Word-Interleaved Cache Architecture

Submitted in partial fulfillment of
the requirements for the degree of

Master of Science (by Research)
in
VLSI and Embedded Systems

by

Tavva Venkata Kalyan
Roll No. 200642004
kalyan_tv@research.iiit.ac.in
Center for VLSI and Embedded Systems Technologies

International Institute of Information Technology
Hyderabad, A.P., INDIA.
July, 2009

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "*Word-Interleaved Cache Architecture*" by Tavva Venkata Kalyan, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Advisor: Madhu Mutyam

Abstract

The proliferation of nanometer technologies with ever decreasing geometric sizes have led to proportionally unprecedented levels of system-on-a-chip (SOC) integration. Embedding a multitude of processor cores has become a widespread design practice due to its advantages of highly reduced time-to-market, lower design cost, and easily reprogrammable implementations. Yet, increased silicon integration, together with increased clock frequencies, has led to proportional increases in terms of power. Consequently, techniques for minimizing system power/energy consumption are of significant importance for achieving high-product quality.

On-chip caches play a major role in overall system performance. Being a part of the memory hierarchy, cache memory in the system has become a critical component for high performance. Though caches improve performance, they are also the major contributors to the system-wide power consumption. The memory hierarchy of a system can consume up to 50% of microprocessor system power. In the advanced process technologies even though static power dissipation due to the leakage takes an increasing fraction of total power dissipation in processors, dynamic power consumption is still an important factor in low-power/energy-efficient design techniques.

Motivated by the fact that at most one word is being read on a hit and no word on a miss, in this thesis we propose a novel energy-efficient cache architecture, namely, *word-interleaved (WI)* cache. Unlike the conventional set-associative caches, where cache line data belong to a single cache way, in the WI cache, cache line data are uniformly distributed among different cache ways in such a way that each cache way gets a portion of the cache line data. This distribution provides an opportunity to activate/deactivate the cache ways based on the offset of the requested address, thus minimizing the overall cache access energy. By experimental validation, we showed that the WI cache architecture, without incurring any performance loss, saves 66.4% and 58% dynamic energy over a $16KB$ 4-way set-associative conventional cache with $32B$ cache lines for instruction and data caches, respectively. With negligible area overhead, WI cache can be implemented easily as it does not change the internal RAM structure.

Dedicated to my parents

Acknowledgments

My thanks go first to my advisor, Madhu Mutyam, for the support, inspiration, and helpfulness he has showed me over the course of my graduate education. I have never met anyone with a keener eye for detail, and all of my research has benefitted greatly from his amazing ability to read not only for content but also for interesting new research directions. I can never thank him enough for all of the care and attention he has shown to me, for which I am truly grateful. Conversations with him were invaluable in helping me stay on track with my work and progress towards graduation. He is not only a good researcher but also a good person, I have learnt a lot from him as a person, I could not have had a better role model.

My parents have had a great influence on me. Where would I be without them? Without their warmth and love I wouldnot have reached this stage. They have been a great source of inspiration and courage for me. Because of their enormous support I am able to pursue a career and field of my choice.

I would like to express how much I have enjoyed working with Abid and Nayan this 2 years. Their abundant good cheer and positive outlook made the working experience tremendously pleasurable, and I specially thank them for helping me to drive out the ghost of programming inside me. I have had many simulating conversations with Anuj, Mahesh and Keerthiram. They have been of great help for me to develope a better perspective about all things around me.

My special thanks go to Raghavendra, our senior. He has been very supportive and helpful during his stay at IIIT-H. He has helped me a lot with understanding of basics of computer architecture, cache and simplescalar tool. I would also like to thank all my batchmates, Abu, Prashanth Pai, Rohit, Chandan, Raju, Rahul, Sai Satyanarayana, Pankaj and Vinayak for making my journey more fun. Also, thanks to all the M.Tech friends of my batch for making my stay at IIIT-H an enjoyable experience.

Whatever you do will be insignificant, but it is very important that you do it.
—Mahatma Gandhi —

Contents

List of Figures

List of Tables

Publications in course of this thesis

1. "*Word-Interleaved Cache: An Energy Efficient Data Cache Architecture*", T. Venkata Kalyan, Madhu Mutyam. In the proceedings of ACM/IEEE *International Symposium on Low Power Electronics and Design* (**ISLPED**), 2008, pp. 265-270.

2. "*Word-Interleaved Cache: An Energy Efficient Data Cache Architecture*", T. Venkata Kalyan, Madhu Mutyam. Communicated to ACM *Transactions on Architecture and Code Optimization* (**TACO**), 2009.

Publications related to the area of energy-efficient designs

1. "*Exploiting Variable Cycle Transmission for Energy-Efficient On-Chip Interconnect Design*", T. Venkata Kalyan, Madhu Mutyam and P. Vijaya Sankara Rao. In the proceedings of IEEE *International Conference on VLSI Design* (**VLSID**), 2008, pp. 235-240.

Chapter 1

Introduction

Since the invention of the transistor, decades ago, through the years leading to the 1990's, power dissipation, though not entirely ignored, was of little concern. Greater emphasis was on performance and miniaturization. With much of the research efforts of the past directed toward increasing the speed of digital systems, present-day technologies possess computing capabilities that make possible powerful personal workstations, sophisticated computer graphics, and multimedia capabilities. High-speed computation has thus become the expected norm from the average user, instead of being the province of the few with access to a powerful mainframe. Historically, applications powered by battery-pocket calculators, hearing aids, implantable pacemakers, portable military equipment used by individual soldier and, most importantly, wrist watches-drove low-power electronics. In all such applications, it is important to prolong the battery life as much as possible. These portable battery-powered applications of the past were characterized by low computational requirement.

1.1 Need for Low-power/Energy-efficient Design Techniques

Although the traditional mainstay of portable digital applications has been in low-power, low-throughput uses, there are an ever-increasing number of portable applications requiring low power and high throughput. People are beginning to expect to have access to same computing power, information resources, and communication abilities when they are travelling as they do when they are at their desk. For example, notebook and laptop computers, representing the fastest growing segment of the computer industry, are demanding

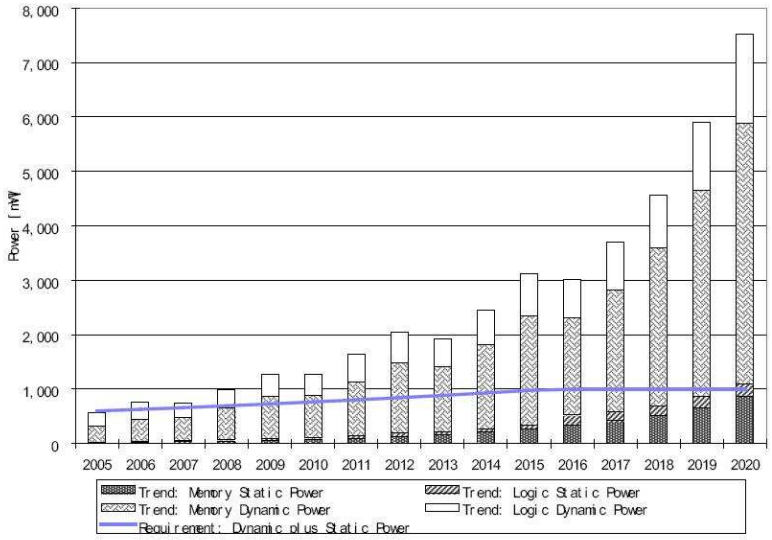


Figure 1.1: Power consumption trends in SOC designs as predicted by ITRS-2005 [22].

the same computation capabilities as found in desktop machines. These portable applications require huge processing power. Conventional nickel-cadmium battery technology only provides a 26W of power for each pound of weight [11] and projected improvement in battery performance is no way comparable to the present demand of power in portable devices. Another important consideration, particularly in portable applications, is that many computation tasks are likely to be real-time; the radio modem, speech and video compression, and speech recognition all require computation that is always at near-peak rates. Conventional schemes for conserving power in laptops, which are generally based on power-down schemes, are not appropriate for these continually active computations. Hence advances in the area of low-power microelectronics are required to make the vision of the inexpensive and high processing portable application a reality.

Even when power is available in nonportable applications, the issue of low-power design is becoming critical. To continue to improve the performance of the circuits and to integrate more functions into each chip, feature size has to continue to shrink. Figure 1.1 shows the trend for total chip power for power efficient System On Chip (SOC) designs, as predicted by International Technology Roadmap for Semiconductors (ITRS)-2005 [22]. In deep sub-micron region, each new process has inherently higher dynamic and leakage current density with minimal speed advantage. As a result the magnitude of power per unit

area is growing, by the end of year 2010 the power density on the chip might equal that of a rocket nozzle (as predicted by Pat Gelsinger, Intel) and the accompanying problem of heat removal and cooling is worsening. Difficulty in providing adequate cooling might either add significant cost to the system or provide a limit on the amount of functionality that can be provided. Circuit cooling requirements also affect package costs and circuit reliability [33]. Thus, it is evident that methodologies for the design of high-throughput, low-power digital systems are needed.

1.2 Sources of Power Consumption

There are three major sources of power dissipation in digital CMOS circuits, which can be summarized in the following equation:

$$P_{total} = p_t(C_L * V * V_{dd} * f_{clk}) + I_{sc} * V_{dd} + I_{leakage} * V_{dd} \tag{1.1}$$

The first term represents the switching component of power, where C_L is the loading capacitance, f_{clk} is the clock frequency, and p_t is the probability that a power consuming transition occurs (the activity factor). In most of the cases the voltage swing V is the same as the supply voltage V_{dd}; however, in some logic circuits (ex. single gate pass-transistor implementations) the voltage swing on some internal nodes may be slightly less. It can be seen that the first term indicates logic transitions. As the "nodes" in a digital CMOS circuit transition back and forth between the two logic levels, the parasitic (or loading) capacitances are charged and discharged. Current flows through the channel resistance of the transistors, and electrical energy is converted into heat and dissipated away. The second term is due to the direct-path short circuit current I_{sc}, which arises when both the NMOS and PMOS transistors are simultaneously active, conducting current directly from supply to ground [38]. With input(s) to the gate stable at either logic level, only one of the two transistors conduct and no short circuit current flows. But when the output of a gate is changing in response to change in input(s), both the transistors conduct simulatenously for a brief interval. The duration of the interval depends on the input and the output transition (rise or fall) times and so does the short circuit dissipation. Finally, leakage current $I_{leakage}$, which can arise from substrate injection and subthreshold effects, is primarily determined by fabrication technology considerations.

The first two sources of power dissipation in CMOS circuits are related to tran-

sitions at gate outputs and are therfore collectively referred to as *dynamic dissipation*. In contrast, the third source of dissipation, the leakage current flows when the input(s) to, and therefore the outputs of, a gate are not changing. This is called *static dissipation*. In the advanced process technologies, even though static power dissipation due to leakage takes an increasing fraction of total power in processors, dynamic power consumption is still an important factor in low power design techniques. While the problem is often stated in terms of power dissipation, the real goal is to minimize energy dissipation in the presence of performance constraints [36]. When the processing rate is a fixed constraint, the energy and power metrics are interchangeable. Thus, the metric to minimize is really energy-delay product.

In order to minimize the power consumption in a chip, in this thesis we concentrate on one of the important components of high performance systems, *on-chip cache*.

1.2.1 On-Chip Cache

On-chip cache memory in modern computer systems has become a critical component for high performance. By exploiting locality principle, cache memories can reduce the memory access latency, which in turn helps in improving the overall system performance. For high clock frequencies, these on-chip caches are implemented using arrays of densely packed static random access memory (SRAM) cells. As cache memories improve the system performance, modern computer systems employ multi-level cache organizations. Several cache organizations have been proposed in literature which include *direct-mapped*, *set-associative*, and *fully-associative* caches. In a n-way set-associative cache, the cache is partitioned into n ways so that data can be placed in one of the n ways. Direct-mapped and fully-associative caches are special cases of n-way set-associative caches, where n is equal to either 1 or the number of sets in the cache, respectively.

In direct-mapped caches, as data is placed in specific locations based on a fixed mapping associated with the cache, the access latency is small as compared to the same-sized associative cache organizations. Though the access latency of direct-mapped caches is small, mapping data to fixed locations can increase the *conflict* miss rate [19]. Associative cache organizations can minimize the conflict miss rate as data can be placed in one of the many locations. In order to minimize the effective access latency for a n-way set-associative cache, all the n ways are accessed in parallel and data is read/written from/to one of the

ways, which is selected by using a multiplexer. Accessing all the ways simultaneously for each cache read/write request results in high cache access energy consumption. Also, as multiplexer delay is in the critical path, the access latency of set-associative caches is higher compared to the direct-mapped cache access latency.

Though on-chip cache is an important component in modern processors for achieving high performance, because of its large area and high access frequency, it also becomes a major power consumer in processors. For instance, caches contribute nearly 16% and 43%, respectively, of the total processor power in Alpha 21264 [9] and StrongARM-110 [31] processors. Consequently, caches are the most attractive targets for power minimization.

1.3 Contributions

A low dynamic power (or energy) cache design can be viewed as one with minimal internal switching activity during an access. The activity can be attributed to charging/discharging of wordlines and bitlines, and firing of sense amplifiers. In this thesis we propose an energy-efficient cache architecture, namely, the *Word-Interleaved cache* wherein we eliminate redundant accesses to data arrays by *offset based decoding* mechanism. Through this method all the accesses in instruction/data cache can be directed to the target cache way immediately. This approach does not require any special circuitry internal to cache RAM, an advantage in industry for easy implementations.

