. The scaled cache access latency is estimated by the CACTI 3.0 model [63]. We assume that the process technique for caches is also scaled down as the processor speed increases. We also assume that the DRAM access latency and bus bandwidth improve by 30% when the processor speed increases from 466 MHz to 1GHz, and from 1 GHz to 2 GHz. Table 5.2 presents the key parameters in the experiment.

We modify the simulator to model our schemes and the pipeline balancing technique. The processor issue rate is dynamically adjusted when the conditions defined by the schemes are satisfied. The modified version also collects the statistics of program's execution behavior under each power-aware design for calculating the power consumption.

SPEC CPU2000 benchmark programs are used as the workloads [66]. We use the pre-compiled Alpha version of SPEC2000 binaries [77]. The reference input data files are used in the experiments. The first four billion instructions are fast-forwarded. Then, the detailed statistics are collected on the next one billion instructions.

5.4.1 Power Savings

Reducing the processor issue rate can reduce the power consumed by the issue logic and execution units [6]. These are two major power consumers in the processor. Among the 72-Watt power consumed on average by the 600 MHz Alpha 21264 processor, 18% is consumed by the instruction issue units, 10% is consumed by the integer execution units, and another 10% is consumed by the floating-point execution units [32]. We use these data in calculating power savings.

We assume that as the processor speed increases, the percentage of power consumed by the issue logic and the execution units does not change. This assumption is conservative and does not favor our schemes. As the processor speed increases, the percentage of power consumed by the issue logic and execution units tends to increase. For example, the issue units and execution units including the driving clocks consume 46% and 22% of total processor power for the projected Alpha 21464 processor [78]. As estimated in [6], the 46% of power consumption includes the power consumed by register files and register mapping,

thus the issue logic consumes about 23% of total power.

The Alpha 21264's pipeline has a cluster structure. The integer registers and functional units are organized as two clusters. The floating-point registers and functional units form another cluster [45, 24]. We assume that reducing the processor issue rate only reduces the power consumed by the issue logic and the integer functional units. This is also a conservative assumption. When the issue width is reduced by half, one of the integer cluster is gated.

There is an argument that reducing processor issue rate cannot bring much energy saving when aggressive clock gating techniques are applied to each component on a cycle-by-cycle basis. However, the increased design and verification complexity, clock skew, and inductive noise make implementing such aggressive clock gating challenging [69, 13]. Thus, the best granularity at which clock gating is applied is a trade-off between power saving and design complexity [69]. In addition, gating at a coarser grain may save more power since an entire resource or block can be powered off [13, 6]. Even if aggressive clock gating is applied, deactivating resources by architecture adaptation may save the remaining power after clock gating [61].

As discussed in [6], since reducing the issue rate does not change the number of instructions executed by the execution units, it can only reduce the power consumed by the clocks driving execution units. The clock network consumes 32% of processor power. Assuming the percentage of power consumed by clocks is a constant for each component, reducing the issue rate by half can reduce the power consumed by the clocks driving the integer execution units by half. It equals the saving on total processor power consumption by $10\% \times 32\% \times 1/2 = 1.6\%$.

| | mcf | art | swim | lucas | applu | ampp | facerec |
|---------------------------------------|------|------|------|-------|-------|------|---------|
| Time in low power mode (%) | 91.9 | 73.8 | 88.1 | 79.9 | 70.2 | 47.8 | 30.8 |
| Power reduction on issue logic (%) | 32.2 | 25.8 | 30.8 | 28.0 | 24.6 | 16.7 | 10.8 |
| Power reduction on execution unit (%) | 14.7 | 11.8 | 14.1 | 12.8 | 11.2 | 7.6 | 4.9 |

Table 5.3: Percentage of execution time in low power mode, and percentage of power reduction on the issue logic and execution units by the load indicator scheme under the system with 1 GHz processor.

It is normally unknown how many instructions can be issued in a given cycle until all the choices are exhausted. Thus, reducing the issue rate will reduce the amount of issue attempts if the program execution time is not extended. Therefore, reducing the issue rate can save power on both issue arbiters and clocks for the issue logic. For the issue logic, the arbiters including the driving clocks consume 70% of the issue queue power. Thus, reducing the issue rate by half can reduce the total processor power consumption by $18\% \times 70\% \times 1/2 = 6.3\%$. The total processor power reduction by halving the issue rate is $6.3\% + 1.6\% = 7.9\%$. This estimation method is the same as that used in [6].

5.5 Effectiveness of Load Indicator Scheme

In this section, we will discuss the effectiveness of the load indicator scheme on reducing processor power consumption for memory-intensive applications. The results on a wider range of applications will be presented in Section 5.7. All results are for the system with 1 GHz processor unless mentioned specifically.

From the SPEC2000 benchmark suite, we select seven memory-intensive applications, whose memory stall portions are higher than 20% under our experimental setup³. Table 5.3

³The memory stall portion of a program is defined as the percentage of time difference between running the program under a real system and under a system with an infinitely large L2 cache.

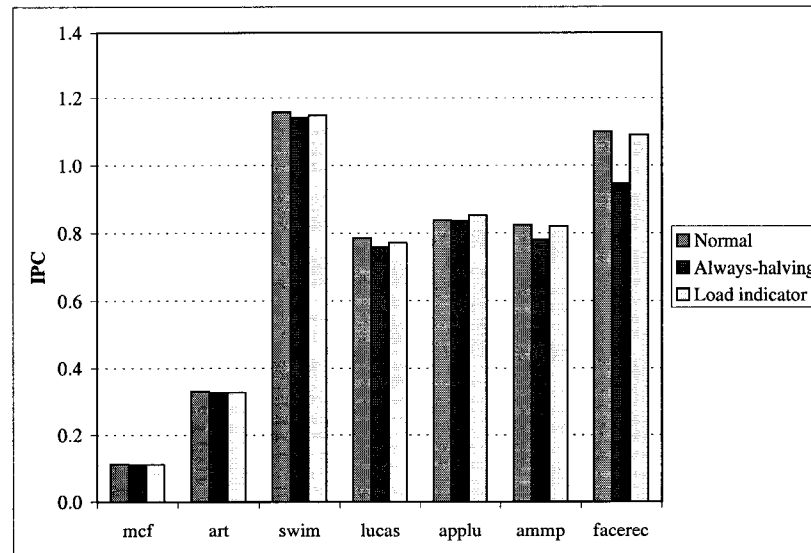


Figure 5.5: IPC values under the system with 1 GHz processors.

shows the percentage of power reduction on the issue logic and execution units for these seven applications by the load indicator scheme. The load indicator scheme puts the processor in the low power execution mode at 30.8% (for program *facerec*) to 91.9% (for program *mcf*) of the total execution time. On average, the processor spends 68.9% of time in the low power mode. This translates to 24.1% ($35\% \times 68.9\%$) and 11.0% ($16\% \times 68.9\%$) average reduction on the power consumed by the issue logic and execution units compared with the normal execution scheme, respectively. Thus, the load indicator scheme is very effective in capturing power saving opportunities for memory-intensive applications.

For general-purpose computer systems, the reduction of processor power consumption must not cause severe performance loss. Figure 5.5 shows the IPC values of the seven

memory-intensive applications under different schemes. Compared with the scheme that the processor always executes under the normal mode (*normal*), the scheme that always reduces the processor issue rate by half (*always-halving*) decreases IPC values by up to 13.9% (for program *facerec*). For program *facerec*, the always-halving issue rate scheme can save power by 7.9% but at the price of 13.9% performance loss. This results in a 7.0% increase on the total energy consumption. Thus, simply reducing the processor issue rate is not a solution to save power and energy. The processor issue rate must be reduced at a right time in order to retain performance and save energy.

The average performance loss by the load indicator scheme on the seven programs is only 0.5%. The maximum IPC decrease is only 1.7% (on program *lucas*). Actually, the performance of program *applu* is improved slightly because fewer mis-speculative instructions are issued and executed. As an example, program *facerec* is executed under the low power mode by 30.8% of its total execution time. If the performance loss of 13.9% by the always-halving scheme were evenly distributed over the whole execution period, the performance loss by reducing the issue rate at 30.8% of time would be 4.3%. However, the load indicator scheme only decreases performance by 0.9% for the program. This indicates that the load indicator can reduce the processor issue rate at the right time, and can save power consumption and retain performance simultaneously.

| Parameters | Values |
|--------------|---|
| Sample width | 64 cycles |
| <i>EC</i> | $(I_{IPC} < 1.1) \ \& \ (FP_{IPC} < 0.4)$ |
| <i>DC</i> | $(I_{IPC} > 1.2) \ \ (FP_{IPC} > 0.5)$ |

Table 5.4: Parameters for instruction indicator. *EC* and *DC* are the conditions for enabling and disabling the low power mode. I_{IPC} and FP_{IPC} are the number of integer instructions issued per cycle and the number of floating point instructions issued per cycle, respectively.

