

Power-Aware Dynamic Cache Partitioning for CMPs

Isao Kotera¹, Kenta Abe¹, Ryusuke Egawa²,
Hiroyuki Takizawa¹, and Hiroaki Kobayashi²

¹ Graduate School of Information Sciences, Tohoku University
Sendai, 980-8578, Japan

{isao@sc., abeken@sc., tacky@}isc.tohoku.ac.jp

² Cyberscience Center, Tohoku University
Sendai, 980-8578, Japan

{egawa, koba}@isc.tohoku.ac.jp

Abstract. Cache partitioning and power-gating schemes are major research topics to achieve a high-performance and low-power shared cache for next generation chip multiprocessors(CMPs). We propose a power-aware cache partitioning mechanism, which is a scheme to realize both low power and high performance using power-gating and cache partitioning at the same time. The proposed cache mechanism is composed of a way-allocation function and power control function; each function works based on the cache locality assessment. The performance evaluation results show that the proposed cache mechanism with a performance-oriented parameter setting can reduce energy consumption by 20% while keeping the performance, and the mechanism with an energy-oriented parameter setting can reduce 54% energy consumption with a performance degradation of 13%. The hardware implementation results indicate that the delay and area overheads to control the proposed mechanism are negligible, and therefore hardly affect both the entire chip design and performance.

1 Introduction

Recently, as CMOS technology advances, the number of available on-chip transistors for microprocessors has been exponentially increasing. So far, this has been the driving force for improving performance of microprocessors. However, it is difficult to keep the exponential performance improvement by technology scaling, due to an increase in the power dissipation, the limitation of increasing the clock frequency and the limitation of instruction-level parallelism[1]. A Chip Multiprocessor(CMP) is a promising architecture to effectively utilize a large amount of hardware budget on a chip and to enhance the performance by using thread-level parallelism[2].

To realize high performance CMP, on-chip shared cache mechanisms play the key role. However, a large shared cache faces two severe problems. One is that resource sharing among cores degrades the CMP performance. The other

problem is that its large area leads to high power dissipation. The performance degradation problem is caused by conflicts on shared resources among cores. Cores generally share an L2 cache in a CMP. Cores can virtually use a large size cache by cache sharing, and threads in an application can share data via the shared cache. However, such a shared cache causes performance degradation if there are many cache access conflicts among cores. One solution to this problem is to increase the cache size. However, this approach leads to an increase in power consumption and an inefficient use of a large cache for most applications.

Suh et al. have first investigated the dynamic partitioning of a shared cache[3] to solve the performance degradation problem. They described a *marginal gain*-based cache partitioning algorithm and a low-overhead control scheme. Chandra et al. have also studied the performance impact of L2 cache sharing by threads on a CMP architecture, and proposed three models in order to accurately predict the performance using the stack distance and circular sequence profile of each thread[4, 5]. The utility-based cache partitioning[6] proposed by Qureshi et al. achieves a high performance benefit with a low-overhead hardware configuration. In addition, they described that their partitioning algorithm has a higher scalability. However, these studies have not discussed the power consumption well.

On the other hand, the power dissipation due to driving a vast amount of hardware on a chip is becoming a critical problem in high-performance micro-processor design. Especially, static power consumption due to leakage current will become more dominant in the total power dissipation, as the gate width becomes smaller[7, 8]. According to the International Technology Roadmap for Semiconductors[9], leakage power is expected to dominate more than 80% of the total power[10]. Therefore, computer architects have to consider static power consumption to realize power-aware computing[11].

Several approaches to reductions in static power consumption of a cache have been proposed. The mechanism of selective cache ways[12] provides functionality to turn off power supply to cache ways for reducing dynamic energy and is controlled by a performance metric given by users. The control is based on the result of application profiling. Powell et al. have proposed the DRI i-cache[13, 14], which is an integrated architectural approach that turns off power supply to a part of cache sets for reduction of cache leakage. In addition, they provided the gated-Vdd transistor for circuit-level supply-voltage gating. However, the DRI i-cache can be applied only to an L1 instruction cache, which is based on a direct-mapped or low-associative cache with a small area. In order to reduce static power consumption, we have to consider not only an L1 instruction cache but also L1 data and a lower-level (L2, L3) caches, which occupy a larger fraction of the chip area. Furthermore, the DRI i-cache employed a coarse-grain resizing mechanism by changing the number of index bits. Therefore, it is difficult to finely adapt its size to the program requirement. To reduce leakage current, the cache decay mechanism[15] shuts off the power supply to invalidate cache lines in a way. It causes the performance degradation, since sleeping lines do not hold data. The drowsy cache[16] has been proposed to cover the drawback of the cache

decay. It keeps supplying minimum power necessary to hold data to each line in the sleep mode, even though it has a large overhead due to the high complexity of its power-gating circuits. Those three approaches considered only the cache of a single-core processor, and hence a novel mechanism with power-aware control as well as effective partitioning for shared caches is highly desired for CMP design.