1.4 Scope of thesis

Following this introduction, chapter 2 surveys the related work. Chapter 3 gives the experimental framework. Then, the main body of this dissertation delves into the important component chosen for low power/energy consumption:

In Chapter 4 we explain our proposed energy-efficient cache architecture, namely, *Word-Interleaved Cache*. An exhaustive analysis for a baseline cache configuration is provided, followed by sensitivity analysis considering different cache size, blocksize and associativity. We then compare/study our WI cache architecture with other cache techniques like cache exploiting fast hits, subbanking, phased-access cache and drowsy cache.

Finally, chapter 5 reviews the contributions of this dissertation and outlines future directions for this research.

Chapter 2

Prior Work

Identifying on-chip cache as an important component of high performance systems, in this chapter we present the existing low-power/energy-efficient designs available for caches.

In *phased-access cache* [18], all the tag arrays are accessed first followed by an access to a data array corresponding to the matching tag (if any). Though, the phased-access cache minimizes power consumption, it requires more time for every access, degrading the performance.

In *predictive sequential associative cache* [10], several prediction mechanisms are proposed to select a cache way in a set-associative data cache. By considering 2-way set-associative caches, the authors showed that their technique yields low miss rate as compared to a conventional 2-way set-associative caches and low cycle time as compared to the same sized direct-mapped caches. In *way-predicting set-associative caches* [21, 35], both the tag and data array of a predicted cache way are accessed first. If a misprediction occurs, the rest of the ways are accessed in parallel in the next cycle.

Reactive-associative cache design [8] uses both selective direct mapping and way prediction. As way prediction accuracy for data caches is very low, the authors exploit selective mapping mechanism to place most of the blocks in direct-mapped positions that are accessed first and reactively displacing the conflicting blocks to set-associative positions. Using the selective mapping and way prediction, it is shown that the performance of 4-way set-associative cache is close to that of the same-sized direct-mapped cache. But this technique requires extra time whenever the access to the first cache way results in a miss. In *access-mode-prediction* cache [48], cache accesses can be adaptively switched between the

way-prediction [21] and the phased accessing [18] modes. In the case of predicted cache hits, the way-prediction scheme determines the desired way and probes only that way. In the case of predicted misses, the phased scheme accesses all tags first, then probes the appropriate way. The proposed predictor uses a global-access history register and a global pattern history table to achieve reasonably high prediction accuracy.

Few cache design techniques have been proposed in which cache configurations can be tuned to a particular application. In *selective cache ways* [7] technique, cache ways are selectively activated based on the application's need. By doing so, power consumption can be minimized, but profiling is required to know the number of cache ways to be selectively activated. *Accounting cache* [14] is based on a re-sizable selective cache ways. The accounting cache first accesses part of the cache ways of a set-associative cache, known as a primary access. If there is a miss, the cache accesses the other ways, known as a secondary access. A swap between the primary and secondary accesses is needed whenever there is a miss in the primary and a hit in the secondary access. Energy is saved only on a hit during the primary access. In [45], the authors proposed a cache architecture wherein *way-concatenation*, *way-shutdown*, and *line-concatenation* techniques are used to configure associativity, cache size, and cache line size, respectively. Tuning these cache parameters to a program is a cumbersome task left for designers. So, in [44] the authors introduced on-chip hardware implementing an efficient cache tuning heuristic that can automatically, transparently, and dynamically tune the cache to an executing program.

Pseudo set-associative cache (PSAC) [20] is a set-associative cache having a tag array and a data array similar to a direct-mapped cache. Upon a miss, a specific index bit is flipped and a second access is made to the cache using the new index. So, each location in the cache is part of a *pseudo-set* consists of itself and the location obtained by flipping the index bit. PSAC achieves the speed of a direct-mapped cache and the hit rate of a 2-way cache, at the expense of slower access time than a 2-way cache due to sequential accessing of the cache ways.

In *low-power way-selective cache* [23], a specialized replacement policy is adopted to allocate cache lines with different least significant four bits of tag address in a set. This restriction can limit cache performance whenever different cache lines which belong to the same set with the same least significant four tag bits are encountered, which is indeed a realistic case. In *way-halting cache* [46], a small fully associative cache is maintained to store the lowest-order tag bits of all the cache ways. The fully associative cache is searched

in parallel with the set-index decoding and if a mismatch is found in the fully associative cache search, further access to the main cache is halted. *Dynamic zero-sensitivity* scheme [12] reduces average cache power consumption by preventing bitlines from discharging in reading a '0'. It requires a special hardware for gating the bitlines and sense amplifiers.

To eliminate redundant accesses to cache tag and data arrays, *tag encoding* technique is proposed in [47]. It places several registers in the cache to store the recently accessed address tags, and a register for each tag to store its state. Before starting an access, the tag states in the target cache set are checked to determine which way(s) should be accessed and which should not. This technique achieves energy savings with small increase in cache access time. In order to minimize power consumption in tag arrays of both instruction and data caches, *tag compression* for low-power embedded processors is proposed in [34]. This is also an application-specific customization of the cache. Compile-time algorithms are proposed for identifying the minimal number of tag bits for completely distinguishing the set of tags accessed by an application loop. By utilizing very few tag bits within the major application loops, a significant amount of energy dissipation in the tag arrays can be eliminated.

Exploiting small buffers between the L1 data cache and processor to reduce the data cache activity is explored in [39, 27]. As buffer access energy is small compared to a data cache access energy, as long as processor requests hit in the buffer, these techniques achieve significant energy savings. But a buffer miss requires an additional clock cycle to access the main cache, leading to performance degradation. A technique to alleviate this degradation for instruction cache is proposed in *hotspot cache* [43]. In [16], a number of low power techniques such as extra buffers between processor and the L1 cache, sub-banking of the cache, and bit-line partitioning, are proposed. Similar to the *filter cache* [27] mechanism, a low-power cache design is proposed in [13] by exploiting two-level filters between the processor and L1 cache.

Victim cache [30] design is proposed to reduce the number accesses to the L2 cache. In [32], a technique is proposed to reduce the number of data cache accesses by modifying the Load/Store queue (LSQ) design to retain previously accessed data values on both loads and stores after the corresponding memory access instruction has been committed. Hit in the modified LSQ is faster and also avoids a cache access hence, reducing the total energy spent on load instructions and improving the energy-delay product.

Several techniques have been proposed to reduce leakage energy in caches. *Cache decay* technique [24] uses a time based strategy, wherein by exploiting the generational behaviour of cache line usage, a cache-line is turned-off if it is not likely to be used. The drawback of this technique is that if a cache-line is turned-off and is needed later, it must be refetched from a lower level memory, which incurs both additional dynamic energy consumption and performance degradation. *Data-retention gated-ground* [5] cache retains the data in the cache line while reducing the leakage by using gated-Ground transistor. But it is not preferable for L1 caches because of the high wake-up penalty between the active mode and the leakage saving mode.

An efficient cache leakage minimization technique, *Drowsy cache* [15] technique lowers the supply voltage to a level near the threshold voltage. When a cache line is in the leakage saving mode (or drowsy mode), data will be retained but it cannot be accessed for a read or a write operation. Due to temporal locality, for reasonably long periods, most of the cache lines are unused. Thus, by putting infrequently used cache lines into the drowsy mode and keeping frequently accessed cache lines in the active mode, the drowsy cache reduces much leakage energy. Since the expected wake-up penalty is small (1 cycle in the $70nm$ process technology), these leakage energy savings are obtained without significant performance degradation. *Super-drowsy* [26] technique is a circuit refinement of the drowsy technique. In this technique, in order to alleviate the interconnect routing space, the multiple supply voltage sources are replaced by a single-V_{dd} cache line voltage controller with a Schmitt trigger inverter. This technique can achieve significant leakage savings with negligible performance penalty.

A word-interleaved cache for a clustered VLIW processor is proposed in [17], wherein following a fixed mapping, a cache line is distributed among different clusters. Each line of a cache bank holds some words of the cache line. In this cache organization, as each sub-cache line will reside in only one cluster, there is no data replication at all, but the tags are replicated in all cache clusters. Effective instruction scheduling techniques are proposed to exploit the interleaving property for improving the IPC.

Most of the previous approaches trade off something or the other to reduce power (or energy) consumption. A phased-access cache and low-power way-selective cache trade performance for energy. On the other hand, predictive sequential associative cache, way-predicting set associative caches, reactive associative cache, access-mode prediction cache, accounting cache, pseudo set-associative cache and filter cache do not have fixed hit-latencies

and hence increase non-determinism in latencies for instruction scheduling, leading to not only performance degradation but also energy wastage due to squashing and re-issuing of the instructions. Few of the above techniques like selective cache ways, way-concatenation, self-tuning cache [44] and tag compression need either profiling or compiler (software) support. For self-tuning cache, dynamic zero sensitivity scheme, way-halt cache and tag encoding techniques special circuitry is needed leading to area overhead.

Orthogonal to the existing works, by exploiting word-interleaving property, we now propose a novel low dynamic energy cache architecture for both instruction and data caches, which is software independent, has fixed hit time, has no performance loss, and negligible area overhead. In our new cache architecture, a cache line is uniformly distributed among the different cache ways in such a way that each cache way holds some words of the cache line. Whenever there is a request, we apply offset decoding to access only the required cache way thereby reducing the dynamic energy consumption significantly. We then study the effectiveness of the WI architecture over the conventional cache architecture by considering features such as *fast hit* mechanism [39], wherein a buffer is used to store most recently accessed cache line, *subbanking* [42, 16, 6] and *drowsy* mechanism [15] for leakage energy minimization. We also compare our WI architecture with the *phased-access cache* [18], which is an energy-efficient cache design.

Chapter 3

Experimental Setup

In order to validate our proposed techniques, we have simulated the processor using SimpleScalar tool and simulated SPEC2000 benchmarks on that processor. Our cache has been modeled using the CACTI tool and the transistor parameters have been obtained using the Predictive Technology Model. Given below are brief details of the tools, benchmarks and models used in evaluation.

3.1 SimpleScalar Tool

SimpleScalar [3] was created by Todd Austin. The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, one can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. The SimpleScalar tools are used widely for research and instruction.

SimpleScalar simulators can emulate the Alpha, PISA, ARM, and x86 instruction sets. The tool set includes a machine definition infrastructure that permits most architectural details to be separated from simulator implementations. Complex instruction set emulation (e.g., x86) can be implemented with or without microcode, making the SimpleScalar tools particularly useful for modeling CISC instruction sets. In our experiments, we have emulated Alpha instruction set. The configuration of the processor which we have considered is given in Table 3.1.

Parameter	**Value**
Issue Width	4 instructions/cycle (out of order)
RUU Size	64 instructions
LSQ Size	32 instructions
Branch Prediction	Bimodal with $2K$ entries
BTB	512 entries,4-way
Misprediction Penalty	18 cycles
# of ALUs	4 integer and 4 floating point
# of Mul/Div units	1 integer and 1 floating point
L1 D-cache	$16Kbyte$, 4-way (LRU), $32byte$ blocks, 2 cycle latency
L1 I-cache	$16Kbyte$, 4-way (LRU), $32byte$ blocks, 1 cycle latency
L2 cache	Unified, $256Kbyte$, 4-way (LRU), $64byte$ blocks, 12 cycle latency
Memory	160 cycles
ITLB	16-entry, $4K$ block, 4-way, 30 cycle miss penalty
DTLB	32-entry, $4K$ block, 4-way, 30 cycle miss penalty

Table 3.1: Base processor configuration

3.2 SPEC2000 Benchmarks

SPEC is the acronym of *Standard Performance Evaluation Corporation.* It was formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC CPU2000 is the industry-standardized CPU-intensive benchmark suite. SPEC designed CPU2000 to provide a comparative measure of compute intensive performance across the widest practical range of hardware. The implementation resulted in source code benchmarks developed from real user applications. These benchmarks measure the performance of the processor, memory and compiler on the tested system.

SPEC CPU2000 is made up of two subcomponents that focus on two different types of compute intensive performance:

- CINT2000 for measuring and comparing compute-intensive integer performance.
- CFP2000 for measuring and comparing compute-intensive floating point performance.

We have used 12 SPEC CPU2000 benchmarks to test our designs. Of these, 6 are integer benchmarks and rest are floating point. Tables 3.2 and 3.3 contain the descriptions of the benchmarks. More detailed descriptions on the benchmarks (with reference to papers, web sites, etc.) can be found in the individual benchmark directories in the SPEC benchmark

Name of Benchmark	**Programming Language**	**General Category**	**Description**
bzip2	C	Compression	Data compression program.
gcc	C	C Programming language complier	Runs a complier for Motorola 88100.
mcf	C	Optimization	vehicle scheduling problems.
parser	C	Word Processing	Analyzes an English sentence.
perlbmk	C	PERL Programming Language	Runs four perl script files.
twolf	C	Place and Route Simulator	Does placement and routing microchips

Table 3.2: Selected benchmarks in SPEC CINT2000 Suite.

tree [4]. For each benchmark we simulated the sequence of instructions which capture the core repetitive phase of the program, as given in Table 3.4 [25]. These benchmarks are chosen for their high L1 miss rates. Table 3.4 also lists the number of instructions skipped to reach the phase start (FFWD) and the number of instructions simulated (RUN).