5.6 Combining Local and Look-ahead Optimizations

5.6.1 Motivation

In previous sections, we have already shown that the existence of outstanding cache load misses is a strong indicator reflecting future variants of program requirements on processor resources. Based on this information, the load indicator scheme dynamically adjusts the processor issue rate to save power consumption without degrading performance. However, this technique only has significant impacts on memory-intensive applications. The programs with negligible memory stall times seldom have chances to go to the low power mode. Thus, their power consumption and performance will not be affected.

Figure 5.6 shows the percentage of program execution time in low power mode under our scheme and the instruction indicator for fifteen SPEC2000 applications with a wide range of computation and memory demands. The programs are sorted by their memory stall portions, which range from 71% for *mcf* to 3% for *twolf*. The parameters used in the instruction indicator scheme are summarized in Table 5.4. The parameters have already been tuned for these applications under the system configuration.

For ten of the fifteen programs, the load indicator puts the processor in the low power mode more or at least comparably frequently, compared with the instruction indicator. For

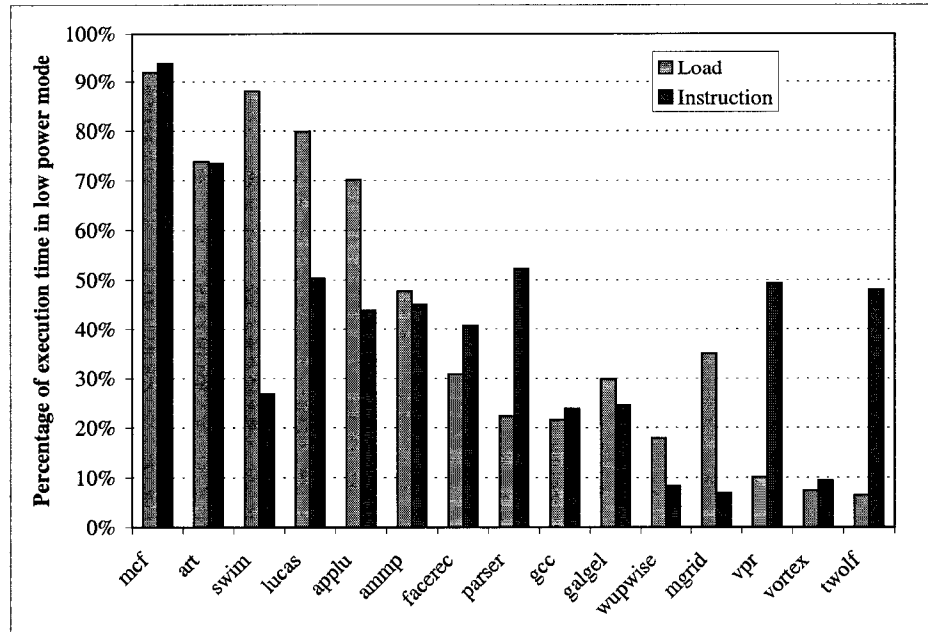


Figure 5.6: Percentage of total execution time in low power mode under our scheme (load) and instruction indicator (instruction).

programs losing more than half of their total execution time to memory stall (*mcf* and *art*), our scheme and the instruction indicator put the processor into the low power mode by almost the same amount of time. Because main memory accesses are so frequent for these two applications (more than five accesses per 100 instructions), the low IPC values are mainly caused by the frequent memory stall. Thus, the load indicator and the instruction indicator schemes capture the same effect from different perspectives. The look-ahead ability of our scheme does not gain much because the execution demanding is already low enough during most of the execution time. For programs with memory stall portions ranging

from 35% to 40% (*applu* to *swim*), our scheme captures much more opportunities to put the processor to the low power mode than the instruction indicator. As discussed in Section 5.2, this mainly comes from the prediction of future variants and the redistribution of current work. For programs with modest memory stall portion (8% for *mgrid* to 29% for *ammp*), the load indicator scheme captures more power-saving opportunities on four applications and less opportunities on three applications. For applications with low memory stall portion (about 3% for *vpr* to *twolf*), our scheme captures less opportunities to put processor to the low power mode since only a small percentage of time has outstanding load misses.

Although using memory access information to guide the processor issue rate adjustment is simple and timely, it fails to capture another important factor that also causes the variants of program resource requirements — the change of program’s inherent instruction-level parallelism (ILP). In contrast, the metric of instructions per cycle (IPC) reflects the overall effects of the change of ILP and the change of program behavior due to limited hardware resources [6]. However, in order to smooth out the spurs of IPC variants caused by temporary factors, the IPC values used to trigger processor issue rate switching must be monitored and averaged during a large enough time window. Thus, simply using IPC variants as the indicator may also lose opportunities to save power or retain performance due to the delayed captures and triggers.

In comparisons, the load indicator scheme is application- and system-independent; while the instruction indicator scheme needs system- and application-dependent parameters. The load indicator scheme captures the future program variants caused by memory accesses; while the instruction indicator scheme captures the current program variants caused by program’s inherent ILP changes and hardware constraints.

5.6.2 Load-instruction Indicator

With the knowledge of future variants, it is possible to redistribute processor work in order to generate more power saving opportunities. With the knowledge of current variants, it is possible to capture current power saving opportunities. Combining these two kinds of optimizations together, we propose another technique, called *load-instruction indicator*, that utilizes both the memory access information and the IPC values to guide the processor issue rate switches.

The scheme works as follows. When there are no outstanding load misses, the hardware monitors the issued IPC and adjusts the processor issue rate, as in the instruction indicator technique. When a cache load miss happens and the processor is in the normal execution mode, the processor will switch to the low power mode. During the service of cache load misses, the monitoring of issued IPC is suspended. It will be resumed when all the outstanding load misses finish. By combining local and look-ahead optimizations, the switching of processor issue rate can be done more effectively and more timely. The overhead of the scheme is negligible, since both the load indicator and the instruction indicator have trivial overhead.

There are several choices in suspending and resuming the IPC monitoring. One choice is to clear all counters when a cache load miss is detected. Another choice is to just freeze the values in the counters when cache load misses are served. We test both choices and find that they almost perform the same. In our experiments, all the counters are cleared when a load miss is detected.

```
if (a load miss happens) {
```

```
if (counter for load misses = 0)

    record current power mode as previous mode

counter for load misses + 1

if (in normal mode)

    transfer to low power mode

reset or stop counters for sampling cycle and issued instructions
}

if (a load miss is resolved) {

    counter for load misses - 1

    if (no outstanding load misses) {

        return to normal mode (OR previous mode)

        restart counters for sampling cycle and issued instructions

    }

}

if (no outstanding load misses) {

    if (reach sampling window boundary) {

        if (in normal mode AND issued instructions below deactivating thresholds)

            transfer to low power mode

        if (in low power mode AND issued instructions above activating thresholds)

            return to normal mode

        reset counters for sampling cycle and issued instructions

    }

    else {
```

```

    sampling cycle + 1
    issued instructions + number of instructions issued in this cycle
}
}

```

5.7 Comparisons between Different Schemes

In this section, we compare the three schemes we have discussed so far: the *load indicator*, *instruction indicator*, and *load-instruction indicator*.