In this paper, we propose a power-aware cache partitioning mechanism for the shared cache in order to solve the problems of the performance degradation and the power dissipation in CMPs, focusing on the CMP architecture organized by two cores and a high-associativity shared cache. The proposed cache mechanism, which is an extension of the way-adaptable cache[17], has two functions: a way-allocation function for cache partitioning and a power control function for power-aware computing. The way-allocation function allocates ways of a set-associative cache to each core based on their contributions to the effective performance; each core can use the allocated ways as its private memory space as with the L1 cache. Ways are allocated to each core in proportion to the degree of locality. As a result, the L2 cache is shared among cores without conflicts, and hence the mechanism allows each core to exclusively use an appropriately-sized cache area. The function also helps to find out less needed cache ways, which hardly contribute to the performance. The power control function is applied to the way-adaptable cache to conserve the power dissipation by disabling these less needed ways. We evaluate the performance of the proposed mechanism in terms of three metrics. First, we use the weighted speedup to assess the performance improvement by the proposed way-allocation function. Then, the cache energy consumption is evaluated to show the power saving capability of the proposed power-aware cache partitioning mechanism. Finally, the hardware overhead to implement the control unit of the proposed mechanism is evaluated.

The rest of this paper is organized as follows. We first discuss the locality assessment in program execution in Section 2. In Section 3, we describe details on our proposed cache-partitioning and power-aware cache mechanism under some assumptions. In particular, we discuss our way-allocation and power control functions. Section 4 shows the performance evaluation of the proposed mechanism. Section 5 concludes the paper with some future work.

2 Locality Assessment

We discuss the locality assessment of memory reference for way-allocation and power control functions of the L2 shared cache for CMPs. We first review the stack distance profiling[4] that can assess the temporal locality of reference. Let C_1, C_2, \dots, C_A , and $C_{>A}$ be $A + 1$ counters for an A -way set-associative cache with the LRU replacement policy. Here, C_i counts the number of accesses to the i -th line in the LRU stack. Therefore, the counters C_1 and C_A are used to count the numbers of accessed MRU (Most Recently Used) and LRU lines, respectively. $C_{>A}$ is used to count the number of cache misses.

Figure 1 shows examples of the stack distance profiling. The histogram of C_i obtained by the stack distance profiling can be used to distinguish two kinds

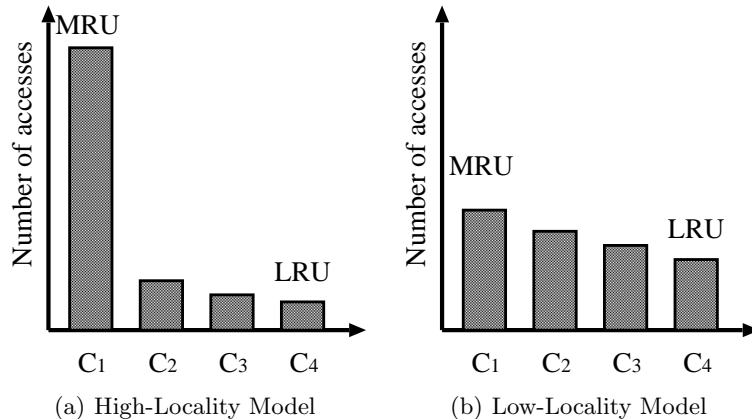


Fig. 1. Stack Distance Profiling

of characteristic cache reference behaviors. In the case of a smaller working set size, the cache accesses tend to concentrate on C_1 as shown in Figure 1(a). In the case of a larger working set, a distribution of cache accesses becomes flatter as shown in Figure 1(b). The locality can be represented by the ratio of an LRU count to an MRU count, using the stack distance profiling.

We define the following metric D to approximately assess the locality of a program[17].

$$D = \frac{LRU\ count}{MRU\ count}. \quad (1)$$

Here, $LRU\ count$ and $MRU\ count$ mean the numbers of LRU and MRU lines referenced in a certain period of cache accesses, respectively. Thus, if a program executed on a core has low-locality, D of the program becomes large. On the other hand, if it has high-locality, D becomes small.

3 Cache Control Mechanism

3.1 Assumptions

We propose a power-aware cache partitioning mechanism under the following assumptions.