3.3 CACTI Tool

In order to model the SRAM based cache structure, we have used CACTI tool. CACTI [2] is an integrated cache access time, cycle time, area, leakage, and dynamic power model. By integrating all these models together, one can have confidence that tradeoffs between time, power and area are all based on the same assumptions and, hence, are mutually consistent. CACTI is intended for use by computer architects to better understand the performance tradeoffs inherent in different cache sizes and organizations. With the internal circuits chosen, the model gives estimates that are within 10% of Hspice results. CACTI 4.1 [40] is the latest version of CACTI to be released. This new version adds a model for leakage power and updates the basic circuit structure and device parameters to better reflect the advances in scaling semiconductors (in the last ten years).

Name of Benchmark	**Programming Language**	**General Category**	**Description**
applu	Fortran 77	Computational Fluid Dynamics and Physics	Solution of five coupled parabolic/ elliptic PDE's
art	C	Image Recognition and Neural Networks	Theory 2 (ART 2) neural network is used to recognize objects in a thermal image.
equake	C	Simulation of seismic wave propagation in large basins	simulates the propagation of elastic waves
galgel	Fortran 90	Computational Fluid Dynamics	Calculates Grashof no. of the convective flow in liquids.
mesa	C	3-D Graphs Library	OpenGL library
mgrid	Fortran 77	Multi-grid solver	Computes 3D potential field.

Table 3.3: Selected benchmarks in SPEC CFP2000 Suite.

SPEC INT2000	FFWD	RUN	SPEC FP2000	FFWD	RUN
bzip2	744M	1.0B	applu	267M	650M
gcc	2.367B	300M	art	2.2B	200M
mcf	5.0B	200M	equake	4.459B	200M
parser	3.709B	200M	galgel	4.0B	200M
perlbmk	5.0B	200M	mesa	570M	200M
twolf	511M	200M	mgrid	550M	1.06B

Table 3.4: Benchmarks used for experiments

3.4 Predictive Technology Model

In order to calculate the transistor parameters like width (W), length (L), threshold voltage (V_{th}), etc., we have used the Predictive Technology Model (PTM). PTM provides accurate, customizable, and predictive model files for future transistor and interconnect technologies. These predictive model files are compatible with standard circuit simulators, such as SPICE, and scalable with a wide range of process variations. With PTM, competitive circuit design and research can start even before the advanced semiconductor technology is fully developed.

Chapter 4

Word-Interleaved Cache Architecture

Realizing the fact that on-chip caches are one of the important components in modern microprocessors, in this chapter[1] our energy-efficient cache architecture is introduced and explained in detail. Section 4.1 presents the working of the conventional cache architecture, considering a baseline configuration. Basic observation and motivation for the proposed architecture are put-forward in Section 4.2. Our *word-interleaved cache* is presented in Section 4.3, followed by complete energy modeling in Section 4.4. Section 4.5 gives brief experimental setup. Exhaustive analysis considering the baseline configuration is presented in Section 4.6. Section 4.7 continues the discussion providing sensitivity analysis considering different cache sizes, associativity and cache line sizes. In Section 4.8, we study our WI cache architecture in comparison with other cache techniques like cache exploiting fast hits, subbanking, drowsy cache and phased-access cache.

4.1 Conventional cache

As a baseline configuration, we consider a $16KB$ cache which is 4-way set-associative with line size of $32B$[2]. The reason for considering a 4-way set associative cache as baseline is that four ways yield a sufficiently good hit rate for most of the applications. Also, high

[1]Part of this chapter has been published as [41]. And most of the work has been communicated to ACM Transactions on Architecture and Code Optimization (TACO) journal.

[2]For basics on cache memories the reader is requested to go through [19].

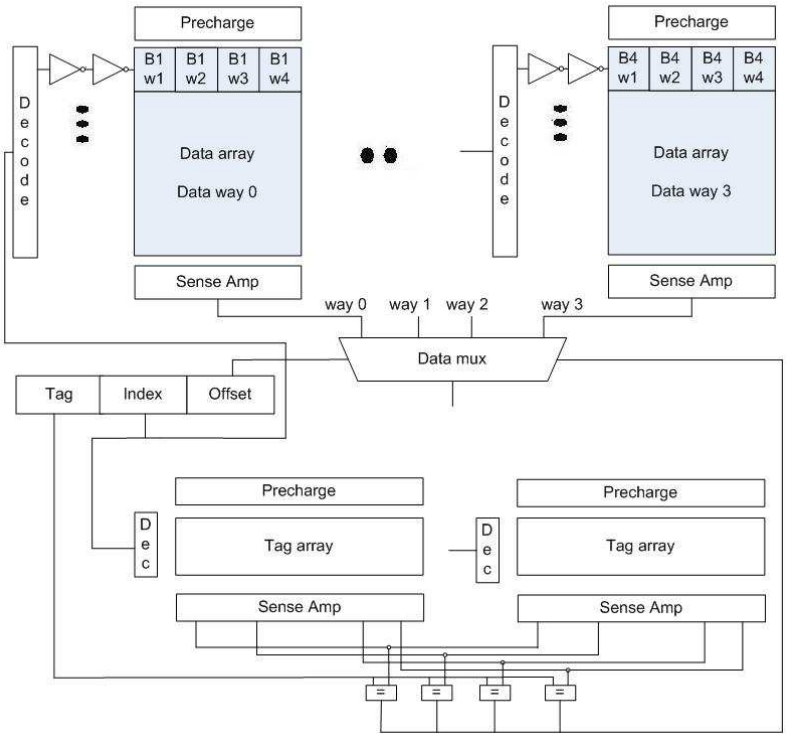


Figure 4.1: Conventional cache.

associativity caches already exist in some of the commercial processors. For example, the Intel Pentium 4 processor exploits 4-way L1 caches along with 8-way L2 cache.

The conventional cache architecture (as shown in Figure 4.1) using the baseline configuration includes a set-decoder, precharge circuits for both tag and data arrays, word-line drivers, four data arrays, four tag arrays, sense amplifiers, tag comparators, one multi-plexer (data_select_mux), and output drivers. During a cache access the following sequence of steps takes place: the index bits from a given address are sent to the set-decoder, which then selects a cache set; The wordline of the selected set is strengthened by the wordline drivers (typically, two back-to-back inverters); The wordline activates four blocks of the selected set (one in each way); Four tags and four blocks are read out simultaneously using the sense amplifiers; The tags are then compared with the tag of the given address using comparators (one for each tag) for a hit; Data of the hit way are sent out through the output drivers; Offset field is then used to extract the appropriate bytes from the selected block of data. This conventional parallel access scheme used in set-associative caches has good performance but is not optimized from energy consumption point of view. This is because, all the data ways are activated eventhough the required data is present in a single way.

4.2 Motivation

Our motivation for the *word-interleaved* (WI) cache design comes from the fact that at most one word is being accessed on a *cache hit* and no word is accessed on a *cache miss*. Whenever there is a request to the (instruction/data) cache, the processor accesses a word of data from a location specified by the request. From the requested data's address, if we know the cache way in which the data is present, we can access only the required cache way and eliminate the dynamic energy due to unnecessary access to other ways.

In the WI cache, data of a cache line are uniformly distributed among different cache ways so that each cache way gets a portion of the cache line data. Cache line data are distributed to all the cache ways uniformly in such a way that the first k words of the line are available in the first cache way, the next k words are available in the second cache way, and so on, and the last k words are available in the last cache way, where $k = S/n$ such that S is the cache line size in bytes and n is the associativity of the cache. With such a cache organization, it is easy to find a cache way for a given address using the offset of the address. We exploit the cache block distribution in the WI cache for the energy minimization by considering the *offset based address decoding* mechanism.

4.3 Word-Interleaved (WI) cache

Word-interleaved (WI) cache architecture is shown in Figure 4.2. While in the conventional cache architecture, all the words of a cache line are placed in contiguous locations (as shown in Figure 4.1), in the WI cache, words of a cache line are uniformly distributed among the different cache ways in such a way that the first $8B$ word (w_1) is available in the first cache way, the second $8B$ word (w_2) is available in the second cache way, the third $8B$ word (w_3) is available in the third cache way, and the last $8B$ word (w_4) is available in the fourth cache way. In general, for a n-way set-associative cache with cache line size as S bytes, we distribute the words of a cache line to all the cache ways in such a way that the i^{th} cache way gets data from $(i-1)\frac{S}{n}$ bytes to $(i\frac{S}{n}-1)$ bytes, where $1 \leq i \leq n$. From Figures 4.1-4.2, it is clear that all the words (w_1, w_2, w_3, and w_4) of a cache line (say B_1) are placed in a single way in the case of conventional cache, whereas in the WI cache, all the w_1's of the four cache lines (B_1, B_2, B_3, and B_4) in a set are placed in way 0, all w_2's are placed in way 1, and so on.

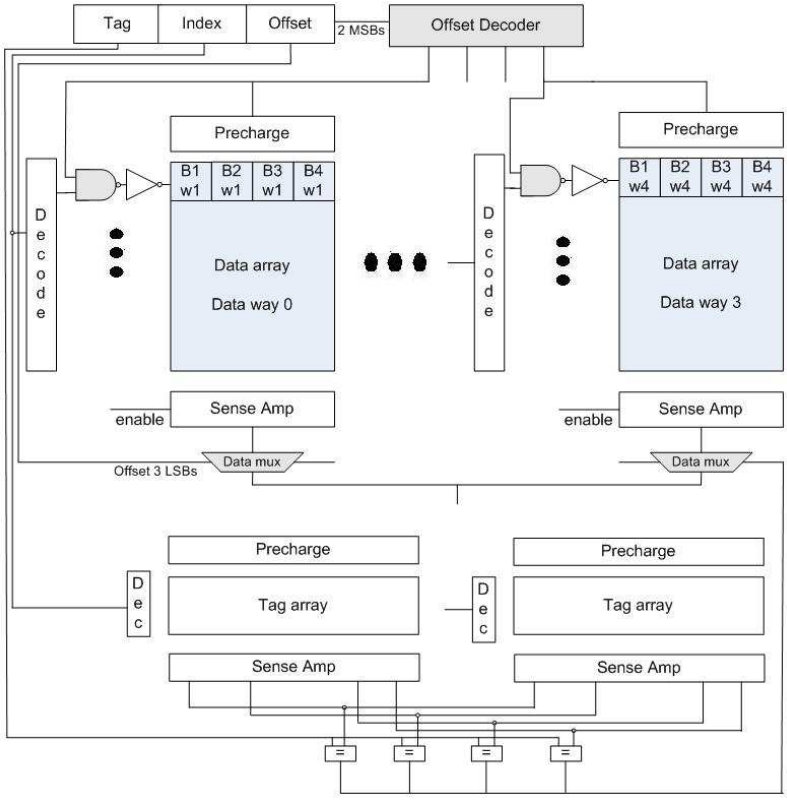


Figure 4.2: WI cache.

With the above arrangement of data in the WI cache, when there is an read request to the cache, both *index* and *offset* fields are decoded in parallel to determine the set and the way, respectively, thus restricting the access to a single way. The offset decoder (shown in Figure 4.3) is a simple 2×4 decoder, which requires the most significant two bits of the offset as input. Offset decoding takes less time as compared to the index decoding. Output of the offset decoder is used to select a cache way by enabling the corresponding precharge circuit and sense amplifiers. Also the decoded values of the offset and index fields are ANDed and the output is used to activate the wordline of the selected way. As words in a selected cache line differ in their tag addresses, we perform tag comparison and send out the selected word using the sense amplifiers and the output drivers. The least significant three bits of the offset are used to extract the appropriate bytes from the selected word.

We concentrate only on the energy consumption in the data array as the data array contributes significantly to the total energy consumption during a cache access. It can be observed that since only one cache way is activated per access, our technique can reduce

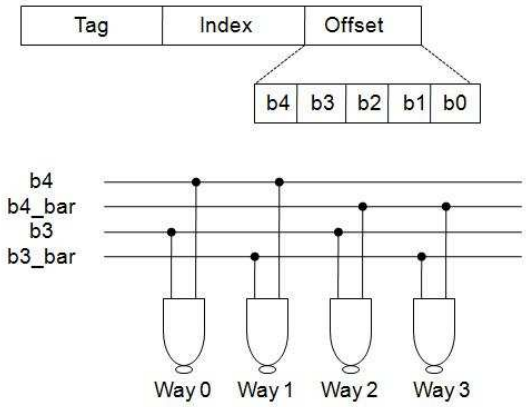


Figure 4.3: Offset decoder.

the energy consumption, eliminating the energy for activating the wordlines, bitlines, sense amplifiers, and data select multiplexers in the other three ways. Though we use four small data multiplexers (one for each way, each with $32 * 8$ input lines and $8 * 8$ output lines) instead of a bulk mux (with $4 * 32 * 8$ input lines and $32 * 8$ output lines) as in the baseline cache, there is a significant reduction in the energy consumption of multiplexer because only one of them is active per access.

During a write operation in the WI cache, as the data has to be written in all the four ways (because of cache line distribution), the wordlines and bitlines of all the four ways are activated. This results in more energy consumption for the WI cache architecture as compared to the conventional cache architecture. However, in order to reduce the energy overhead due to the sense amplifiers, during a write we activate the sense amplifiers of the bitlines of only those words which need to be written. Though energy consumption for the WI cache is more for a write operation as compared to the baseline cache, our technique achieves significant energy savings as the number of writes into a L1 cache is very less when compared to the number of read operations [19].

4.4 Energy modeling

In order to evaluate the effectiveness of the WI cache over the conventional cache in terms of dynamic energy minimization, in this section, we give analytical equations for dynamic energy consumption per cache access. Notation used in the analytical equations is shown in Table 4.1.