5.7.1 Power Savings

Figure 5.7 shows the processor power reductions under three different schemes for the fifteen programs. We can see that for the seven memory intensive applications, the load indicator reduces the processor power consumption by 5.4%, compared with a 4.2% power saving by the instruction indicator scheme. In some cases, the power reduction difference is large. For example, the load indicator saves 7.0% of total power for program *swim*, while the instruction indicator only saves 2.1% of total power. As we have discussed in Section 5.2, for this program, 60.4% of sampling windows have IPC values higher than the thresholds under the normal execution, although almost no instructions are issued in 21.9% of sampling windows. Thus, under the instruction indicator scheme, less than 40% of time windows can be executed in the low power mode. In contrast, the load indicator scheme foresees the low demanding and idle periods and delays current work. Thus, the processor can stay at the low power mode much longer without causing severe performance loss. For three programs whose memory stall portions are only about 3% (*vpr* to *twolf*), the load indicator saves less

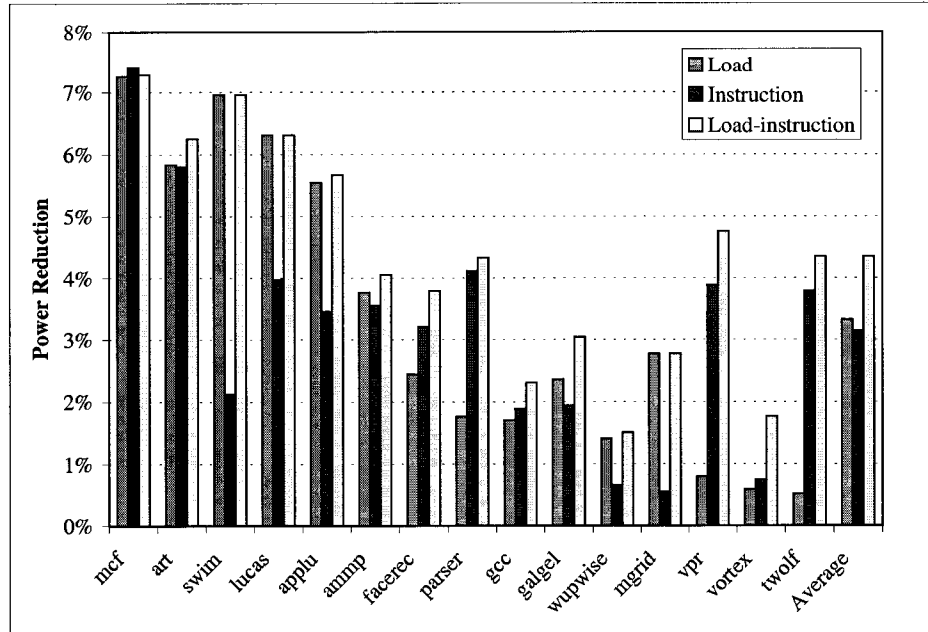


Figure 5.7: Percentage of power reduction.

power than the instruction indicator. For all the fifteen applications, the power reduction by the load indicator ranges from 0.5% (*twolf*) to 7.3% (*mcf*). The instruction indicator saves power by 0.6% (*mgrid*) to 7.4% (*mcf*). The average power reductions by the load indicator and the instruction indicator are 3.3% and 3.1%, respectively.

The load indicator performs better for memory-intensive applications, while the instruction indicator performs better for programs with small memory stall portions. The load-instruction technique combines the merits of both techniques and achieves higher power saving than either technique. The power reduction by the load-instruction technique ranges from 1.5% (*wupwise*) to 7.3% (*mcf*). The average power saving is 4.4%.

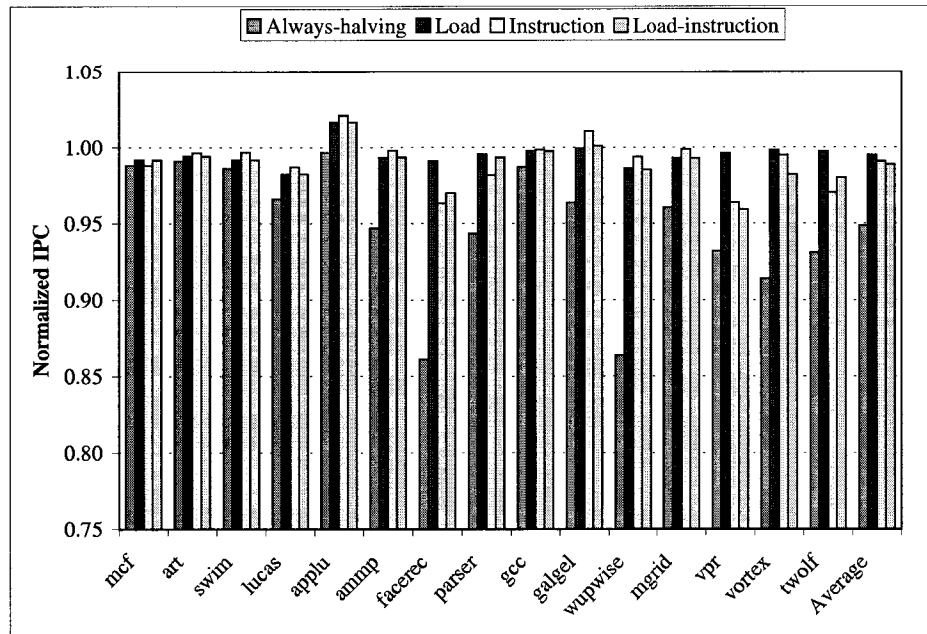


Figure 5.8: Normalized IPC values.

5.7.2 Performance Impact

Figure 5.8 presents the IPC values under the three schemes for the fifteen programs. The results under the always-halving scheme are also presented as a reference. All the values are normalized to the IPC results under the processor that is always in the normal execution mode.

The load indicator causes the performance loss by up to 1.7% (*lucas*). The average performance loss is 0.5%. The instruction indicator degrades performance by up to 3.7% (*facerec*). The average performance loss is 0.9%. The load-instruction technique causes a

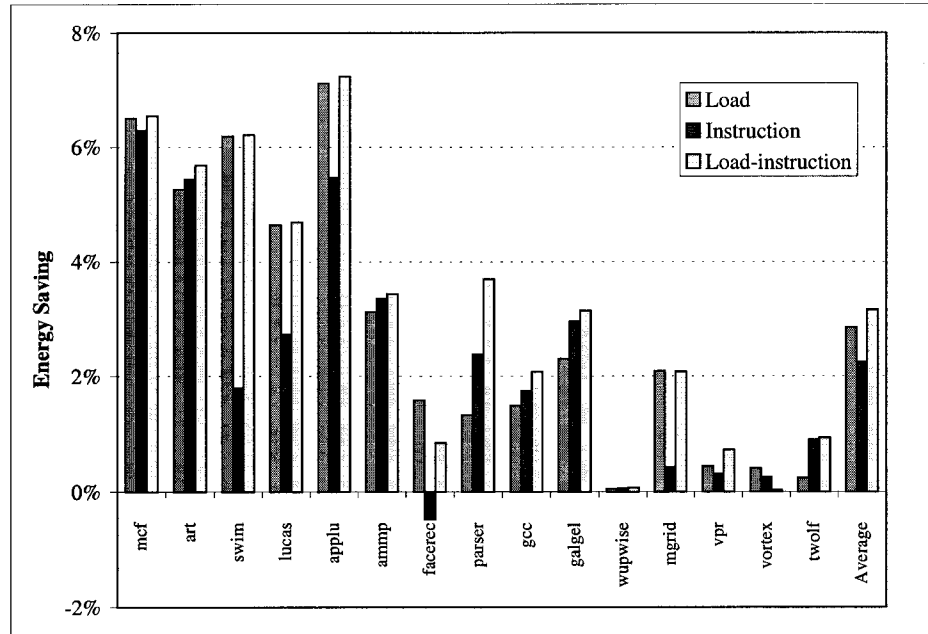


Figure 5.9: Percentage of energy reduction.

performance loss by up to 4.1% (*vpr*). The average performance loss is 1.1%. Among the fifteen applications, the load indicator scheme gains higher IPC values than the instruction indicator scheme on six programs. The load-instruction indicator scheme gains the lowest performance on only six programs although it puts the processor into the low power mode the most frequently.

5.7.3 Energy Reduction

The energy reduction is shown in Figure 5.9. The load indicator reduces the processor energy consumption by 0.1% (*wupwise*) to 7.1% (*applu*). The maximum energy reduction by the

instruction indicator is 6.3% (*mcf*). The load-instruction technique reduces the processor energy consumption by up to 7.2% (*applu*). The load indicator and the load-instruction indicator do not increase energy consumption for any program. However, the instruction indicator increases the energy consumption for program *facerec* slightly (0.5%). This is because the energy saved by the scheme cannot pay off the increase due to performance loss. Among the fifteen programs, the load-instruction technique achieves the lowest energy consumption for eleven programs. The average energy reduction by the load-instruction technique is 3.2%, compared with 2.9% by the load indicator and 2.2% by the instruction indicator.