- Two cores sharing an L2 unified cache on a chip configure a building block for CMPs. Each core has L1 private data/instruction caches.
- The L2 shared cache is a large, highly-associative on-chip cache, in which the supply voltage to each way can be shut off independently for power control using the power-gating circuits. Each core can access each way exclusively; the cache includes a mechanism that permits a core to access a way. In addition, we introduce an access monitoring unit to the L2 cache. This unit can count the numbers of misses and accesses to MRU and LRU lines.

- Our cache counts the number of accesses to LRU lines for data replacement based on the true-LRU policy.
- Co-scheduled threads on different cores are spawned from different applications. Therefore, they do not share any memory space. So, each core executes a different application in our evaluations. However, our cache can work as a conventional shared cache to access the shared data on the cache among co-scheduled threads spawned from the same application. Our mechanism assumes that the operating system, which manages the thread scheduling, is responsible for switching the partitioning mode and the conventional sharing mode. Both context switch and thread migration between cores are also not considered in this paper.

3.2 Mechanism Overview

We propose a power-aware cache partitioning mechanism under the above assumptions. Figure 2 shows the basic concept of the proposed cache mechanism. Each way is allocated to one of two cores. A virtual partition defined as the boundary between two areas, each allocated to one core, dynamically moves over the L2 cache during execution. Some ways are activated according to locality of memory reference in each area, and the other ways are inactivated for power saving. Each core can access allocated and activated ways only.

Figure 3 shows a control flow graph for an L2 cache shared with two cores. The first step is cache access sampling to obtain statistics used in calculation of D . This step is carried out at fixed intervals, e.g., every 100,000 L2 accesses. Our mechanism has a way-allocation function and a power control function. The former function decides which ways each core can use, and the latter decides how many ways are inactivated for power saving. In the case where both of the two cores have one or more inactivated ways, the way-allocation function is not performed, because way-allocation in such a situation does not decrease conflicts at all but causes a certain overhead.

The cache can randomly select a cache way to be reallocated or inactivated. Before inactivation, the data on the selected way are written back to the main memory for data coherency.

3.3 Way-Allocation Function

The way-allocation function allocates each way of a set-associative cache to a core; each core can use the allocated ways as its private memory space like the L1 cache.

We propose an allocation method that considers the cache reference locality using D . Here, we assume that the number of ways required by a program is proportional to the degree of the locality. After calculating D_i from the number of cache accesses of core i ($i = 0, 1$), the following inequality is used to determine whether the number of ways allocated to core i should be increased or decreased.

$$\text{If } D_j > D_k \quad \begin{cases} Alloc_j + = 1 \\ Alloc_k - = 1. \end{cases} \quad (2)$$

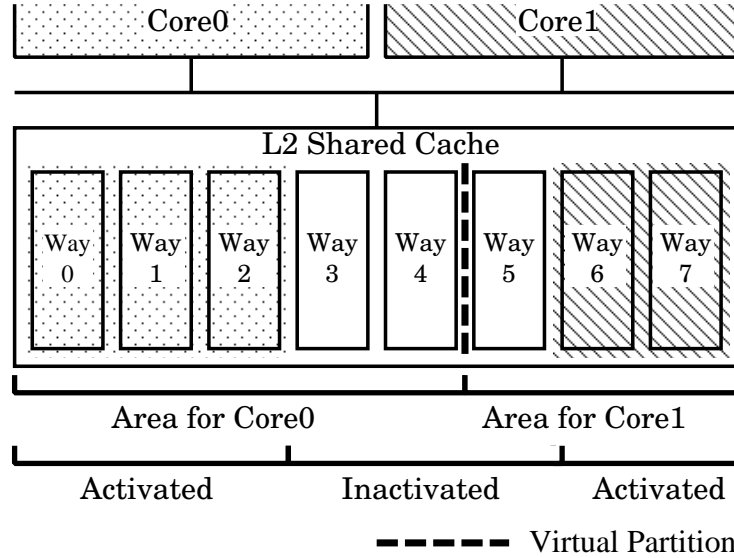


Fig. 2. Basic concept of way-allocation and power control (in the case of 2 cores with an L2 shared 8-way cache)

Here, $Alloc_i$ is the number of ways that are allocated to core i , and satisfies the following conditions.

$$Alloc_{all} = \sum Alloc_i, \quad (3)$$

where $Alloc_{all}$ denotes the cache associativity. If $D_0 = D_1$, way-allocation is not performed.

3.4 Power Control Function

The power control function used in the proposed cache mechanism is based on the way-adaptable cache control[17]. The number of activated ways increases so as to keep D_i between two thresholds to prevent unacceptable performance degradation.

To avoid iterating activation and inactivation of a way, we have to both locally and globally observe the behavior in memory accesses. For locally observing the behavior, D defined by Equation (1) is again employed. On the other hand, for globally observing its local behavior, we adopt an n -bit state machine.