Notation	**Meaning**
E_{d_dec}	energy consumption in the data array decoder
E_{t_dec}	energy consumption in the tag array decoder
E_{d_rd}	sum of energy consumptions due to wordlines and bitlines of a single data array when the array is being accessed for a read
E_{d_wr}	sum of energy consumptions due to wordlines and bitlines of a single data array when the array is being accessed for a write
E_{t_rd}	sum of energy consumptions due to wordlines and bitlines of a single tag array when the array is being accessed for a read
E_{t_wr}	sum of energy consumptions due to wordlines and bitlines of a single tag array when the array is being accessed for a write
$E_{d\_SA\_blk}$	energy consumption values in sense amplifiers of a data array when a cache line is accessed
$E_{d\_SA\_4B}$	energy consumption values in sense amplifiers of a data array when a $4B$ word is accessed
E_{t_SA}	single tag way sense amplifier energy consumption
E_{mux_cc}	energy consumption in bulky multiplexer in the conventional cache
E_{mux_wi}	energy consumption in a single small mux in the WI cache
E_{comp}	energy consumption for comparison of tag bits of all the ways
E_{opd_blk}	output driver energy consumption when a cache line is accessed
E_{opd_word}	output driver energy consumption when a word of data is accessed
E_{opd_4B}	output driver energy consumption when a $4B$ data is accessed
E_{CC}	total dynamic energy consumption per access in the conventional cache
E_{WI}	total dynamic energy consumption per access in the WI cache
$E_{t\_1way\_rd}$	energy consumption to read a single tag array
$E_{t\_1way\_wr}$	energy consumption to write a single tag array
E_{t_4ways}	energy consumption in tag array to access all the four ways

Table 4.1: Notation used in the energy modeling equations.

4.4.1 Cache hit

To start with, let us consider an access to the L1 cache which resulted in a hit. Now we have two cases, the access can be for reading or writing data.

- *Read hit:*

$$E_{CC_{RH}} = E_{t\_4ways} + E_{d\_dec} + 4 * (E_{d\_rd} + E_{d\_SA\_blk}) + E_{mux\_cc} + E_{opd\_4B} \quad (4.1)$$

$$E_{WI_{RH}} = E_{t\_4ways} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{mux\_wi} + E_{opd\_4B} \quad (4.2)$$

where E_{t_4ways} is given by

$$E_{t\_4ways} = E_{t\_dec} + 4 * (E_{t\_rd} + E_{t\_SA}) + E_{comp}$$

It is clear from Equations (4.1) and (4.2) that whenever there is a read hit in the cache, the WI cache saves dynamic energy of three data ways. Also the multiplexer energy is reduced significantly since only one small mux is active in the WI cache to that of a bulk mux in the conventional cache.

- *Write hit:*

$$E_{CC_{WH}} = E_{t\_4ways} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \quad (4.3)$$

$$E_{WI_{WH}} = E_{t\_4ways} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \quad (4.4)$$

From the above equations, we know that both the conventional and WI caches consume same amount of energy for all writes which hit in the cache.

4.4.2 Cache miss

When an access to the L1 cache results in a miss, a cache line in a given set is selected as a victim following a replacement algorithm. If the victim line is valid and the dirty bit is *set*, we writeback the victim line to the lower level cache and replace it with the requested line (taken from the lower level cache). But if the dirty bit of the victim line is not set, we directly replace the victim line with the requested line. The access that resulted in a miss can be either a read or a write.

Here we have four possibilities: a read miss with the victim line as dirty and clean and a write miss with the victim line as dirty and clean.

- *Read miss with dirty victim line:*

$$\begin{aligned} E_{CC_{RMDV}} &= E_{t\_4ways} + E_{d\_dec} + 4 * (E_{d\_rd} + E_{d\_SA\_blk}) + E_{mux\_cc} + \\ & E_{t\_1way\_rd} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{opd\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_blk} \end{aligned} \tag{4.5}$$

where $E_{t\_1way\_rd}$ and $E_{t\_1way\_wr}$ are given by

$$\begin{aligned} E_{t\_1way\_rd} &= E_{t\_dec} + E_{t\_rd} + E_{t\_SA} \\ E_{t\_1way\_wr} &= E_{t\_dec} + E_{t\_wr} + E_{t\_SA} \end{aligned}$$

$$\begin{aligned} E_{WI_{RMDV}} &= E_{t\_4ways} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{mux\_wi} + \\ & E_{t\_1way\_rd} + E_{d\_dec} + 4 * E_{d\_rd} + E_{d\_SA\_blk} + E_{opd\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + 4 * E_{d\_wr} + E_{d\_SA\_blk} \end{aligned} \tag{4.6}$$

- *Read miss with clean victim line:*

$$\begin{aligned} E_{CC_{RMCV}} &= E_{t\_4ways} + E_{d\_dec} + 4 * (E_{d\_rd} + E_{d\_SA\_blk}) + E_{mux\_cc} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_blk} \end{aligned} \tag{4.7}$$

$$\begin{aligned} E_{WI_{RMCV}} &= E_{t\_4ways} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{mux\_wi} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + 4 * E_{d\_wr} + E_{d\_SA\_blk} \end{aligned} \tag{4.8}$$

From Equations (4.5) and (4.6) (or (4.7) and (4.8)), it can be observed that the WI cache consumes almost same energy as that of the conventional cache even though we need to activate all the four ways to read or write a complete line data for replacement. This is because of the fact that before we can ensure the access is a miss, we would have already read four data ways in conventional cache and only one data way in WI cache, making overall energy consumption almost equal.

- *Write miss with dirty victim line:*

$$\begin{aligned} E_{CC_{WMDV}} &= E_{t\_4ways} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{opd\_blk} + \\ & E_{t\_1way\_rd} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \end{aligned} \tag{4.9}$$

$$\begin{aligned} E_{WI_{WMDV}} &= E_{t\_4ways} + E_{d\_dec} + 4 * E_{d\_rd} + E_{d\_SA\_blk} + E_{opd\_blk} + \\ & E_{t\_1way\_rd} + E_{d\_dec} + 4 * E_{d\_wr} + E_{d\_SA\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \end{aligned} \quad (4.10)$$

- *Write miss with clean victim line:*

$$\begin{aligned} E_{CC_{WMCV}} &= E_{t\_4ways} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \end{aligned} \quad (4.11)$$

$$\begin{aligned} E_{WI_{WMCV}} &= E_{t\_4ways} + E_{d\_dec} + 4 * E_{d\_wr} + E_{d\_SA\_blk} + \\ & E_{t\_1way\_wr} + E_{d\_dec} + E_{d\_wr} + E_{d\_SA\_4B} \end{aligned} \quad (4.12)$$

For Equations (4.9) to (4.12), we assume *write allocate* [19] option for all write misses.

On a cache miss, if the selected victim line is invalid, it is replaced by the desired cache line (taken from the lower level cache). In this case, energy consumption is same as that of valid and clean victim line case, so that energy consumption can be modeled by Equations (4.7) and (4.8) or Equations (4.11) and (4.12) based on whether the access is for a read or a write, respectively.

From Equations (4.9) and (4.10) (or (4.11) and (4.12)), we know that the WI cache consumes more energy than the conventional cache as all the four ways have to be activated to read or write a complete cache line for replacement.

It can be noted that Equations (4.3),(4.4),(4.5),(4.6),(4.9) to (4.12) do not hold for instruction caches as processor does not write into a instruction cache.

4.5 Experimental Setup

In order to evaluate the WI cache, we consider $70nm$ technology node, modify the CACTI 4.0 power model. We simulate L1 cache with different configurations (total size, associativity and block size) to get the energy per access values of all the parameters described in Section 4.4. The area overhead is calculated by modifying the area model. The area overhead includes area for four small data mux, the area for replacing the first inverter in the wordline driver with a NAND gate and the offset decoder. To get the statistics of the L1 cache, we simulate 12 SPEC2000 CPU benchmarks (six integer and six floating point) using the Simplescalar 3.0 considering the base processor configuration given in Table 3.1.

Scenario	E_{CC} **(in** pJ**)**	E_{WI} **(in** pJ**)**
Read hit (RH)	89	29.9
Write hit (WH)	20.5	20.5
Read miss with dirty victim line (RMDV)	154	159
Read miss with clean victim line (RMCV)	107	84.6
Write miss with dirty victim line (WMDV)	89.7	154
Write miss with clean victim line (WMCV)	37.1	76.5

Table 4.2: Dynamic energy values per access in both the conventional and WI L1 cache architectures.

4.6 Effectiveness of the WI Cache Architecture

We now study the effectiveness of the WI cache architecture in terms of dynamic energy consumption and performance.

4.6.1 On dynamic energy consumption

Using CACTI 4.0, we calculate the energy consumption per access for both the conventional and WI caches for all possible scenarios as discussed in Section 4.4 and the corresponding energy values are shown in Table 4.2.

The overall dynamic energy consumption for both conventional and WI caches is computed as follows:

$$\begin{aligned} E_{overall} &= no\_of\_rd\_hits * E_{rd\_hit} + no\_of\_wr\_hits * E_{wr\_hit} \\ &\quad + no\_of\_rd\_misses * E_{rd\_miss} + no\_of\_wr\_misses * E_{wr\_miss} \end{aligned} \tag{4.13}$$

where $no\_of\_rd\_hits$ and $no\_of\_rd\_misses$ are the total number of read hits and misses, respectively, and $no\_of\_wr\_hits$ and $no\_of\_wr\_misses$ are the total number of write hits and misses, respectively. E_{rd_hit}, E_{wr_hit}, E_{rd_miss}, and E_{wr_miss} are the energy consumption values in the cache due to a read hit, a write hit, a read miss, and a write miss, respectively.

When the access energy for a read hit (refer to the first row in Table 4.2) is considered, the WI cache consumes about 33.6% of the total dynamic energy of a conventional cache. From Table 4.2, we can also observe that both the WI and conventional caches consume same energy for a write hit. For a read miss with dirty victim line, the WI cache consumes almost same energy as that of the conventional cache, but when it comes to write miss with dirty victim line, the WI cache consumes nearly twice the energy of the

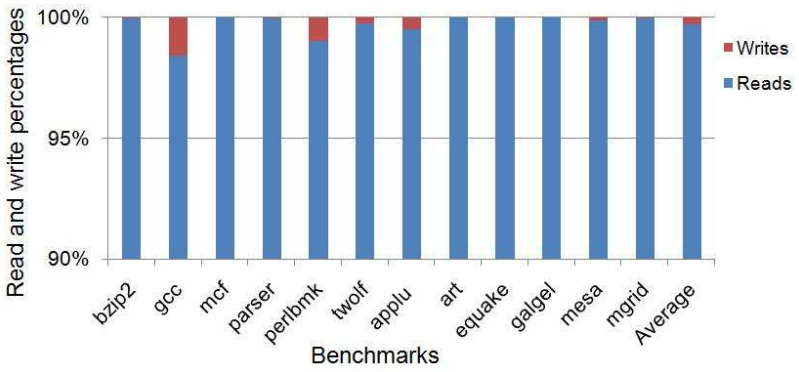


Figure 4.4: Percentage of reads and writes to the total number of accesses in the L1 instruction cache.

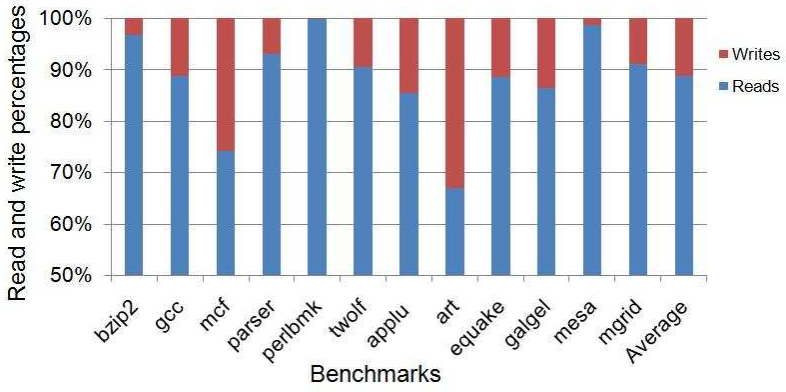


Figure 4.5: Percentage of reads and writes to the total number of accesses in the L1 data cache.

conventional cache. Figures 4.4 and 4.5 show the percentage of reads and writes in the L1 instruction and data caches, respectively. Note that writes in the instruction cache are due to replacements on a read miss only.

In the instruction caches, it is observed that reads contribute 99.7% of the total accesses whereas writes contribute very small fraction (0.3%). On the other hand, in the data caches, reads and writes contribute 88.8% and 11.2% of the total accesses, respectively, on an average. When overall energy consumption is considered (Eq. (4.13)), as the number of writes is very less compared to the number of reads, our technique can achieve significant energy savings in both instruction and data caches.

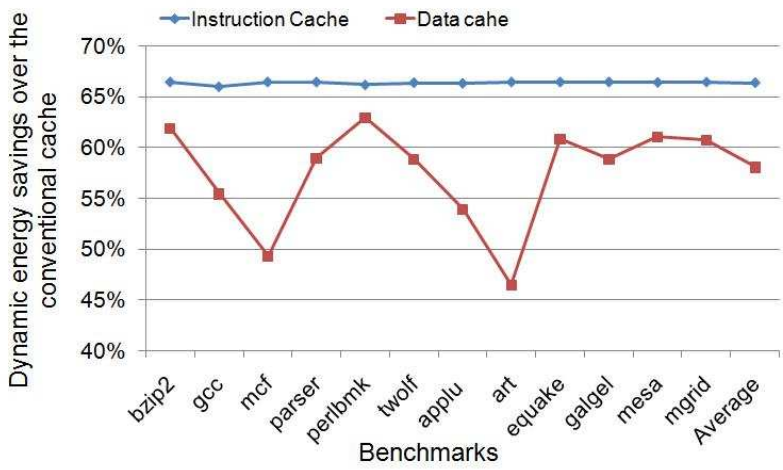


Figure 4.6: Dynamic energy savings of the WI L1 data cache over the conventional L1 data cache.