For high performance systems, reducing power is mainly for temperature control. Under this aspect, the energy-delay product (EDP) and ED^2P are two suitable metrics [13]. The load indicator, instruction indicator, and load-instruction indicator can reduce the average EDP by 2.4%, 1.3%, and 2.0%; and can reduce the average ED^2P by 1.9%, 0.3%, and 1.0%, respectively.

5.7.4 Different Configurations

Next, we will compare the effectiveness of the load indicator, instruction indicator, and load-instruction indicator under different system configurations. We only present the power saving results here. Figure 5.10 shows the percentage of reduction on the whole processor power consumption. For clarity, we only present five representative applications in the figure, which consist applications with high, moderate, and low memory access portions. The average values are for all the fifteen applications. When the processor speed scales from 466 MHz to 1 GHz and 2 GHz, the average power reduction by the load indicator

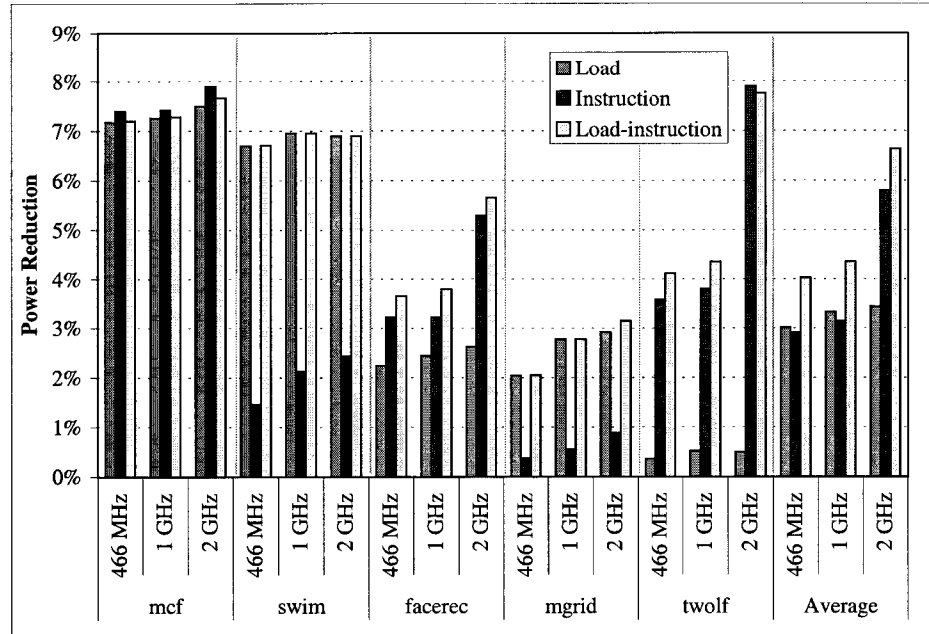


Figure 5.10: Power reduction under systems with processor speed scaling from 466 MHz to 1 GHz and 2 GHz.

increases from 5.1% to 5.4% and 5.8%. As the speed gap between processor and memory widens, the processor spends more time waiting for memory accesses to return. Thus, the effectiveness of the load indicator in reducing power and energy consumption increases. As the processor speed scales from 466 MHz to 1 GHz and 2 GHz, the average power reduction by the instruction indicator increases from 2.9% to 3.1% and 5.8%; and the average power reduction by the load-instruction indicator increases from 4.0% to 4.4% and 6.6%. The load-instruction indicator always gains the largest power saving under different system configurations.

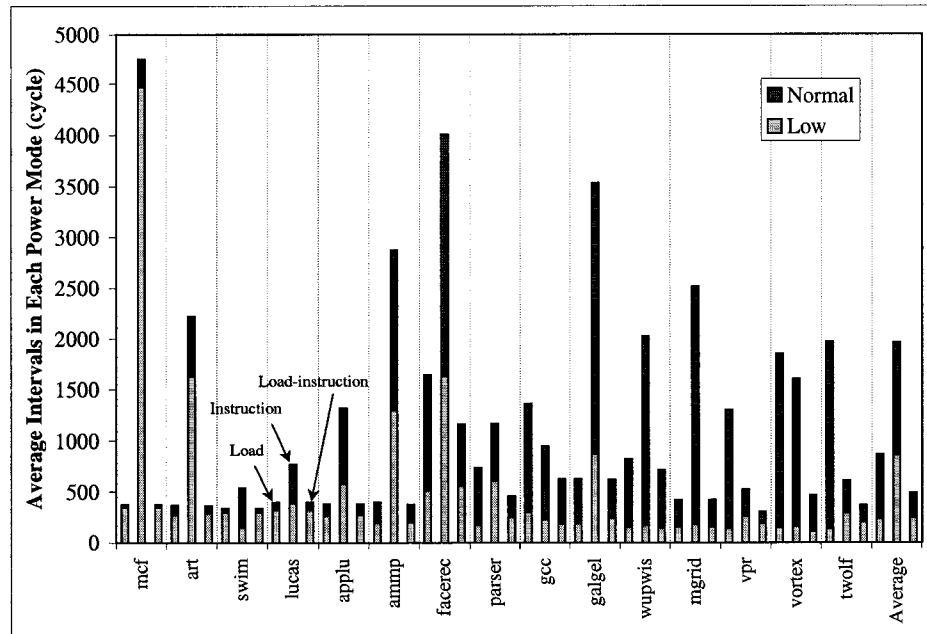


Figure 5.11: Average intervals in the low power mode and the normal execution mode under different schemes.

5.7.5 Average Intervals in Each Power Mode

Figure 5.11 compares the average intervals in the low power mode and the normal execution mode under the three schemes. On average, the programs stay at the low power mode for 236 cycles before returning to the normal execution mode under the load indicator scheme. The average interval in the normal execution mode is 631 cycles. Under the instruction indicator scheme, the average intervals in the low power mode and the normal execution mode are 859 cycles and 1101 cycles, respectively. The average low power interval of the load-instruction scheme is 260 cycles, and the average normal execution interval is 196

cycles.

In general, the load indicator scheme generates shorter intervals in each power mode than the instruction indicator scheme. This indicates that the load indicator scheme are more responsive than the instruction indicator scheme. On the other hand, the load indicator scheme causes the programs with low memory stall portions to stay at the normal execution mode longer than the instruction indicator scheme. This is because the load indicator cannot capture the changes of program behavior due to the program ILP variants. For most programs, the load-instruction indicator scheme causes the most frequent power mode switching. It can capture the variants of program behavior more timely than the other two schemes, and can cover both compute- and memory- intensive applications.

5.8 Load-register Indicator Scheme

In previous sections, we have already shown that the load indicator scheme is an effective look-ahead technique in capturing the power saving opportunities caused by main memory accesses. The load indicator can be further combined with the instruction indicator to save power for both compute- and memory-intensive applications. A different direction to extend the load indicator in order to cover a wider range of applications is to combine it with other early indicators that reflect the changes of program behavior. One of such indicators is the usage of hardware resources.

Each on-the-fly instruction occupies some hardware resources, such as physical registers, reorder buffer entries, and load/store queue entries. When there are only a small number of free entries left in those hardware resources, it is very likely that the processor will stall in

the near future. The cumulation of occupied entries may come from the lack of ILP among the on-the-fly instructions or from the lack of other hardware resources.

For the system used in our study, we find that the number of free physical registers can be used to identify the execution periods in which programs only require partial processor issue width. We propose a scheme that uses both the register usage information and the cache load miss information to trigger the issue rate switching. We call this scheme *load-register indicator*. The scheme reduces the processor issue rate when the number of free registers drops below a threshold or when there are outstanding load misses. The full issue rate is resumed when the number of free registers increases above another threshold and when there are no outstanding load misses. Considering that the shortage of functional units may also cause instructions holding registers for a long time, the scheme checks the availability of functional units when the number of free registers drops below the switching threshold. If any instruction fails to get the required functional unit in the previous cycle, the processor will stay at the normal execution mode.

```
if (a cache load miss happens)
    miss register value + 1;
    if (processor is in normal mode)
        processor transfers to the low power mode;

if (number of free registers < threshold_low AND no FU shortage)
    if (processor is in normal mode)
        processor transfers to the low power mode;
```


| Schemes | Enabling low power mode | Disabling low power mode |
|-------------------------|---|---|
| Load indicator | $LD > 0$ | $LD = 0$ |
| Load-register indicator | $(LD > 0) \mid (I_{reg} < TE_{IR})$ $\mid (F_{reg} < TE_{FR})$ | $(LD = 0) \ \& \ (I_{reg} > TD_{IR})$ $\ \& \ (F_{reg} > TD_{FR})$ |

Table 5.5: The conditions for enabling and disabling the low power mode under different schemes. LD is the number of outstanding load misses. I_{reg} and F_{reg} are the numbers of free integer and floating-point registers. TE_{XX} and TD_{XX} are thresholds for enabling and disabling the low power mode, respectively.

```

if (a cache load miss resolves)

    miss register value - 1;

    if (miss register value = 0 AND number of free registers > threshold_high

        AND processor is in low power mode)

        processor returns to the normal mode;

```

In contrast to the load indicator scheme, which needs no system-dependent parameters, the load-register scheme requires system-dependent threshold values to trigger the processor issue rate switching. However, the load-register scheme has effects on a wider range of programs, as shown later in the experimental results. The increase of hardware complexity of the load-register scheme is also trivial. Only the counters for the free registers and the comparators to the thresholds are needed.