Observation of Local Behavior To quantify the absolute magnitude of local demands for cache resources, D is compared with two thresholds, t_1 and t_2 ($t_1 < t_2$). If D , which is obtained in execution of a program on a core, is larger than t_2 , the program can be considered to have low locality and hence to need

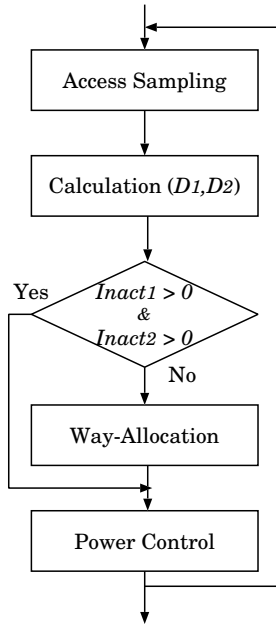


Fig. 3. Control flow graph

many ways. In this situation, our mechanism outputs a signal *inc* (up-sizing request) to increase the number of activated ways. On the other hand, it gives a signal *dec* (down-sizing request) to decrease the number of activated ways if D is smaller than t_1 . If D is between t_1 and t_2 , our mechanism outputs a signal *keep* to keep the current configuration. Our mechanism tends to output *dec* if both t_1 and t_2 are relatively large. On the contrary, it tends to output *inc* if both t_1 and t_2 are relatively small. Thus smaller thresholds make our mechanism performance-oriented, and larger ones make it energy-oriented. As a result, we can adjust the control policy from a performance-oriented configuration to an energy-oriented one.

Observation of Global Behavior Comparison of D with the two thresholds gives the cache resizing requests: *inc*, *dec*, or *keep*. Activation and inactivation of a ways are basically controlled based on cache resizing demands obtained by local behavior observation. However, in the case of highly-irregular and unstable cache accesses, observation of only local behavior leads to iterations of activation and inactivation in a short period. This causes an increase in cache control overheads and cache misses. Thus, the power control should be done conservatively during highly-irregular and unstable situations. To this end, we incorporate an n -bit state machine into the power control function, in order to reflect a global behavior in the cache access, resulting in stable power control. The state machine

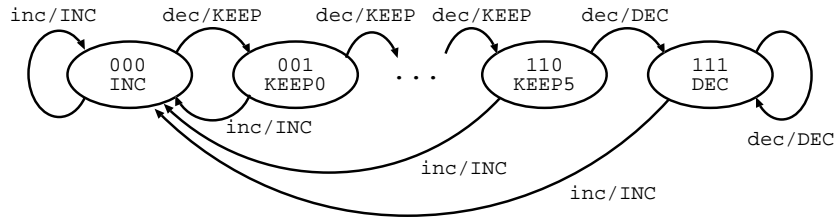


Fig. 4. The 3-bit State Machine

judges that the requests are strong if the same resizing demands continue for a certain period. Only if the request is considered strong, the mechanism resizes the activated area. In addition, we have to be conservative about decreasing the cache size, because it may cause severe performance degradation. Therefore, we use an asymmetric n -bit state machine.

Figure 4 shows a state transition diagram of an asymmetric 3-bit state machine. When *inc* is given to the state machine, it outputs the cache up-sizing control signal *INC* and then always transits to State 000 from any state. However, in the case of *dec* given, the machine works conservatively to generate the down-sizing signal *DEC*. Before moving into State 111 for generating the *DEC* signal, the machine transits to intermediate states from 001 to 110 to judge the continuity of the down-sizing requests. During these states, it outputs *KEEP* to keep the current cache configuration. After continuing *dec* requests, the machine outputs the *DEC* signal for down-sizing and then transits to State 111.

This state machine can prevent responding to temporary disturbances, and further make inactivation conservative to minimize the performance degradation induced by shortage of activated ways. Because the asymmetric machine realizes the cache resizing control so as to react quickly to up-sizing requests and slowly to down-sizing requests, it can minimize performance degradation.

4 Performance Evaluation

4.1 Methodology

We developed a cycle accurate simulator based on the M5 microprocessor architectural simulator tools[18] and the CACTI version 4.2 cache access time, cycle time, area, leakage, and dynamic power models[19] for the architectural function simulation. For the experiments, we examine a CMP of two Alpha-based cores with an L2 shared cache. The parameters used in the simulation are listed in Table 1. We simulate the first 500 million instructions using the reference input set. The sampling span of the proposed cache is 100,000 L2 cache accesses.