Figure 4.6 shows benchmark wise dynamic energy savings for the WI architecture over the conventional architecture for both instruction and data caches. When WI architecture is implemented in instruction caches, we obtain an average savings of 66.4% with minimum of 65.9% for "gcc". All the other benchmarks except "perlbmk" (66.1%) and "applu" (66.3%) achieve savings of 66.4%. On the other hand, for data caches, we obtain an average savings of 58.0% with a minimum of 46.4% for "art" and a maximum of 62.9% for "perlbmk" benchmark.

When WI architecture was considered for instruction cache, we obtain more savings than that of data cache because processor never writes into an instruction cache, eliminating the energy consumption because of writebacks. Also since the number of hits (Figure 4.7) in instruction caches are very much greater than the misses, the energy overhead incurred due to block replacements is decreased significantly, achieving near optimal savings when WI instruction cache is used.

From the Figure 4.6, we know that different benchmarks achieve different energy savings, much predominant in the case of data caches. This is mainly because, as shown in Figure 4.7, L1 data cache hit rates are different in all these benchmarks and the energy savings of the WI cache mainly depend on the L1 cache hit rate.

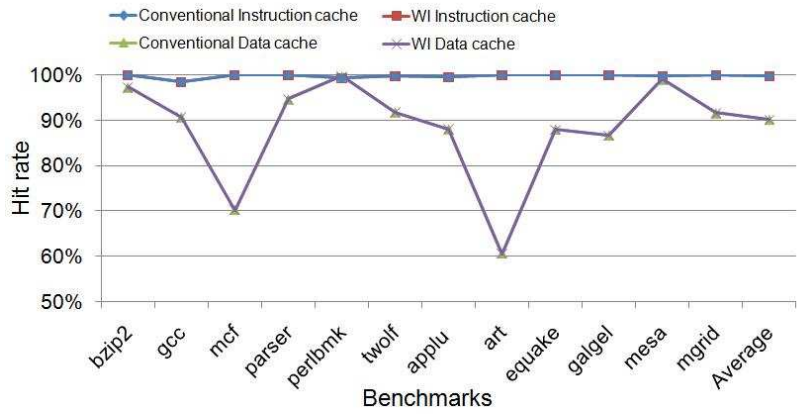


Figure 4.7: Benchmark-wise L1 cache hit rates for the WI cache. Note that the conventional cache has also the same hit rate as that of the WI cache.

4.6.2 On cache performance and area

Considering the baseline configuration, in this section we provide analysis of the WI cache architecture from performance point of view.

In order not to lengthen the critical path of the cache by adding an AND gate after the wordline driver inverters, we achieve the same logic by replacing the first inverter with a NAND gate. By carefully choosing the size of NAND gate, we can design it to be as fast as the original inverter. An identical technique of replacing the first inverter by a resized NAND gate is used in the way-halting cache [46] and way-concatenate cache [45] designs. The area overhead of the WI cache with respect to the baseline configuration is found to be very small (approximately 0.032%). The area overhead includes area for four small data mux (replacing one bulk mux), the area for replacing the first inverter in the wordline driver with a NAND gate and the offset decoder.

From the basic operation of the WI cache (and also from Figure 4.7), it can be noted that the hit rate of the WI cache is same as that of the conventional cache as we merely change the placement of the data in the data array. Thus, the WI cache does not incur any performance penalty with respect to the conventional cache.

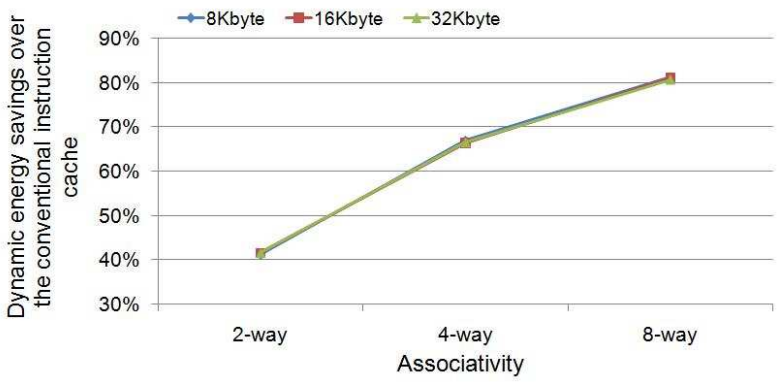


Figure 4.8: Average dynamic energy savings of the WI cache over the conventional cache for L1 instruction cache with different associativity and cache sizes, with $32B$ cache lines.

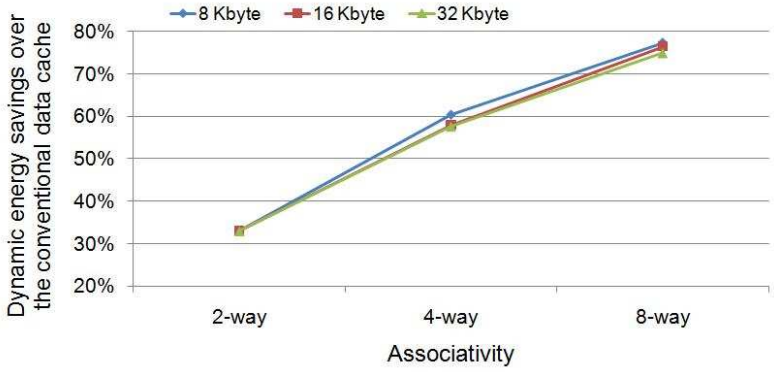


Figure 4.9: Average dynamic energy savings of the WI cache over the conventional cache for L1 data cache with different associativity and cache sizes, with $32B$ cache lines.

4.7 Sensitivity analysis

In this section, we study the effectiveness of our technique in terms of dynamic energy savings for L1 instruction and data caches by considering different associativity, cache sizes, and cache line sizes.

Figures 4.8 and 4.9 show the average dynamic energy savings obtained across the 12 SPEC2000 CPU benchmarks for 2-, 4-, and 8-way set-associative caches of various sizes with $32B$ cache lines, for instruction and data caches respectively. For a given cache size, we can observe a similar pattern of significant increase in the energy savings with increase in the associativity. This is because irrespective of the associativity, we always activate only

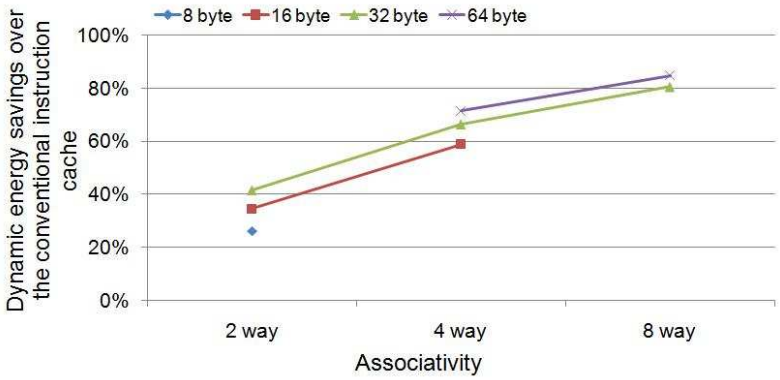


Figure 4.10: Average dynamic energy savings of the WI cache over the conventional cache for different associativity and blocksizes for a $32KB$ L1 instruction cache.

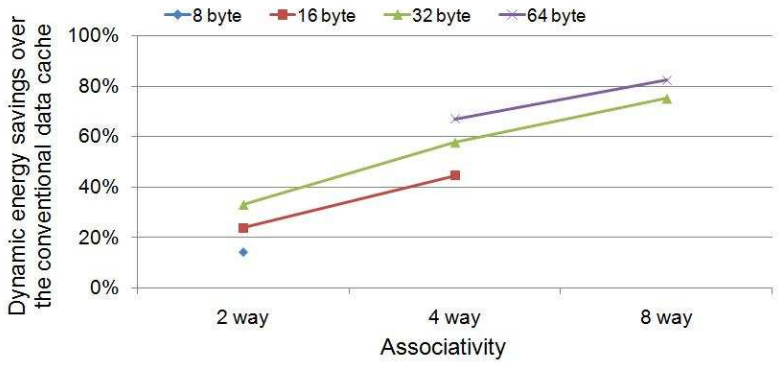
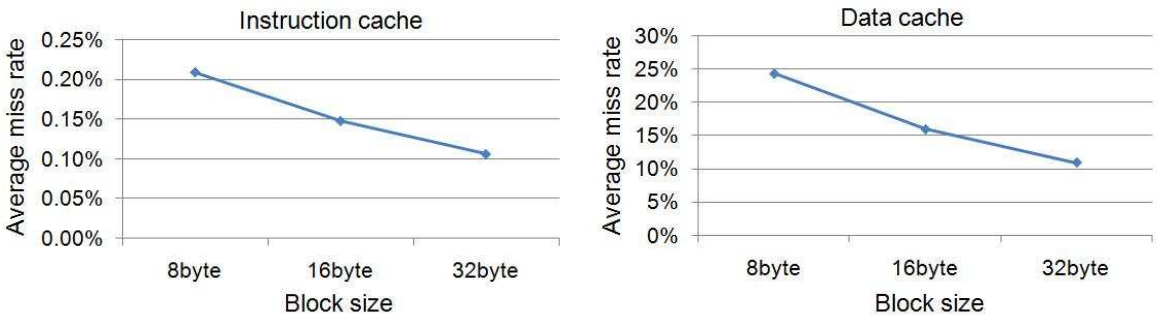


Figure 4.11: Average dynamic energy savings of the WI cache over the conventional cache for different associativity and blocksizes for a $32KB$ L1 data cache.

one way per access, yielding higher savings at higher associativity. We also see from the figure that the energy savings of the WI cache are not much dependent on the cache size (and also on the number of sets), but for data caches, there is a small decrease in the energy savings (about 3% to 4%) as size increases. For a given associativity and cache line size, as cache size increases, the number of sets in the cache increase, which in turn increases the bitline activation energy, thus the dynamic energy consumption increases, reducing the overall dynamic energy savings.



(a) Instruction cache

(b) Data cache

Figure 4.12: L1 instruction and data cache miss rates for a 2-way $32KB$ cache with different cache line sizes. Note that miss rate is same for both conventional and WI caches.

For instruction and data caches, Figures 4.10 and 4.11 show the variation in the average dynamic energy savings of the WI caches as cache line size is varied from $8B$ to $64B$, respectively. We can observe that as line size increases, the overall energy savings increase. When a program exhibits higher spatial locality, a larger cache line size can reduce cache misses. For example, Figure 4.12 shows average miss rates in L1 instruction and data caches for a 2-way $32KB$ cache with cache line sizes varied from $8B$ to $32B$. So, as the cache line size increases, miss rate decreases, which in turn increases the energy savings (since this reduces the need to activate all the ways for writing back the data block and writing the new/requested cache line).

It can be observed that in Figures 4.10 and 4.11, we omit few cache line sizes for different associativity because to implement the WI architecture, a minimum block size is needed, which is calculated from the product of minimum possible word size ($4B$) and associativity. For example, for an 8-way set associative cache, minimum block size needed to implement the WI cache architecture is $32B$.

4.8 Comparison of the WI cache architecture with other cache techniques

In this section we study WI architecture in comparison with other cache techniques like cache exploiting fast hits, subbanking, phased-access cache and drowsy cache. For comparison/study with phased-access cache and drowsy cache we consider only data caches.

4.8.1 Exploiting fast hits

Due to memory locality property, there is a high probability that the same cache line can be requested in the subsequent cache accesses. By exploiting this property, a cache design technique is proposed in [39], which stores the last accessed cache line in a buffer. So, whenever there is a request to the cache, the buffer is checked first and if there is a match, we avoid accessing the cache, thus save the dynamic energy required to access the cache. We call such hits in the buffer as *fast hits*. Exploiting fast hits phenomenon not only reduces the overall system power but also improves IPC as access to the buffer is much faster than access to cache. Note that, the cache is accessed as and when the request misses in the buffer, so that in such cases, we incur one extra cycle penalty for cache access.

When we combine the fast hit mechanism (by considering a buffer to store the most recently accessed word) along with our WI cache architecture, the WI cache achieves more dynamic energy savings over the conventional cache with fast hit mechanism. The reasons are several-fold: the WI cache with fast hit mechanism accesses only one way as compared to accessing all 4-ways in the conventional cache with fast hit mechanism; multiplexer consumes less energy in the WI cache; output drivers of only a wordsize are active in the WI cache with fast hit mechanism as opposed to output drivers of a complete cache line size are active in the conventional cache with fast hit mechanism; buffer consumes more energy in the conventional cache with fast hit mechanism as it is required to store index and tag along with the whole cache line data, whereas in the case of WI cache with fast hit mechanism, the buffer stores index, tag, and offset bits with only one word of data. From our simulations, we observe that an access to the buffer consumes $9.45pJ$ and $4.93pJ$ in the conventional and WI caches with fast hit mechanisms, respectively.