The load-register scheme can capture the variants of program resource requirements caused by both the changes of ILP and the limited hardware resources. In addition, it can identify the program behavior changes more timely than the instruction indicator because the existence of cache load misses and the number of free registers can be monitored at finer time granularity than the IPC values.

Table 5.5 summarizes the conditions for enabling and disabling the low power mode under the load indicator and the load-register indicator schemes.

5.9 Effectiveness of Load-register Scheme

In this section, we will show that, the combination of cache load misses and the number of free physical registers can be used to identify power saving opportunities for both compute- and memory- intensive applications.

5.9.1 Power Savings

Figure 5.12 compares the processor power reduction under the load-register scheme with the load indicator and the load-instruction indicator schemes. For all the fifteen applications, the load-register indicator scheme saves more power than the load indicator scheme. Compared with the load-instruction indicator scheme, the load-register indicator scheme saves more or at least comparable power for thirteen programs. Only for two programs, *vpr* and *twolf*, the load-register scheme saves less power than the load-instruction indicator. The power reduction by the load-register scheme ranges from 2.4% (for *gcc*) to 7.8% (for *swim*). The average power reduction is 5.5%, which is 69% of the maximum power saving that can be achieved by halving the processor issue rate. In comparisons, the load indicator scheme and the load-instruction indicator scheme only achieve 42% and 56% of the maximum power saving, respectively. The power savings for ten programs by the load-register indicator scheme is higher than 5%. In contrast, only five programs get power saving higher than 5% under the load indicator scheme; and five programs have power saving higher than 5% under the load-instruction indicator scheme.

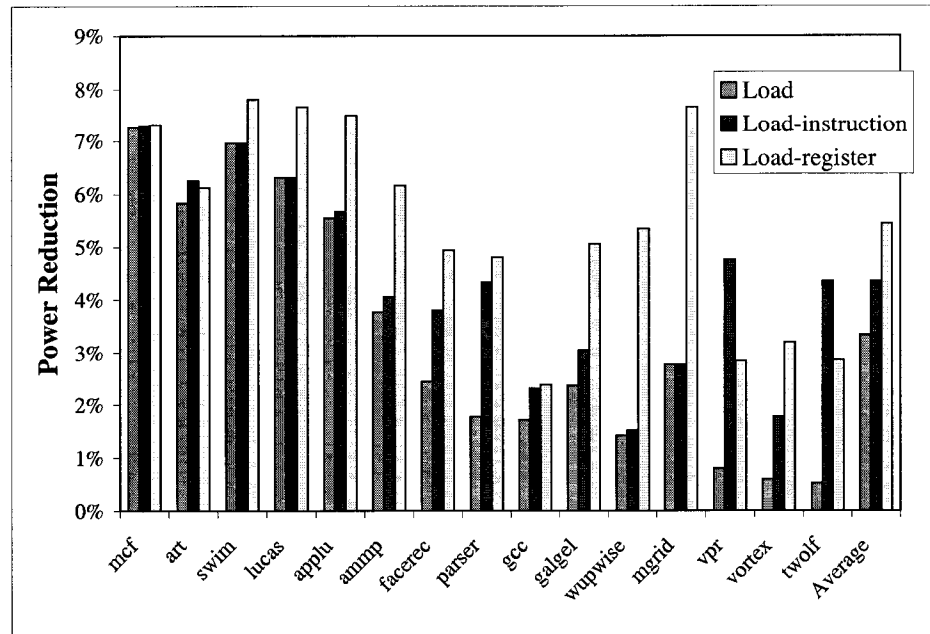


Figure 5.12: Percentage of power reduction under the load indicator, load-instruction indicator, and load-register indicator schemes on systems with 1 GHz processors.

Figure 5.13 further shows the decomposition of program execution time in the low power mode under the load-register indicator scheme for the fifteen SPEC2000 applications. The load and register portions correspond to the low power execution periods triggered only by the load indicator and the register indicator, respectively. The overlap portion corresponds to those caused by both triggers. In general, the existence of cache load misses triggers more transitions to the low power mode for applications with high memory stall portions, and the shortage of free registers plays more important roles for applications with low memory stall portions.

Table 5.6 summarizes the parameters used in the load-register scheme.

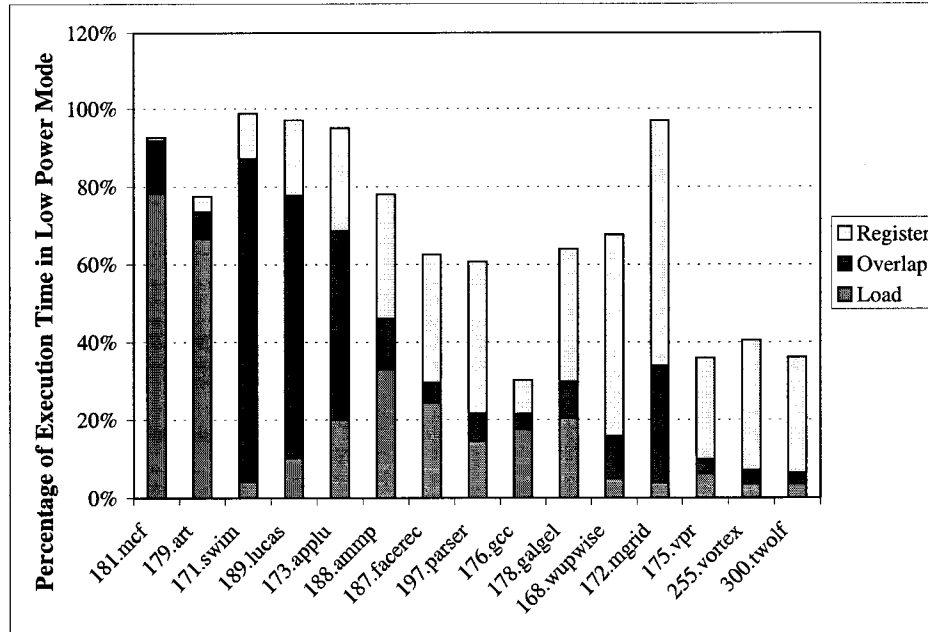


Figure 5.13: Percentage of total execution time in the low power mode under the load-register indicator scheme. The load and register portions correspond to the low power execution periods triggered only by the load indicator and the register indicator, respectively. The overlap portion corresponds to those caused by both triggers.

5.9.2 Energy Savings

The reduction of energy consumption is shown in Figure 5.14. The load-register technique reduces the processor energy consumption by up to 8.8% (*applu*). However, it increases the energy consumption for programs *wupwise* and *vortex* by 3.0% and 0.2%, respectively. This is because the energy saved by the scheme cannot pay off the increase due to performance loss. The average energy reduction by the load-register technique is 3.0%, compared with 2.9% by the load indicator and 3.2% by the load-instruction indicator.

| Conditions | Values |
|------------|---|
| Enabling | $(LD > 0) \mid (I_{reg} < 8) \mid (F_{reg} < 4)$ |
| Disabling | $(LD = 0) \ \& \ (I_{reg} \geq 12) \ \& \ (F_{reg} \geq 8)$ |

Table 5.6: The conditions for enabling and disabling the low power mode by the load-register indicator scheme. LD is the number of outstanding load misses. I_{reg} and F_{reg} are the numbers of free integer and floating-point registers, respectively.