We use 15 workloads that consist of combinations of six benchmarks. Table 2 shows the benchmarks selected from the SPEC CPU2006 suite[20] for performance evaluation. Each core runs one independent benchmark program. In

Table 1. Simulation parameters

Parameter	Value
fetch width	8 insts
decode width	8 insts
issue width	8 insts
commit width	8 insts
Inst. queue	64 insts
LSQ size	32 entries
L1 Icache	32kB, 2-way, 32B-line, 1 cycle latency
L1 Dcache	32kB, 2-way, 32B-line, 1 cycle latency
L2 shared cache	1024kB, 32-way, 64B-line, 14 cycle latency
main memory	100 cycle latency
Frequency	1GHz
Technology	70nm
Vdd	0.9V

order to evaluate our proposal fairly, we select various benchmarks based on their cache resource utility graphs[6]. The utility graphs of the benchmark programs are shown in Figure 5. These graphs indicate their performance in IPC as a function of the number of activated ways. Based on the resource utility graphs, we classify the benchmarks into three groups: *high-utility* (High), *saturating utility* (Sat) and *low-utility* (Low). The applications that have *high-utility* benefit from an increase in activated ways (e.g. `gcc`, `bzip2`). These applications have a lower access locality as shown in Figure 1(b). The applications with *saturating utility* have a smaller working set than the applications with *high-utility*; giving more than eight ways dose not significantly improve their performance (e.g. `dealIII`, `sjeng`). The applications that have *low-utility* do not benefit significantly from an increase in activated ways (e.g. `mcf`, `cactusADM`). These applications have a higher access locality as shown in Figure 1(a). We select two applications from each group. Table 2 shows the selected benchmarks and their utilities.

We evaluate our cache with the *weighted speedup* and its energy consumption. The weighted speedup is used as a metric for quantifying the performance of parallel processing, in which multiple applications execute in parallel on different cores. Let $SingleIPC_i$ be the IPC of the i -th application when it is executed on a single core and can exclusively use the entire resource of the CMP, and IPC_i be the IPC of an application when running with another application on the CMP. The weighted speedup is given by:

$$Weighted\ Speedup = \sum \left(\frac{IPC_i}{SingleIPC_i} \right). \quad (4)$$

With the information from M5 and CACTI, the cache energy consumption is calculated as follows:

$$E_{total} = E_d + E_s, \quad (5)$$

Table 2. Benchmark programs

Name	Function	Utility
<code>gcc</code>	C Compiler	High
<code>bzip2</code>	Compression	High
<code>dealIII</code>	Finite Element Analysis	Sat
<code>sjeng</code>	Artificial Intelligence: chess	Sat
<code>mcf</code>	Combinatorial Optimization	Low
<code>cactusADM(cactus)</code>	General Relativity	Low

$$E_d = \int (E_{d_data_array} \times \frac{W_{active}}{W_{total}} \times A_{L2}) dt + E_{d_data_other} + E_{d_tag}, \quad (6)$$

$$E_s = \int (P_{s_data_array} \times \frac{W_{active}}{W_{total}}) dt + E_{s_data_other} + E_{s_tag}, \quad (7)$$

where,

$E_{d_data_array}$ = dynamic energy consumed at bit lines in a data array when all ways are activated,

$P_{s_data_array}$ = static power consumed at word lines when all ways are activated,

W_{active} = the number of activated ways,

W_{total} = the total number of ways in the L2 cache,

A_{L2} = the number of L2 cache accesses,

$E_{d_data_other}$ = the dynamic energy consumption at the other elements in a data array,

E_{d_tag} = the dynamic energy consumption at the others,

$E_{s_data_other}$ = the static energy consumption at the other elements in a data array, and

E_{s_tag} = the static energy consumption at the others.

The total cache energy consumption, E_{total} , is the sum of E_d and E_s that are the dynamic energy consumption for transistor switching and the static energy consumption due to leakage current, respectively. We also estimate E_d and E_s from the product of the array energy consumption and the proportion of the activated area.

4.2 Evaluation of Way-Allocation Function

To evaluate just our way-allocation function, we compare the performance of the way-allocation function with three caches: the utility-based partitioning[6] (*Utility-Based*) which is representative of current partitioning schemes, Half-and-Half which is a static equal partitioning between two cores, and the non-partitioning cache (*CONV*).