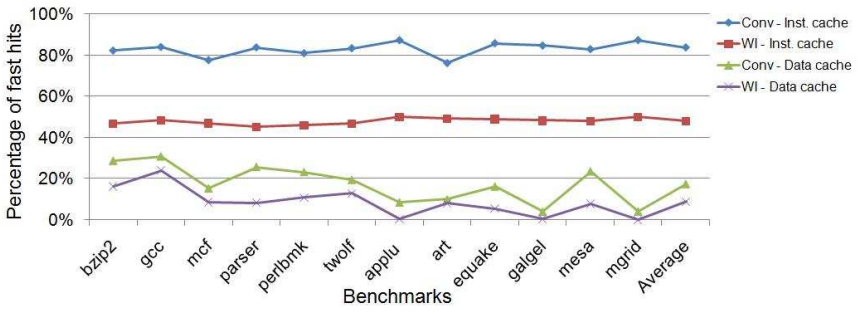


Figure 4.13: Benchmark-wise fast hits percentage for both instruction and data caches, considering both conventional and WI architectures.

From the above description, when fast hits are considered, the energy consumption for a read hit can be modeled as

$$E_{CC_{RH}} = E_{t\_4ways} + E_{d\_dec} + 4 * (E_{d\_rd} + E_{d\_SA\_blk}) + E_{mux\_cc} + E_{opd\_blk} \quad (4.14)$$

$$E_{WI_{RH}} = E_{t\_4ways} + E_{d\_dec} + E_{d\_rd} + E_{d\_SA\_blk} + E_{mux\_wi} + E_{opd\_word} \quad (4.15)$$

From equations 4.14 and 4.15 it was found that, WI cache consumes only 23.0% of total read access energy of conventional cache. For all the other scenarios, Eq. (4.3) to (4.12) are valid for the fast hits case also.

Figure 4.13 shows the percentage of fast hits in the case of instruction and data caches when the conventional and WI cache architectures are considered. Due to low temporal and spatial locality in data belongs to the data cache, the number of fast hits is very less. It can be seen from Figure 4.13 that in the case of data cache the fast hits constitute only 17.5% and 8.8% of the total hits for the conventional and WI cache architectures, respectively. Hence the impact of fast hits on the WI cache architecture is not so significant in terms of performance and energy consumption as compared to the conventional architecture. But in the case of instruction caches, due to high temporal and spatial locality, the number of fast hits is very high for the conventional cache (83.8%) as compared to the WI cache (48.1%). Because of less number of fast hits in WI instruction cache, the performance loss when compared to conventional instruction cache is very much high. This makes the WI cache architecture less preferable for the instruction caches, when considered in combination with fast hits.

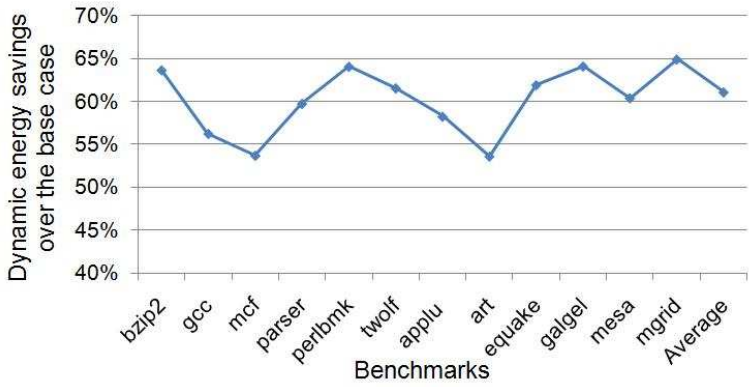


Figure 4.14: Energy savings of the WI L1 data cache over the conventional L1 data cache considering fast hits.

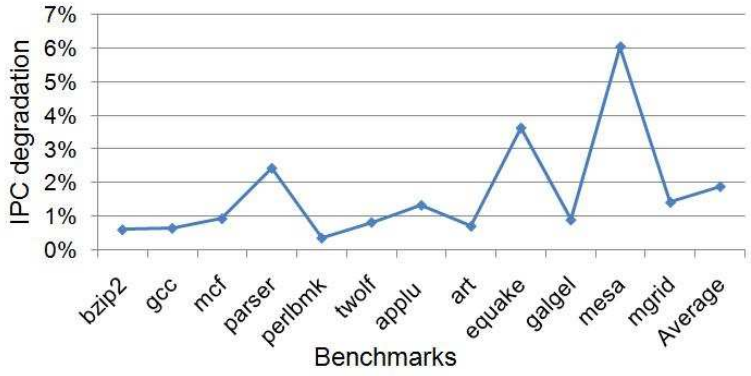


Figure 4.15: Benchmark-wise performance degradation for the WI L1 data cache with fast hit mechanism over the conventional L1 data cache with fast hit mechanism.

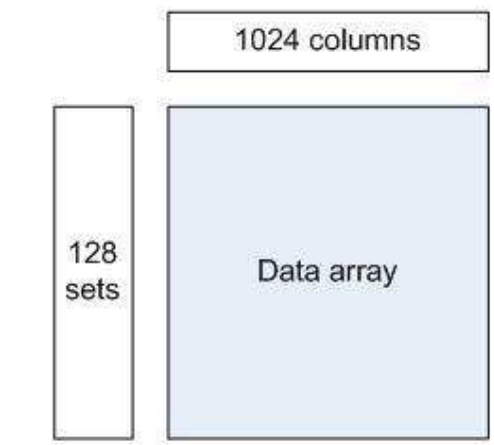
Figure 4.14 shows benchmark wise energy savings for the WI L1 data cache with fast hit mechanism over the conventional L1 data cache with the fast hit mechanism. We obtain total energy savings in the range of nearly 53.6% ("art") to 64.9% ("mgrid") with an average of 61.0%. When overall energy savings are observed, the savings of the WI data cache with fast hit mechanism are more than that of the WI cache without fast hit mechanism. Note that even though reduction in the number of fast hits increases the number of accesses to the L1 data cache, we obtain higher overall energy savings because of two reasons: in each access, significant energy is saved in the output drivers; buffers in the WI cache consumes less energy than that of the conventional cache.

Figure 4.15 shows benchmark-wise performance degradation for the WI L1 data cache with fast hit mechanism over the conventional L1 data cache with fast hit mechanism. From the figure, it is clear that most of the benchmarks incur small performance degradation. On the other hand, benchmarks "parser", "equake", and "mesa" incur 2.4%, 3.6%, and 6.0% IPC degradation, respectively. This is because for these benchmarks, the number of fast hits in the WI cache is very small as compared to that of the conventional cache. Overall, the WI data cache with fast hit mechanism incurs a performance penalty of 1.9% with respect to the conventional L1 data cache with fast hit mechanism.

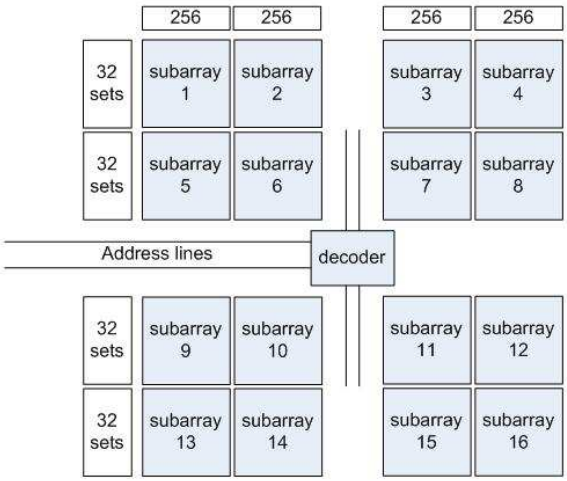
4.8.2 Considering subbanking

Typical organization of a SRAM cache was shown in Figure 4.1. Such an organization could result in an (data/tag) array that is larger in one direction or the other, creating slower bitlines or wordlines and resulting in a longer than necessary access time. To avoid this, a large array is partitioned into a number of identically sized subarrays (or subbanks) [42, 16, 6]. In order to achieve this, CACTI tool uses six organizational parameters, three for each of data and tag arrays. For a given cache configuration (total size, associativity and block size), N_{dwl} and N_{dbl} indicate the extent to which data array can be divided. The parameters N_{dwl} and N_{dbl} specify the number of times the array has been spilt with vertical cut lines (creating shorter wordlines) and horizontal cut lines (creating shorter bitlines), respectively. The total number of subarrays is thus given by $N_{dwl} * N_{dbl}$. The third parameter which mainly depends on block size and output bit width is N_{spd}, which indicates the number of sets that are mapped on to a single wordline. This parameter is very important as it allows the overall access time of the array to be changed without breaking it into even smaller subarrays. N_{spd} can even take fractional values, implying that few words of a block of each way can be mapped to a single wordline. Similar to the data array, for tag array we have three parameters N_{twl}, N_{tbl} and N_{tspd}.

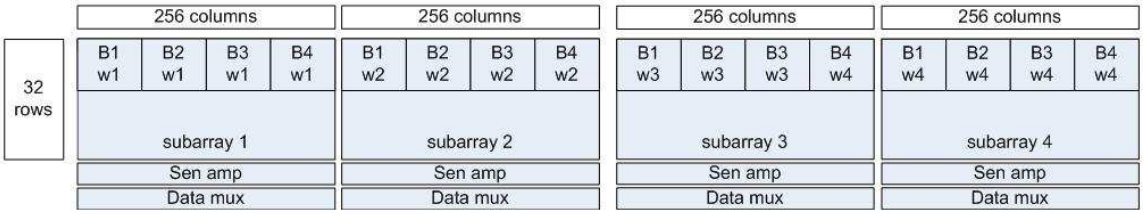
For each cache configuration, CACTI calculates optimal values of these six organizational parameters via exhaustive search to achieve the best trade off of cache size, performance and energy consumption. As it can be seen, these organizational parameters effect the overall energy consumption in a cache by reducing the energy dissipation in wordlines, bitlines, sense amplifiers and data multiplexor of the data array. Here we point out that in order to implement the WI cache we have not changed the subarray organization.



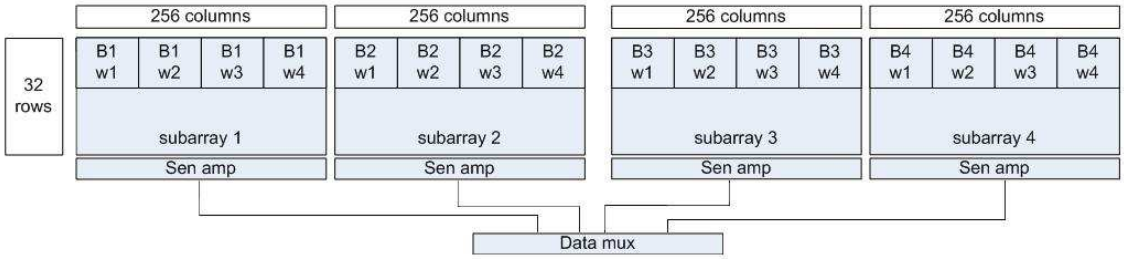
(a) Original data array



(b) Data array after subbanking



(c) Subarrays 1, 2, 3 and 4 - Conventional cache



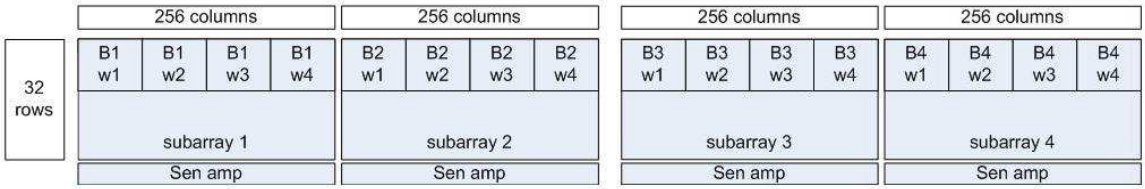
(d) Subarrays 1, 2, 3 and 4 - WI cache

Figure 4.16: Subbank division and arrangement in the data array, considering conventional and WI cache architectures.

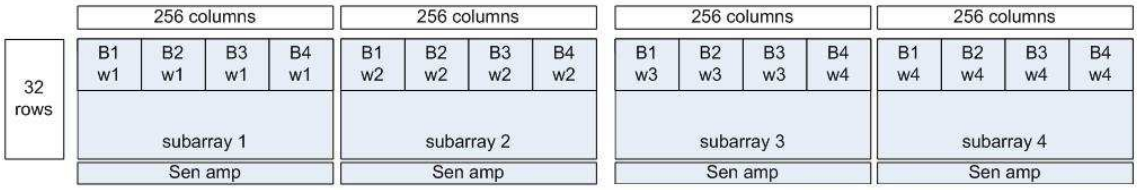
For the baseline configuration considered, the cache outputs only $4B$ of data. The optimal organizational parameters for the data array, for such a configuration are found to be $N_{dwl} = 2$, $N_{dbl} = 8$ and $N_{spd} = 0.5$. So, the complete data array is spilt into 16 small subarrays. Figures 4.16(a) and 4.16(b) show the original data array and arrangement of subbanks, respectively. Each subbank consists of 32 rows and 256 columns. In the Figure 4.16(b) only the first level of address decoding H-tree is presented. Figures 4.16(c) and 4.16(d) present the details of subbanks 1, 2, 3 and 4 for conventional and WI cache architectures, respectively. Each row of these four subarrays forms a set. Since N_{spd} is 0.5, subarrays 1 and 2 (similarly, 3 and 4) contribute half of each of the lines, hence share a single wordline. As assumed previously cache lines B1, B2, B3 and B4 form a set, each with four words (w1, w2, w3 and w4) of size $8byte$ each.