In comparison, both load-instruction indicator and load-register indicator can effectively reduce power consumption for compute- and memory-intensive applications. Both need system-related parameters to guide the issue rate adjustment. The load-register scheme can make the adjustment at a finer grain than the load-instruction indicator. Our experimental results indicate that further combining the load indicator, the instruction indicator, and the register indicator can only bring diminishing power-saving return compared with either load-instruction or load-register schemes.

5.10 Related Work

Recently, there has been an increasing number of studies targeting reducing the power consumption of general-purpose processors. Pipeline gating reduces the processor energy consumption by preventing wrong-path instructions from entering the pipelines [51]. Authors in [29] propose power saving techniques to avoid unnecessary comparisons at the wake-up logic. Dynamically adjusting the issue queue size, the reorder buffer size, and the load/store queue size is an effective approach to reducing the power consumption of issue logic [17, 29, 56].

Authors in [30] propose a method to reduce processor energy consumption by switching the processor execution between out-of-order and in-order execution modes under the

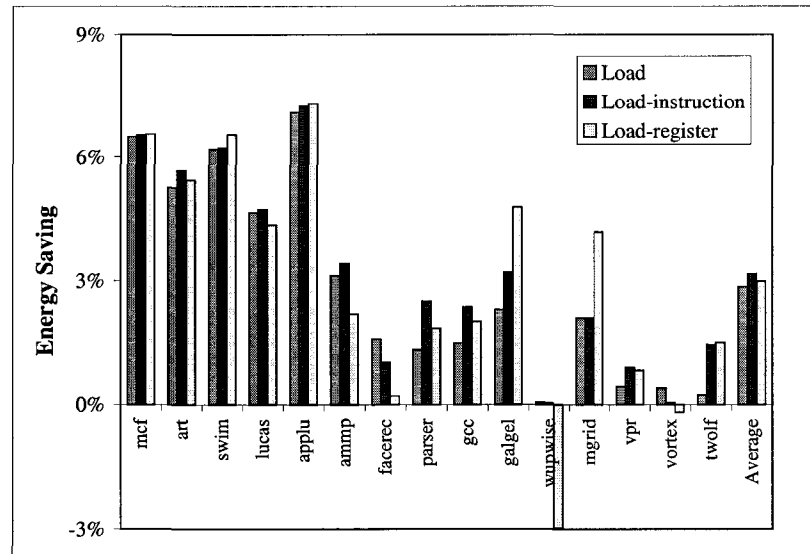


Figure 5.14: Percentage of energy reduction under the load indicator, load-instruction indicator, and load-register indicator schemes under systems with 1 GHz processors.

guidance of an external performance indicator. The dual speed pipeline processor saves power by executing non-critical instructions in slow components and retains performance by executing critical instructions in fast components [57, 62]. The Pentium 4 processor explores the StopClock structure to reduce the power consumption, where the clock signal to a bulk of processor logic is halted for a short time period [33]. The pipeline balancing technique dynamically adjusts the processor issue rate for each sampling window based on the monitored IPC [6]. Authors in [8] propose to use the current rate of instructions passing pipeline stages to throttle the processor front-end for power saving. Authors in [38, 61] target energy saving for multimedia applications by using dynamic voltage scaling and architectural adaptation at the granularity of a frame and a fixed period within the frame.

Buyuktosunoglu et. al. propose a combination of fetch gating and issue queue adaptation to reduce the power consumed by the front-end instruction delivery path [16]. Huang et. al. propose a positional adaptation technique that chooses configurations based on particular code sections [35].

The effectiveness of load indicator scheme in reducing processor power consumption depends on the memory system performance. Recently, researchers have proposed to utilize the idle intervals to perform precomputation-based prefetching [4, 7, 23, 49, 91]. Those are performance-oriented techniques, which perform a large number of speculative (maybe duplicate or useless) operations, thus may not be efficient from an energy consumption point of view. Different from those performance-oriented techniques, our work targets energy saving by utilizing the idle intervals.

5.11 Summary

Adjusting the processor issue rate based on the resource requirement of programs can effectively reduce the processor power consumption with a negligible performance loss. Our study has shown that the existence of cache load misses is a strong indicator reflecting the reduction on a near future resource requirement. For memory-intensive applications, simply applying this indicator can capture most of the program behavior variants timely. Compared with the pipeline balancing technique using the instruction indicator, the load indicator can save more power with comparable performance losses.

The load-instruction scheme adaptively utilizes both the load indicator and the instruction indicator used in the pipeline balancing technique. Compared with the load indicator,

this scheme can capture the current power saving opportunities caused by the program ILP variants. Compared with the pipeline balancing, it can identify future program variants and redistribute current work. As shown by our experimental results, the load-instruction scheme works well for both compute- and memory- intensive applications.

The number of free registers can also indicate the variants of program future resource requirements timely. The load-register indicator utilizes the information of both cache load misses and the register utilization, and is also effective for both compute- and memory-intensive applications.

The load indicator, the load-instruction indicator, and the load-register indicator schemes reduce the processor power consumed by the issue logic and execution units. They can be applied together with other techniques that reduce power consumed by other components such on-chip caches to further reduce the total processor power consumption.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

With the dramatic improvements on fabrication techniques and architecture designs, computer system performance has been increasing exponentially for several decades. A negative byproduct of this fast performance improvement, nevertheless, is the undesirable increase of power consumption. High power consumption shortens battery lifetime of portable systems and causes high cooling and package cost on performance-oriented systems. To countercheck this trend, researchers have proposed a number of techniques at various design levels. As a natural bridge between circuits and applications, architectural designs play an important role in reducing system power consumption. They can effectively utilize low-power circuit techniques by exploiting application characteristics.

A principle in low-power architecture designs is to avoid unnecessary operations whenever possible. For general-purpose systems, the system designs are optimized to achieve the best performance for a wide range of applications. Thus, it is common that a significant amount of hardware resources are underutilized for a specific application or during a certain execution period. However, those underutilized resources will consume the same amount

of power if no power-aware designs are applied. In this dissertation, we have discussed two memory-related low-power microarchitecture designs that reduce unnecessary accesses to caches and processor pipelines.

Set-associative caches are widely used in modern computer systems because they can reduce cache conflict misses and improve the memory system performance. However, the conventional implementations of set-associative caches are not energy-efficient by their nature. Our first work proposes an access mode prediction cache structure to address this issue. We find that the way-prediction technique only works well for applications with good locality, while the phased technique only works well for application with poor locality. The cache energy consumption can be minimized when a cache hit is handled by the way-prediction technique and a cache miss is handled by the phased technique. The AMP cache dynamically applies these two techniques for the next cache reference based on the cache hit/miss prediction. We also optimize a way-prediction scheme, multicolumn-based way-prediction, for energy reduction. Our experimental results indicate that the AMP cache with multicolumn way-prediction is a nearly optimal cache structure in terms of both energy consumption and performance, compared with existing solutions.

Our second work focuses on reducing power consumption on the processor pipeline. Dynamically adjusting the processor issue rate is an effective technique to reduce the power consumed by the issue logic and execution units. We find that a simple scheme based on the existence of main memory load accesses can predict the future degrade on computation demands and capture the power saving opportunities. Our study indicates that this load indicator scheme can effectively reduce the power consumption for memory-intensive applications with negligible performance impacts. The load indicator scheme can be further

combined with other indicators to save power for both compute- and memory-intensive applications. The load-instruction indicator scheme utilizes both the existence of memory accesses and the number of instruction issued in each time window. The load-register scheme uses the memory access information and the number of unoccupied registers to guide the issue rate switching. Different from the load indicator scheme, the later two schemes need some system-dependent parameters to make the adaptation. However, they can cover a wider range of applications.

The techniques we have proposed in this dissertation can be applied together with each other or with power-saving techniques that target other components to further reduce the power consumption of the whole system.

6.2 Future Work

This dissertation has been focused on hardware approaches. Based on our experience on low power designs, further significant improvements are likely to come from hardware/software cooperations at different levels, which have been studied but mainly for performance objectives. At the system level, for example, operating systems may actively turn on/off computing nodes, hard disks, or a portion of processors in multiprocessors, as long as future performance demand is predictable. At a lower level, profiling tools may analyze phases of runtime power demands for a particular application, and then the processor power can be adjusted to the phases with appropriate hardware support. Currently, it is hard to predict the performance loss of a program after applying the architecture adaptation. This fact is undesirable especially for applications requiring guaranteed or predictable response time.

we want to first study how to control the performance loss with software support. With the flexibility and long-range visibility, software may help to address this issue by providing some feedback information to the adaptation mechanism. We also want to study, when software support is present, how to dynamically choose the best indicator or the best combination of indicators, and how to reduce the complexity and overhead of the implementation.