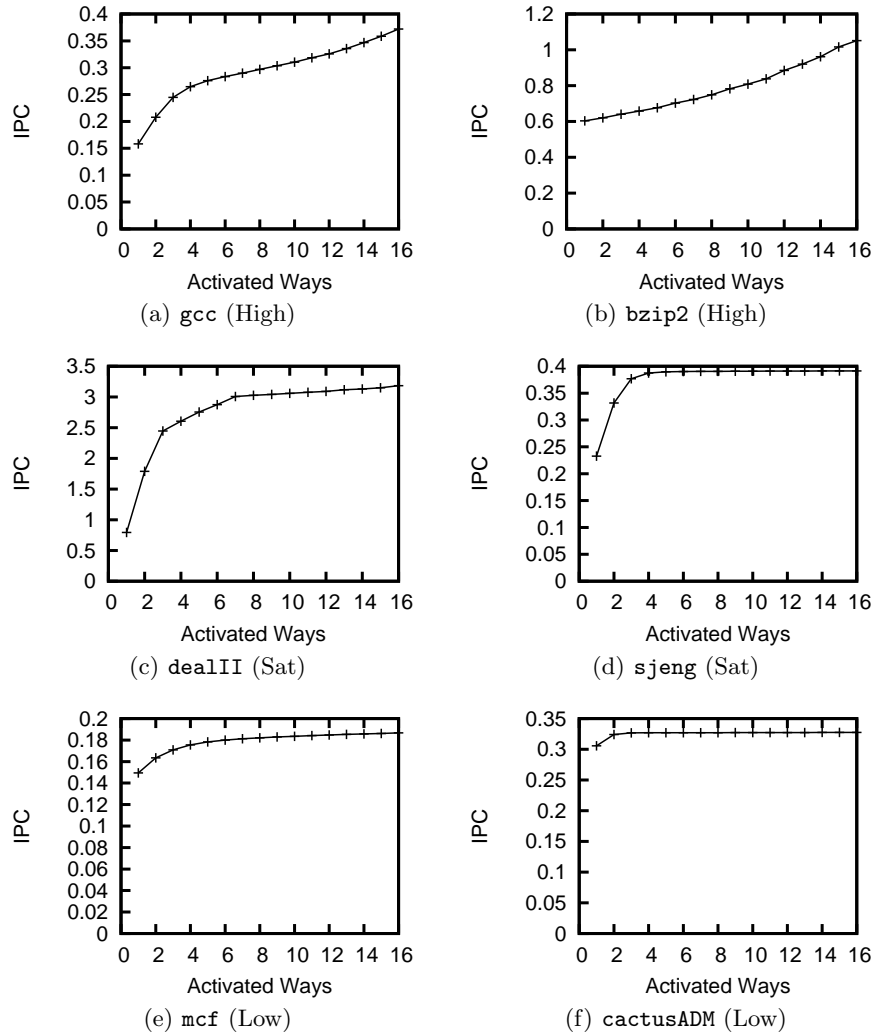


Fig. 5. Activated Ways vs. IPC

Figure 6 shows the evaluation results of the proposed way-allocation function. The horizontal axis in Figure 6 shows the combination of the benchmarks. The vertical axis shows the weighted speedups of CMPs. The bars labeled AVERAGE indicate the geometric means of weighed-speedups of each cache mechanism for individual benchmarks.

The weighted speedup of the proposed way-allocation always exceeds one; it outperforms the conventional cache for every combination listed in Figure 6. It improves by 2% on average. Therefore, it is obvious that the proposed

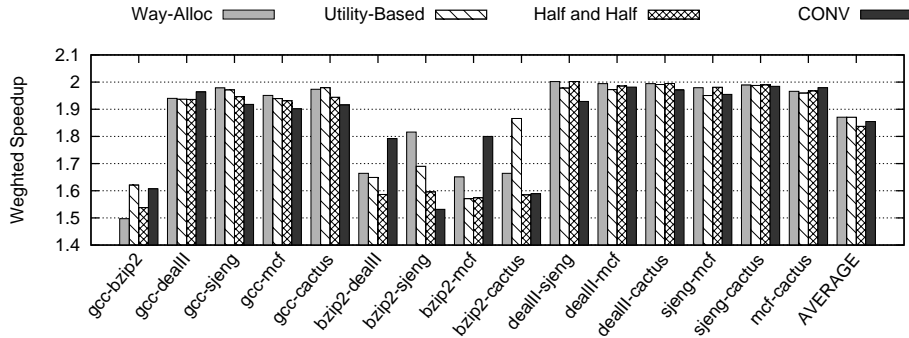


Fig. 6. Performance of Way-Allocation

cache mechanism can reduce cache conflicts, compared to the conventional one. As a result, the way-allocation based on locality assessment can prevent the performance degradation.

The weighted speedups of our cache and *Utility-Based* are the same as or more than that of *Half and Half* in almost all workloads. The static and equal partitioning cannot achieve the performance improvement. Therefore, the results indicate that the cache partitioning with the adaptive mechanism is beneficial. Moreover, the results also indicate that our way-allocation has a performance comparable to the utility-based cache partitioning scheme. In addition, our scheme can save the power consumption, while keeping a certain level performance by adding the power control function.