From Figure 4.16(c), the arrangement of words (and also four small data mux instead of one bulk mux) in the conventional cache after the subbanking looks much similar to that of WI cache (without subbanking, Figure 4.2), but in subbanking, the arrangement is based only on the organizational parameters. During a read access, the conventional cache with such an arrangement of words among subarrays activates all the wordlines and bitlines in all the four subarrays (1, 2, 3 and 4). But in CACTI, in order to reduce energy consumption in sense amplifiers and data mux for output widths less than a block size, sense amplifiers and data mux of only one subarray are activated, signifying that inherently there is offset decoding done at this stage. For a write access, when writing a complete line of data, due to distribution of the words among subarrays, wordlines and bitlines of all four subarrays are activated along with activation of sense amplifiers of word size in each of the subarray.

In case of WI cache architecture, as seen from Figure 4.16(d), the arrangement of words after the subbanking is much similar to that of conventional cache (without subbanking, Figure 4.1). With this arrangment, when there is a read access, since we consider access to a subbank (or subarray) only at line size granularity, we need to activate all the wordlines and bitlines in the four subarrays (1, 2, 3 and 4). But, when it comes to sense amplifiers, since we know the word needed (from offset decoding) we can activate sense amplifiers of only a word size from these subarrays. The four words obtained from the four subarrays are driven to the data mux, wherein using the result of tag comparision, only the required word is sent out. On the other hand, for writing a complete block of data, with such arrangement of words, we need to activate wordlines and bitlines of only one subarray.



(a) Subbanked conventional cache



(b) Subbanked WI cache

Figure 4.17: Subbanks 1, 2, 3 and 4 and arrangement of words in them when the baseline cache outputs $32B$ of data.

From the above details of working of a subbanked conventional cache and a subbanked WI cache, we can observe that both the caches consume same amount of energy for a read operation. For a write operation though, subbanked WI cache saves some energy because it activates wordlines and bitlines of only one subarray. When the overall energy savings are considered, it was found that in case of instruction cache both conventional and WI caches with subbanking consume almost same amount of energy. For data caches though, we obtain around 4% of savings in subbanked WI cache over subbanked conventional cache. This should not discourage one from implementing the WI cache architecture along with subbanking because the arrangement of words after subbanking completely depends on the cache configuration alone.

In order to prove this, we consider another cache configuration and study the combined affect of subbanking and word interleaving. Consider the baseline configuration, $16KB$, 4-way, $32B$ cache now with output bitwidth of $32B$. This is the case when the L1 cache exploits the fast hits phenomenon. The organizational parameters for this configuration are found to be $N_{dwl} = 4$, $N_{dbl} = 4$ and $N_{spd} = 1$. In this case also we have 16 subarrays and subarray arrangement shown in Figure 4.16(b) still holds here. But major difference comes in the arrangement of words in the subbanks. Figure 4.17 shows the subarrays 1, 2, 3 and 4 of the data array and arrangement of words in these subarrays for both conventional and WI cache architectures. Here the four subarrays share a common wordline (since N_{spd}

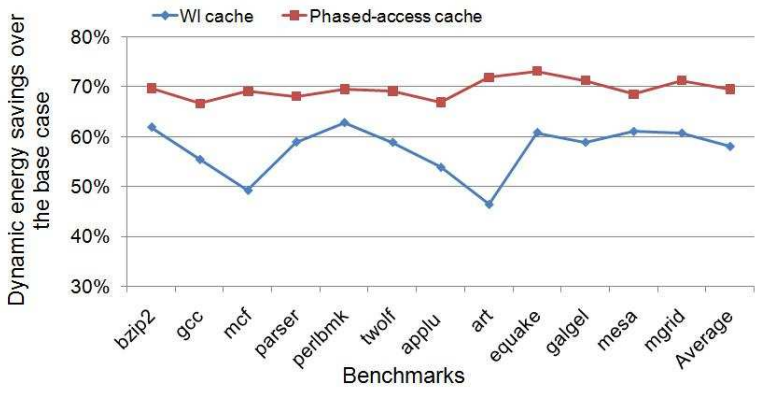


Figure 4.18: Benchmark-wise energy savings of the WI cache and phased-access cache over the conventional L1 data cache.

is 1). From Figures 4.17(a) and 4.17(b), we can observe that arrangement of words is similar to that of conventional and WI caches in Figures 4.1 and 4.2, respectively. So, the equations of energy consumption in conventional cache and WI cache (given in Section 4.4) are valid here. The average energy savings obtained considering fast hits and subbanking in WI data cache is 58.7%, much close to average savings obtained when subbanking is not considered (61%).

From the above examples, we can observe that when WI architecture is considered along with subbanking technique, the effectiveness of WI cache architecture in terms of dynamic energy savings is completely dependent on the cache configuration.

4.8.3 Phased-access cache

In this subsection we compare the WI cache with phased-access cache [18], considering only data caches because phased cache is generally preferred only for data caches. When we consider dynamic energy consumption, phased-access cache provides the best possible energy-efficient cache design, accessing one and only one way per access. But the downside of it, due to sequential access of tag and data arrays, the phased-access cache incurs significant performance penalty.

Figure 4.18 shows the dynamic energy savings obtained by the WI and phased-access caches over the conventional data cache. For all the benchmarks, phased-access cache achieves significantly more savings than WI cache (in case of "art" savings are 25.5%

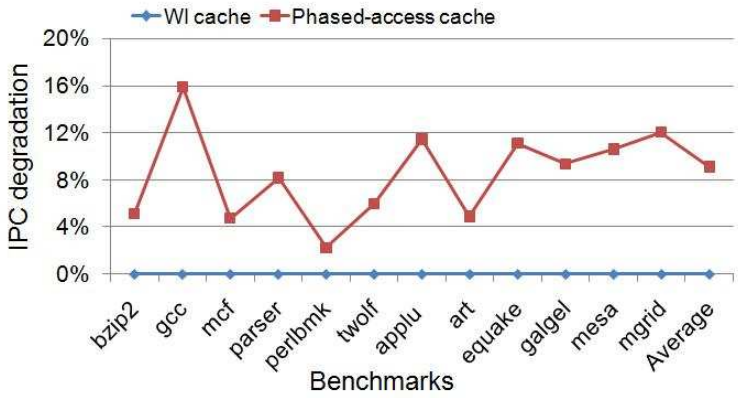


Figure 4.19: Benchmark-wise performance degradation of the WI cache and phased-access cache over the conventional L1 data cache.

more). On an average, phased-cache achieves 11.5% more savings than the WI cache. This is mainly because of two reasons: in phased-access cache, data select multiplexer is not present as only one data way is accessed; unlike the conventional and phased-access caches, the WI cache incurs more energy consumption when writing a complete line of data.

Figure 4.19 shows benchmark-wise performance loss incurred by the WI and phased-access caches over the conventional data cache. As discussed earlier in Section 4.6.2, the WI cache does not incur any performance penalty with respect to the conventional cache. The phased-access cache incurs an average performance loss of 9.1% with a maximum performance degradation 15.9% (in the case of "gcc" benchmark).

4.8.4 Exploiting drowsy mechanism

In order to minimize the leakage energy consumption, we implement the drowsy cache mechanism in our WI L1 data cache. Drowsy cache technique [15] is a well known leakage minimization technique for L1 data caches as it retains the data in the SRAM cell and has a short wake-up latency. In drowsy mechanism, data is put into a low-leakage or drowsy mode whenever it is not likely to be reused. In drowsy mode, the supply voltage of the cell is very less, near to the threshold voltage, thus the leakage is reduced significantly. Data is accessible only when the block is in the normal or active mode. On the other hand, in drowsy mode, accessing the data will destroy the cell contents. So, whenever data in drowsy mode needs to be accessed, it must first be brought back to normal mode. Drowsy

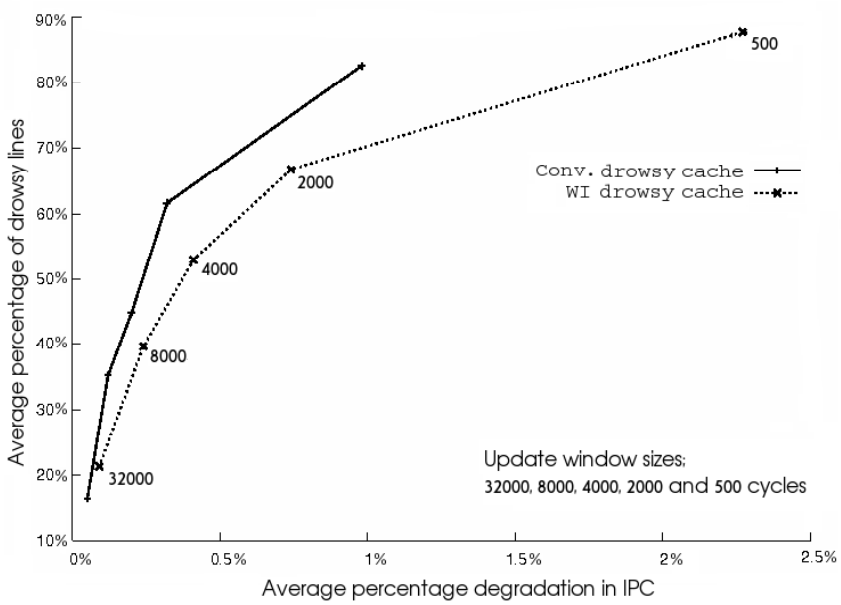


Figure 4.20: Impact of update window size on performance and the percentage of cache lines in drowsy mode for both the conventional and WI drowsy L1 data caches.

to normal mode transition incurs some performance loss.

We now compare the WI drowsy L1 data cache with conventional drowsy L1 data cache. In the conventional drowsy cache, whenever there is an access, cache lines in all the ways of the target set are woken up from drowsy to the active mode even though required data is present only in one way. When the WI drowsy cache is considered, with the help of offset decoding, we can activate only one cache line per access. Thus the time for which a block remains in drowsy mode is relatively more in the WI drowsy cache as compared to the conventional drowsy cache.

In order to verify the above observation, we consider a simple policy wherein all the cache lines are put into drowsy mode periodically. This simple policy is found to be efficient in reducing leakage [15]. In [26], the authors proposed a circuit refinement of drowsy technique called *super-drowsy* technique. In this technique, in order to alleviate the interconnect routing space, multiple supply voltage sources are replaced by a single-V_{dd} cache line voltage controller with a Schmitt trigger inverter. Also, using this technique, we can lower the supply voltage much more than that of normal drowsy technique [15] to reduce cell leakage power. This circuit technique, though proposed for instruction caches,

can also be used for data caches. In our experiments, we consider the circuit technique of super-drowsy mechanism with the simple policy of periodically placing all the cache lines in drowsy mode. It can be noted that, in this study, we do not concentrate on reducing the bitline leakage power.

Figure 4.20 shows the impact of update window size on both performance and the percentage of drowsy lines of both conventional and WI drowsy cache architectures for a $16KB$ 4-way set associative data cache with $32B$ cache lines. The figure shows the average IPC degradation and the average percentage of drowsy lines across the 12 SPEC2000 CPU benchmarks. We consider one cycle wake-up penalty. It can be clearly seen that the percentage of drowsy lines is more in the WI drowsy caches as we bring only one cache line to active mode per access.

On the other hand, there is a negative effect of word-interleaving on the performance of drowsy cache. Performance loss is more in the WI drowsy cache as compared to the conventional drowsy cache because there is a high probability that between two update periods different words of a same cache line can be requested. In such a case, the cache line is already in the active mode in the conventional cache, but in the WI drowsy cache, since only one word is active, when there is request for another word of the same cache line, an additional cycle penalty is incurred to activate the block containing the newly requested word. For example, consider a cache line B_1 which contains four $8B$ words w_1, w_2, w_3, and w_4. In the conventional drowsy cache, all the words are brought into active mode when accessed for the first time, even though only one word is required. This has an advantage that if another word of the same cache line is requested during the same interval, the word can be accessed immediately, without any wake-up penalty. When we consider WI cache, during an interval, if w_1 of B_1 is requested, only that word is turned active (along with w_1's of other blocks of the set). Now during the same interval, if any other word of B_1 is requested, we have to wake-up the corresponding word. We consider different update window sizes such as 500, 2000, 4000, 8000 and 32000 cycles. As the window size increases, performance penalty is decreased along with the reduction in the percentage of cache lines in drowsy mode.

For the remaining part of the work, we consider an update window size of 2000 cycles as it provides good compromise between percentage of drowsy lines and performance. With update window size of 2000 cycles, on an average, we incur 0.3% and 0.7% of performance degradation with 61.6% and 66.7% of drowsy lines for the conventional drowsy

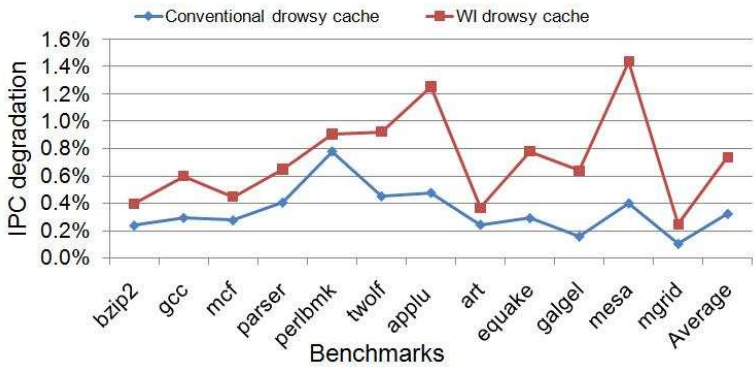


Figure 4.21: Benchmark-wise IPC degradation for both the conventional and WI drowsy caches with respect to the base case with an update window size of 2000 cycles.