SMT (simultaneous multithreading) [71], a recent advance in computer architecture, also provides new platforms for low power research. On SMT processors, multiple threads share some common hardware resources such as instruction fetching units, physical registers, instruction issue queue, scheduling logic, caches, and main memory units. How to efficiently and effectively utilize those resources among multiple threads is an important research topic. Some hardware resources, particularly caches, may be either shared or partitioned for the threads, each with tradeoffs on performance for different application scenarios. If all or most threads are memory-intensive applications, cache conflict may be a severe issue and thus partitioned cache is preferred to avoid significant performance degrade. On the other hand, when the threads have unbalanced working set sizes, shared caches are favored because the caches will be better utilized than partitioned caches. While performance-oriented studies have been conducted, energy efficiency has received much less attentions. From an energy consumption point of view, building caches using small blocks is much more energy-efficient, because a smaller fraction of chip area will be charged. Thus, partitioning data caches, especially large L2 or even L3 caches, is favored by energy saving. We are investigating approaches that partition the caches into a number of small blocks and dynamically assign those blocks to threads according to their current data access patterns. Our future research attempts to answer the following questions: (1) how to implement cache

partitioning efficiently, without the increase of cache access time; (2) how to dynamically assign blocks to multiple threads to maximize performance; and (3) how to improve cache energy efficiency by dynamically turning on/off cache blocks.

Bibliography

- [1] A. AGARWAL AND S. PUDAR. Column-associative caches: a technique for reducing the miss rate for direct-mapped caches. In *Proceedings of the 20th International Symposium on Computer Architecture*, San Diego, CA, 1993.
- [2] A. AGARWAL, M. HOROWITZ, AND J. HENNESSY. Cache performance of operating systems and multiprogramming workloads. *ACM Transactions on Computer Systems*, 6(4):393–431, November 1988.
- [3] D. H. ALBONESI. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-32)*, pages 248–259, 1999.
- [4] M. ANNAVARAM, J. M. PATEL, AND E. S. DAVIDSON. Data prefetching by dependence graph precomputation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 52–61, 2001.
- [5] Avant! Corporation. *Avant! Star-Hspice Data Manual*. <http://www.avantcorp.com>.
- [6] R. I. BAHAR AND S. MANNE. Power and energy reduction via pipeline balancing. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 218–229, 2001.
- [7] R. BALASUBRAMONIAN, S. DWARKADAS, AND D. H. ALBONESI. Dynamically allocating processor resources between nearby and distant ILP. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 26–37, 2001.
- [8] A. BANIASADI AND A. MOSHOVOS. Instruction flow-based front-end throttling for power-aware high-performance processors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, pages 16–21, 2001.
- [9] L. BENINI, A. BOGLIOLO, AND G. D. MICHELI. A survey of design techniques for system-level dynamic power management. 8(3), June 2000.
- [10] J. E. BENNETT AND M. J. FLYNN. Prediction caches for superscalar processors. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-30)*, pages 81–91, 1997.
- [11] W. J. BOWHILL AND ET. AL. A 300 MHz 64 b quad-issue CMOS RISC microprocessor. In *Proceedings of IEEE International Solid-State Circuits Conference*, 1995.

- [12] D. BROOKS, V. TIWARI, AND M. MARTONOSI. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, 2000.
- [13] D. M. BROOKS, P. BOSE, S. E. SCHUSTER, H. JACOBSON, P. N. KUDVA, A. BUYUKTOSUNOĞLU, J.-D. WELLMAN, V. ZYUBAN, M. GUPTA, AND P. W. COOK. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November/December 2000.
- [14] D. C. BURGER AND T. M. AUSTIN. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, 1997.
- [15] J. A. BUTTS AND G. S. SOHI. A static power model for architects. In *Proceedings of the 33rd annual IEEE/ACM international symposium on Microarchitecture*, pages 191–201, 2000.
- [16] A. BUYUKTOSUNOĞLU, T. KARKHANIS, D. ALBONESI, AND P. BOSE. Energy efficient co-adaptive instruction fetch and issue. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.
- [17] A. BUYUKTOSUNOĞLU, S. SCHUSTER, D. BROOKS, P. BOSE, P. COOK, AND D. ALBONESI. An adaptive issue queue for reduced power at high performance. In *Workshop on Power-Aware Computer Systems, in conjunction with ASPLOS-IX*, 2000.
- [18] B. CALDER, D. GRUNWALD, AND J. EMER. Predictive sequential associative cache. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA '96)*, pages 244–253, 1996.
- [19] B. CALDER AND D. GRUNWALD. Next cache line and set prediction. In *Proc. of the 22nd Annual International Symposium on Computer Architecture*, pages 287–296, 1995.
- [20] R. CANAL, A. GONZÁLEZ, AND J. E. SMITH. Very low power pipelines using significance compression. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*, pages 181–190, 2000.
- [21] D. CARMEAN. Invited talk: Power, a perspective from the desktop. In *Kool Chips Workshop, in conjunction with MICRO33*, 2000.
- [22] J. H. CHANG, H. CHAO, AND K. SO. Cache design of a sub-micron CMOS system/370. In *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pages 208–213, 1987.
- [23] J. D. COLLINS, H. WANG, D. M. TULLSEN, C. HUGHES, Y.-F. LEE, D. LAVERY, AND J. P. SHEN. Speculative precomputation: Long-range prefetching of delinquent loads. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 14–25, 2001.
- [24] Compaq Computer Corporation. *21264/EV68CB and 21264/EV68DC Hardware Reference Manual*, 2001.

- [25] V. CUPPU, B. JACOB, B. DAVIS, AND T. MUDGE. A performance comparison of contemporary DRAM architectures. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, pages 222–233, 1999.
- [26] R. DESIKAN, D. BURGER, AND S. W. KECKLER. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, 2001.
- [27] X. DU, X. ZHANG, AND Z. ZHU. Memory hierarchy considerations for cost-effective cluster computing. *IEEE Transactions on Computers*, 49(9):915–933, Sept 2000.
- [28] W. FEN LIN, S. K. REINHARDT, AND D. BURGER. Reducing DRAM latencies with an integrated memory hierarchy design. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 301–312, 2001.
- [29] D. FOLEGNANI AND A. GONZÁLEZ. Energy-effective issue logic. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 230–239, 2001.
- [30] S. GHIASI, J. CASMIRA, AND D. GRUNWALD. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity-Effective Design, in conjunction with the 27th Annual International Symposium on Computer Architecture*, 2000.
- [31] R. GONZALEZ AND M. HOROWITZ. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.
- [32] M. K. GOWAN, L. L. BIRO, AND D. B. JACKSON. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of the 1998 Conference on Design Automation*, pages 726–731, 1998.
- [33] S. H. GUNTHER, F. BINNS, D. M. CARMEAN, AND J. C. HALL. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, Q1, 2001.
- [34] A. HASEGAWA, I. KAWASAKI, K. YAMADA, S. YOSHIOKA, S. KAWASAKI, AND P. BISWAS. SH3 — high code density, low-power. *IEEE Micro*, 15(6):11–19, December 1995.
- [35] M. HUANG, J. RENAU, AND J. TORRELLAS. Positional adaptation of processors: application to energy reduction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.
- [36] M. HUANG, J. RENAU, S.-M. YOO, AND J. TORRELLAS. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*, pages 202–213, 2000.
- [37] M. HUANG, J. RENAU, S.-M. YOO, AND J. TORRELLAS. L1 data cache decomposition for energy efficiency. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design*, 2001.