The weighted speedup obtained by the proposed cache mechanism outperforms the conventional cache for almost all combinations without *bzip2*. Especially, the speedup improves in every combinations with *sjeng*. Therefore, it is obvious that the way-allocation function of our cache is adequate for execution of the applications including the higher cache access locality: saturating or low utility. The validity of our locality assessment model are confirmed by these results.

Every benchmark pair including *bzip2* leads to either significant performance improvement or degradation. As the utility of *bzip2* does not saturate, its performance improves in proportion to the number of activated ways as shown in Figure 5(b). Therefore, it is difficult to define the appropriate position of the virtual partition. A solution to this problem remains as our future work.

4.3 Evaluation of Way-Allocation with Power Control

Performance and Power Consumption The proposed cache mechanism is evaluated in terms of the average number of activated ways and the weighted speedup of CMP. Three different (t_1, t_2) -threshold settings for power control, $(0.1, 0.5)$, $(0.01, 0.05)$ and $(0.001, 0.005)$ are examined. We use an asymmetric

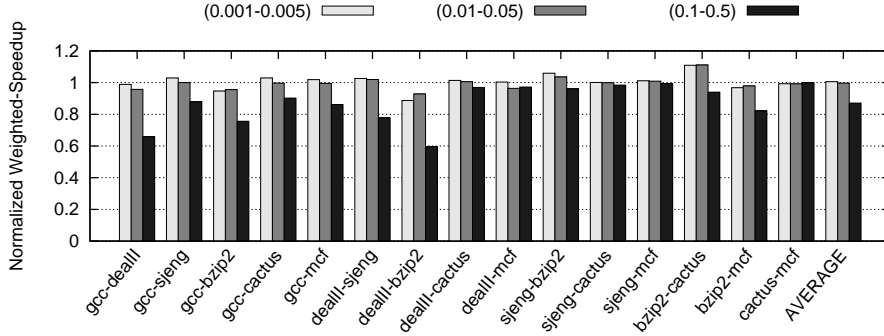


Fig. 7. Effects of t_1 and t_2 on Weighted Speedup

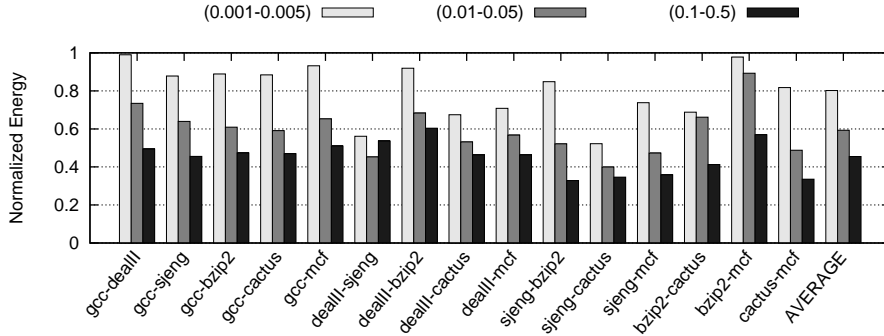


Fig. 8. Effects of t_1 and t_2 on Energy Consumption

3-bit state machine to observe the global behavior in memory accesses. We consider a write-back overhead that is needed to keep data coherency when ways are inactivated, but do not consider an overhead in power-gating for cache ways. Figures 7 and 8 show the weighted speedup and the energy consumption in all the benchmark combinations, respectively. The values of Figures 7 and 8 are normalized by the conventional cache. Figures 7 and 8 indicate that both of the weighted speedup and the energy consumption become their maximum values when the thresholds are (0.001, 0.005) on almost all benchmark combinations. In contrast, when the thresholds are (0.1, 0.5), both of them are their minimum values. When both benchmarks in the combinations have saturating or low utility (e.g. `sjeng-cactus`), the weighted speedup and the energy consumption are not sensitive to the configurations, and achieve high performance and low energy consumption. For example, in the case of a smaller threshold configuration (0.001, 0.005) and the `sjeng-cactus` benchmarks, it can reduce 48% of the energy consumption while keeping the weighted speedup.

On average, the proposed cache can reduce energy consumption by 20% while keeping the performance, when the thresholds are smaller such as (0.001, 0.005). On the other hand, in the case of larger thresholds such as (0.1, 0.5), it can reduce a 55% energy consumption with a performance degradation of 13%. We have confirmed that the values of the thresholds can decide the degree of the control policy between the performance-oriented and the energy-oriented configurations.