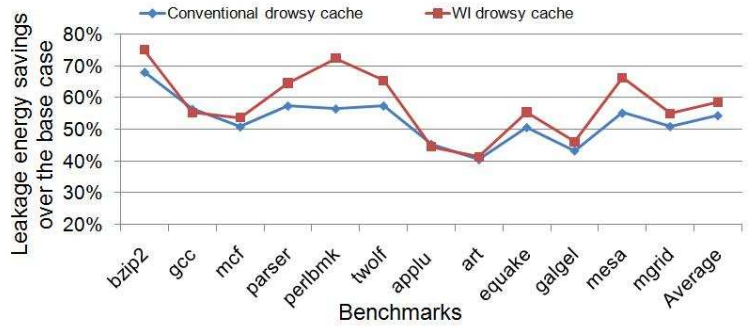


Figure 4.22: Benchmark-wise leakage energy savings for both the conventional and WI drowsy caches with respect to the base case with an update window size of 2000 cycles.

cache and WI drowsy cache, respectively.

Detailed power values for drowsy and normal modes are obtained from HSPICE studies on a 6 transistor SRAM memory cell in $70nm$ technology using Predictive Technology Model [1] circuit simulation parameters. We consider supply voltage (V_{dd}) and threshold voltage (V_t) as $1V$ and $0.2V$, respectively. Our simulations show that a memory cell consumes leakage power of $58.3nW$ per active bit at $V_{dd} = 1V$ and $6.6nW$ per drowsy bit at $V_{dd} = 0.25V$ with $2.64fJ$ and $6.37fJ$ transition energy penalties for drowsy-to-active and active-to-drowsy transitions, respectively. Figure 4.21 gives benchmark wise performance degradation incurred by the conventional drowsy and WI drowsy data caches as compared to the baseline data cache.

Figure 4.22 shows the leakage energy savings for the conventional and WI drowsy caches with respect to the baseline data cache by considering an update window size of 2000 cycles. We can observe that in almost all the benchmarks we obtain more savings in the WI drowsy cache than the conventional drowsy cache. On an average, the WI drowsy cache achieves nearly 9% more savings than that of the conventional drowsy cache. Due to transitions between active-to-drowsy and drowsy-to-active modes, we incur some dynamic energy consumption. On an average, for the conventional drowsy cache, overhead due to these transitions is 4.4% of the dynamic energy consumption of the base case and for the WI drowsy cache the overhead is found to be 4%.

Chapter 5

Conclusions and Future Work

This chapter summarizes the contributions of the thesis and then throws light on possible directions for future work.

As technology is scaling, both for portable and non-portable applications, energy/power consumption minimization has become the most important parameter in designs. This is because of the fact that for portable applications battery performance is not improving as demanded by the applications and when non-portable applications are considered we have drastic increase in power density values, which arise many realiability and cooling issues. On-chip cache is an important component in modern processors for achieving high performance. Because of its large area and high access frequency, it also becomes a major energy consumer in processors. Consequently, in this thesis we proposed an energy-efficient cache architecture for high performance microprocessor systems.

Ideally, one would desire performance of a set-associative cache with energy consumption of that of a direct-mapped cache. We proposed an energy-efficient cache architecture, namely, the word-interleaved (WI) cache which nearly achieves this ideal scenario. In the WI cache, a cache line is uniformly distributed among the different cache ways in such a way that each cache way holds some words of the cache line. By experimental validation, we showed that the WI cache architecture, without incurring any performance loss, saves 66.4% and 58% dynamic energy over a $16KB$ 4-way set-associative conventional cache with $32B$ cache lines in instruction and data caches, respectively.

Sensitivity analysis was done considering different cache sizes, associativity and cache line sizes. It was noticed that for a given cache line size, as associativity increases, there is a significant increase in energy savings because irrespective of the associativity, we

always activate only one way per access. But the savings were not much dependent on the cache size (also number of sets). When various cache line sizes were considered for a given cache size and associativity, we found that savings increase with increment in line size. The reason behind this being decrease in the number of misses with increase in line size, which in-turn reduces the miss penalty incurred.

Realizing the importance of fast hits phenomenon, when a cache which exploits fast hits is considered, it was found that the WI cache architecture is not suitable for instruction caches and achieves average dynamic energy savings of 61.0% for data caches. But due to less number of fast hits in the WI data cache, these energy savings are obtained with a performance penalty of 1.9%. We also studied the combined effect of WI cache with subbanking and found that the effectiveness of WI cache is highly dependent on the cache configuration.

In order to study the effectiveness of WI cache architecture on leakage energy consumption, we implemented drowsy mechanism in our WI cache architecture. It was found that the WI drowsy cache achieves higher minimization in leakage than that of conventional drowsy cache. This is because whenever there is an access, in WI drowsy cache, only one cache line is activated (woken-up from drowsy to active mode) per access, whereas in the conventional drowsy cache, all the lines of a set are activated in every access. But this increased savings are obtained at an increased performance degradation over the conventional drowsy cache. For the baseline configuration, on an average, the conventional drowsy and WI drowsy data caches incur 0.3% and 0.7% of performance penalty with 54.5% and 58.6% of leakage energy savings, respectively.

Advantages of our architecture lie in its simple control logic, no modification to SRAM array in cache and no increase in non-determinism in hit latency of L1 caches. It can be implemented in caches with associativity larger than one, and is worthy of being implemented in high-associativity caches, usually employed in some high-performance commercial processors.

Process variation is a serious problem in nanoscale technologies. The special feature of WI cache is that one can have access at block level granularity. This feature might be helpful in reducing the impact of process variations on performance of caches. As a part of future work one can investigate variation tolerant techniques for caches which can exploit access to block level granularity.

Bibliography

[1] Predictive technology model. http://www.eas.asu.edu/~ptm/

[2] CACTI Tool. http://www.hpl.hp.com/personal/Norman_Jouppi/cacti4.html

[3] SimpleScalar toolset. http://www.simplescalar.com

[4] SPEC 2000 Benchmark. http://www.spec.org

[5] A. Agarwal, *et al.* "DRG-cache: A Data Retention Gated-ground cache for low power". Proceedings of the *conference on Design automation*, 2002, pp. 473-478.

[6] B. Amrutur and M. Horowitz,"Speed and power scaling of sram's". *IEEE Journal of Solid-State Circuits (JSSC)*, 2000, pp. 175-185.

[7] D. Albonesi, "Selective cache ways: on-demand cache resource allocation". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 1999, pp. 248-259.

[8] B. Batson and T.N. Vijaykumar. "Reactive-associative caches". Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 2001, pp. 49-60.

[9] D. Brooks, *et al.*, "Wattch: a framework for architectural-level power analysis and optimizations". Proceedings of the *International Symposium on Computer Architecture (ISCA)*, 2000, pp. 83-94.

[10] B. Cadler, *et al.*, "Predictive sequential associative cache". Proceedings of the *International Symposium on High Performance Computer Architecture (HPCA)*, 1996, pp. 244-253.

[11] A. Chandrakasan, S. Sheng, and R. Brodersen. "Low-power CMOS digital design". *IEEE Journal of Solid-State Circuits (JSSC)*, 27(4), 1992, pp. 473-484.

[12] Y. Chang and F. Lai, "Dynamic zero-sensitivity scheme for low-power cache memories". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 25(4), 2005, pp. 20-32.

[13] Y. Chang, *et al.*, "Design and analysis of low-power cache using two-level filter scheme". *IEEE Transactions on Very Large Scale Integration Systems*, 11(4), 2003, pp. 568-580.

[14] Dropsho, *et al.*, "Integrating adaptive on-chip storage structures for reduced dynamic power". Proceedings of the *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2002, pp. 141-152.

[15] K. Flautner, *et al.*, "Drowsy caches: simple techniques for reducing leakage power". Proceedings of the *International Symposium on Computer Architecture (ISCA)*, 2002, pp. 148-157.

[16] K. Ghose and M.B. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 1999, pp. 306-315.

[17] E. Gibert, J. Snchez, A. Gonzlez. "Effective Instruction Scheduling Techniques for an Interleaved Cache Clustered VLIW Processor". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 2002, pp. 123-133.

[18] A. Hasegawa, *et al.*, "SH3: high code density, low power". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 15(6), 1995, pp. 11-19.

[19] L. Hennessey and D.A. Patterson, *Computer Architecture: A Quantitative Approach.* 4th Edition, Elsevier Science & Technology Books, 2006.

[20] M. Huang, *et al.*, "L1 data cache decomposition for energy efficiency". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2001, pp. 10-15.

[21] K. Inoue, *et al.*, "Way-predicting set-associative cache for high performance and low energy consumption". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 1999, pp. 273-275.

[22] "Power-Efficient System-on-Chip power Trends, System Drivers", International Technology Roadmap for Semiconductors (ITRS), 2005.

[23] P. Jung-Wook, *et al.*, "Power-aware deterministic block allocation for low-power way-selective cache structure". Proceedings of the *International Conference on Computer Design (ICCD)*, 2004, pp. 42-47.

[24] S. Kaxiras, *et al.*, "Cache decay: exploiting generational behavior to reduce cache leakage power". *ACM SIGARCH Computer Architecture News*, 29(2), 2001, pp. 240-251.

[25] C. Kim, D. Burger and S. W. Keckler, "An Adaptive, NonUniform Cache Structure for WireDelay Dominated OnChip Caches". Proceedings of the *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002, pp. 211-222.

[26] N. S. Kim, *et al.* "Single-vdd and single-vt super-drowsy techniques for low-leakage high-performance instruction caches". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2004, pp. 54-57.

[27] J. Kin, *et al.*, "The filter cache: an energy efficient memory structure". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 1999, pp. 184-193.

[28] D. Liu et al. "Power consumption estimation in CMOS VLSI chips". *IEEE Journal of Solid-State Circuits (JSSC)*, 1994, 26, pp. 663-670.

[29] M. Meijer, J. Pinede de Gyvez, and R. Otten. "On-Chip Digital Power Supply Control for System-on-Chip Applications". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2005, pp. 311-314.

[30] G. Memik, *et al.*, "Reducing energy and delay using efficient victim caches". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2003, pp. 262-265.

[31] J. Montanaro, *et al.*, "A 160 MHz, 32b 0.5W CMOS RISC microprocessor". Proceedings of the *International Solid-State Circuits Conference (ISSCC)*, 1996, pp. 214-215.

[32] D. Nicolaescu, *et al.*, "Reducing data cache energy consumption via cached load/store queue". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2003, pp. 252-257.

[33] M. Pedram, "Power minimization in IC design: principles and applications". *ACM Transactions on Design Automation of Electronic Systems*, 1996, pp. 3-56.

[34] P. Petrov and A. Orailoglu, "Tag compression for low power in dynamically customizable embedded processors". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7), 2004, pp. 1031-1047.

[35] M. Powell, *et al*, "Reducing set-associative cache energy via way-prediction and selective direct-mapping". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 2001, pp. 54-65.

[36] J. M. Rabaey, "System-level power estimation and optimization challenges and perspectives. Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 1997, pp. 158-160.

[37] J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Prentice-Hall, Inc., 1996.

[38] K. Roy and Sharat Prasad, *Low-Power CMOS VLSI Circuit Design*, John Wiley & Sons, Inc., 2000.

[39] C. Su and A. Despain, "Cache design tradeoffs for power and performance optimizations: a case study". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 1995, pp. 63-68.

[40] D. Tarjan, S. Thoziyoor and N. Jouppi, "CACTI 4.0: An Integrated Cache Timing, Power and Area Model", Technical report, HP Laboratories, Palo Alto, June 2006.

[41] T. Venkata Kalyan, Madhu Mutyam, "Word-Interleaved Cache: An Energy Efficient Data Cache Architecture". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2008, pp. 265-270.

[42] T. Wada, S. Rajan and S. Przybylski, "An analytical access time model for on-chip cache memories". *IEEE Journal of Solid-State Circuits (JSSC)*, 1992, pp. 1147-1156.

[43] C. L. Yang and C. H. Lee, "Hotspot cache: joint temporal and spatial locality exploitation for i-cache energy reduction". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2004, pp. 114-119.

[44] C. Zhang, *et al.*, "A self-tuning cache architecture for embedded systems". *ACM Transactions on Embedded Computing Systems*, 2004, pp. 407-425.

[45] C. Zhang, *et al.*, "A highly configurable cache for low energy embedded systems". *ACM Transactions on Embedded Computing Systems*, 2005, pp. 363-387.

[46] C. Zhang, *et al.*, "A way-halting cache for low-energy high-performance systems". *ACM Transactions on Architecture and Code Optimization*, 2005, pp. 34-54.

[47] M. Zhang, *et al.*, "Reducing cache energy consumption by tag encoding in embedded processors". Proceedings of the *International Symposium on Low Power Electronics and Design (ISLPED)*, 2007, pp. 367-370.

[48] Z. Zhu and X. Zhang, "Access-mode predictions for low-power cache design". Proceedings of the *International Symposium on Microarchitecture (MICRO)*, 22(2), 2002, pp. 58-71.