- [38] C. J. HUGHES, J. SRINIVASAN, AND S. V. ADVE. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pages 250–261, 2001.
- [39] K. INOUE, T. ISHIHARA, AND K. MURAKAMI. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.
- [40] Intel Corporation. *ACPI specification minimizes power usage by PCs*. <http://www.intel.com>.
- [41] M. J. IRWIN AND V. NARAYANAN. Low power design: From soup to nuts. Tutorial in conjunction with ISCA 2000, 2000.
- [42] M. B. KAMBLE AND K. GHOSE. Analytical energy dissipation models for low-power caches. In *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, pages 143–148, 1997.
- [43] S. KAXIRAS, Z. HU, AND M. MARTONOSI. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.
- [44] R. E. KESSLER, R. JOOSS, A. LEBECK, AND M. D. HILL. Inexpensive implementations of set-associativity. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, pages 131–139, 1989.
- [45] R. E. KESSLER. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.
- [46] H. KIM, V. NARAYANAN, M. KANDEMIR, AND M. IRWIN. Multiple access caches: Energy implications. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00)*, 2000.
- [47] J. KIN, M. GUPTA, AND W. H. MANGIONE-SMITH. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 184–193, 1997.
- [48] A. LEBECK, X. FAN, H. ZENG, AND C. ELLIS. Power aware page allocation. In *Proceedings of the 9th international conference on Architectural support for programming languages and operating systems*, pages 105–116, 2000.
- [49] C.-K. LUK. Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processors. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 40–51, 2001.
- [50] A. MA, M. ZHANG, AND K. ASANOVIĆ. Way memoization to reduce fetch energy in instruction caches. In *Workshop on Complexity-Effective Design, in conjunction with the 28th International Symposium on Computer Architecture*, 2001.
- [51] S. MANNE, A. KLAUSER, AND D. GRUNWALD. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–141, 1998.

- [52] M. MARTONOSI, D. BROOKS, AND P. BOSE. Modeling and analyzing cpu power and performance: Metrics, methods, and abstractions. Tutorial in conjunction with HPCA 2001, 2001.
- [53] S. MCFARLING. Combining branch predictors. Technical Report TN-36, Digital Equipment Corporation, Western Research Lab, June 1993.
- [54] J. MONTANARO AND ET. AL. A 160-MHz 32-b 0.5-W CMOS RISC microprocessor. *Digital Technical Journal*, 9(1):49–62, 1997.
- [55] S.-T. PAN, K. SO, AND J. T. RAHMEH. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 76–84, 1992.
- [56] D. PONOMAREV, G. KUCUK, AND K. GHOSE. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pages 90–101, 2001.
- [57] R. PYREDDY AND G. TYSON. Evaluating design tradeoffs in dual speed pipelines. In *Workshop on Complexity-Effective Design, in conjunction with the 28th International Symposium on Computer Architecture*, 2001.
- [58] G. REINMAN AND N. JOUPPI. An integrated cache timing and power model. Technical report, COMPAQ Western Research Lab, 1999.
- [59] S. RIXNER, W. J. DALLY, U. J. KAPASI, P. MATTSON, AND J. D. OWENS. Memory access scheduling. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 128–138, 2000.
- [60] M. ROSENBLUM, E. BUGNION, S. DEVINE, AND S. A. HERROD. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, January 1997.
- [61] R. SASANKA, C. J. HUGHES, AND S. V. ADVE. Joint local and global hardware adaptations for energy. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002.
- [62] J. S. SENG, E. S. TUNE, AND D. M. TULLSEN. Reducing power with dynamic critical path information. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pages 114–123, 2001.
- [63] P. SHIVAKUMAR AND N. JOUPPI. An integrated cache timing, power, and area model. Technical report, COMPAQ Western Research Lab, 2001.
- [64] D. SINGH AND V. TIWARI. Power challenges in the internet world. In *Cool Chips Tutorial – An industrial perspective on low power processor design. In conjunction with the 32nd Annual International Symposium on Microarchitecture*, pages 8–15, 1999.

- [65] J. E. SMITH. A study of branch prediction strategies. In *Proceedings of 8th Annual International Symposium on Computer Architecture*, pages 135–148, 1981.
- [66] Standard Performance Evaluation Corporation. *SPEC CPU2000*. <http://www.spec.org>.
- [67] C.-L. SU AND A. M. DESPAIN. Cache design trade-offs for power and performance optimization: a case study. In *Proceedings 1995 International Symposium on Low Power Design*, pages 63–68, 1995.
- [68] Synopsys Inc. *PowerMill Data Sheet*. <http://www.synopsys.com>.
- [69] V. TIWARI, D. SINGH, S. RAJGOPAL, G. MEHTA, R. PATEL, AND F. BAEZ. Reducing power in high-performance microprocessors. In *Proceedings of the 1998 Conference on Design Automation*, pages 732–737, 1998.
- [70] Tom’s Hardware Guide. *Comparison of 21 Power Supplies*. <http://www.tomshardware.com/howto>.
- [71] D. TULLSEN, S. EGGERS, AND H. LEVY. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 392–403, 1995.
- [72] G. TYSON, M. FARRENS, J. MATTHEWS, AND A. R. PLESZKUN. A modified approach to data cache management. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 93–103, 1995.
- [73] O. UNSAL, I. KOREN, C. KRISHNA, AND C. MORITZ. Cool-fetch: Compiler-enabled power-aware fetch throttling. *ACM Computer Architecture Letters*, 1:100–103, 2002.
- [74] N. VIJAYKRISHNAN, M. KANDEMIR, M. J. IRWIN, H. S. KIM, AND W. YE. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106, 2000.
- [75] L. VILLA, M. ZHANG, AND K. ASANOVIĆ. Dynamic zero compression for cache energy reduction. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*, pages 214–220, 2000.
- [76] D. W. WALL. Limits of instruction-level parallelism. In *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1991.
- [77] C. WEAVER. <http://www.simplescalar.org/spec2000.html>. SPEC2000 binaries.
- [78] K. WILCOX AND S. MANNE. Alpha processors: A history of power issues and a look to the future. In *Cool Chips Tutorial – An industrial perspective on low power processor design. In conjunction with the 32nd Annual International Symposium on Microarchitecture*, pages 16–37, 1999.

- [79] S. J. E. WILTON AND N. P. JOUPPI. An enhanced access and cycle time model for on-chip caches. Technical Report WRL Research Report 93/5, DEC Western Research Laboratory, 1994.
- [80] J. YANG, Y. ZHANG, AND R. GUPTA. Frequent value compression in data caches. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*, pages 258–265, 2000.
- [81] T.-Y. YEH AND Y. N. PATT. Alternative implementations of two-level adaptive branch prediction. In *The 19th Annual International Symposium on Computer Architecture (ISCA)*, pages 124–134, 1992.
- [82] A. YOAZ, M. EREZ, R. RONEN, AND S. JOURDAN. Speculation techniques for improving load related instruction scheduling. In *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA '99)*, pages 42–53, 1999.
- [83] C. ZHANG, X. ZHANG, AND Y. YAN. Two fast and high-associativity cache schemes. *IEEE Micro*, 17(5):40–49, September/October 1997.
- [84] X. ZHANG, Z. ZHU, AND X. DU. Analysis of commercial workload on SMP multiprocessors. In *Performance'99 (extended abstract)*, 1999.
- [85] Y. ZHANG AND J. YANG. Low cost instruction cache designs for tag comparison elimination. In *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design*, 2003.
- [86] Z. ZHANG, Z. ZHU, AND X. ZHANG. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, pages 32–41, 2000.
- [87] Z. ZHANG, Z. ZHU, AND X. ZHANG. Breaking address mapping symmetry at multi-levels of memory heirarchy to reduce dram row-buffer conflicts. *Journal of Instruction-Level Parallelism*, 3, October 2001.
- [88] Z. ZHANG, Z. ZHU, AND X. ZHANG. Cached DRAM for ILP processor memory access latency reduction. *IEEE Micro*, 21(4):22–32, July/August 2001.
- [89] Z. ZHU AND X. ZHANG. Access-mode predictions for low-power cache design. *IEEE Micro*, 22(2):58–71, March/April 2002.
- [90] Z. ZHU, Z. ZHANG, AND X. ZHANG. Fine-grain priority scheduling on multi-channel memory systems. In *Proceedings the Eighth International Symposium on High-Performance Computer Architecture*, pages 107–116, 2002.
- [91] C. ZILLES AND G. SOHI. Execution-based prediction using speculative slices. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 2–13, 2001.
- [92] V. V. ZYUBAN. *Inherently low-power high-performance superscalar architectures*. PhD thesis, University of Notre Dame, Department of Computer Science and Engineering, 2000.

VITA

Zhichun Zhu

Zhichun Zhu is born in Zhuzhou, Hunan, China. She received her B.S. degree in Computer Engineering from Huazhong University of Science and Technology (HUST), Wuhan, Hubei, China, in 1992. She entered the Ph.D. program in Computer Science at the College of William and Mary in Fall 1998. Her research interests are computer architecture, performance evaluation, and parallel and distributed computing. She is a student member of the IEEE and the ACM.