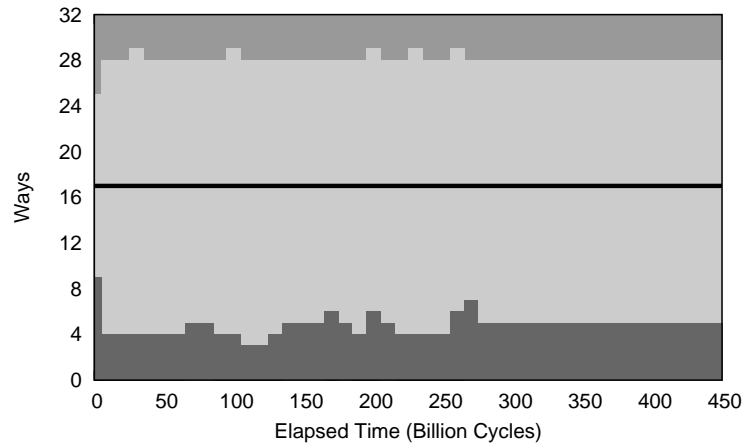
Figure 9 shows the dynamic behavior of the number of allocated and activated ways across time, when `gcc` and `sjeng` are executed in parallel. Most of the time, the allocation function is skipped, because most ways are inactivated for the energy-oriented configuration (0.1, 0.5) as shown in Figure 9(a). The numbers of activated ways are not a significant difference between the cores. On the other hand, with the performance-oriented configuration (0.001, 0.005) as shown in Figure 9(b), almost all of ways are activated, and the way-allocation function effectively works. Especially, the activated ways of `gcc` increases significantly, because `gcc` have the lower access locality than `sjeng`. Therefore, our mechanism appropriately allocates cache ways, and controls the activated cache area based on the access locality.

In the cases of `dealIII-sjeng`, the energy consumption is not minimum when thresholds are energy-oriented. This is because the execution time increases rapidly when the number of activated ways decreases. In fact, Figure 10 shows the energy consumptions of each benchmark with the number of activated ways. All but `dealIII` show monotonic increase with the number of activated ways. However, in the case of combinations including `dealIII`, the minimum energy consumption is achieved when three ways are activated. Therefore, our cache may not decrease the energy consumption on the energy-oriented configuration when the benchmarks like `dealIII` are executed.

In the case of the combinations including `bzip2` (e.g. `gcc-bzip2`), the maximum weighted speedup is achieved at medium thresholds (0.01, 0.05). This is attributed to the fact that the benchmark has the high access utility and the large working set as shown in Figure 5. The behavior analysis of combinations including high utility applications will be discussed in our future work.

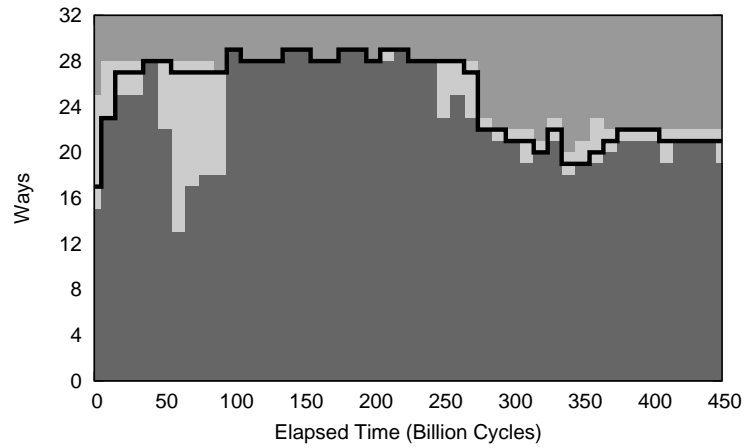
Effects of Sampling Intervals The parameter N , which is the bit width of the sampling counter, defines the maximum sampling interval. Our control mechanism works after 2^N L2 cache accesses. When N is too small, the way-allocation and the power control are performed too often, resulting in an increase in the write-back overheads. On the other hand, the mechanism cannot control the number of ways on demand, when it is too large.

Figure 11 compares the weighted speedups of four interval configurations ($N=8, 12, 16, 20$; each sampling interval is $2^8, 2^{12}, 2^{16}$, or 2^{20} L2 cache accesses.) and the conventional shared cache. We use the performance-oriented configuration $(t1, t2) = (0.001, 0.005)$. The results indicate that decrease in the number of bits of the counter leads to a performance degradation. The results of the 8-bit configuration fall below the conventional cache in all workloads. Moreover, each workload has its own optimal control interval. A majority of workloads



Ways allocated to sjeng Ways allocated to gcc
 Inactivated ways Virtual partition

(a) $(t_1, t_2) = (0.1, 0.5)$



Ways allocated to sjeng Ways allocated to gcc
 Inactivated ways Virtual partition

(b) $(t_1, t_2) = (0.001, 0.005)$

Fig. 9. Elapsed time vs. Allocated ways (gcc-sjeng)

reach their peak performances in the cases of 12-bit or 16-bit intervals. Hence, the control interval is an important parameter that decides the performance of our cache control mechanism.

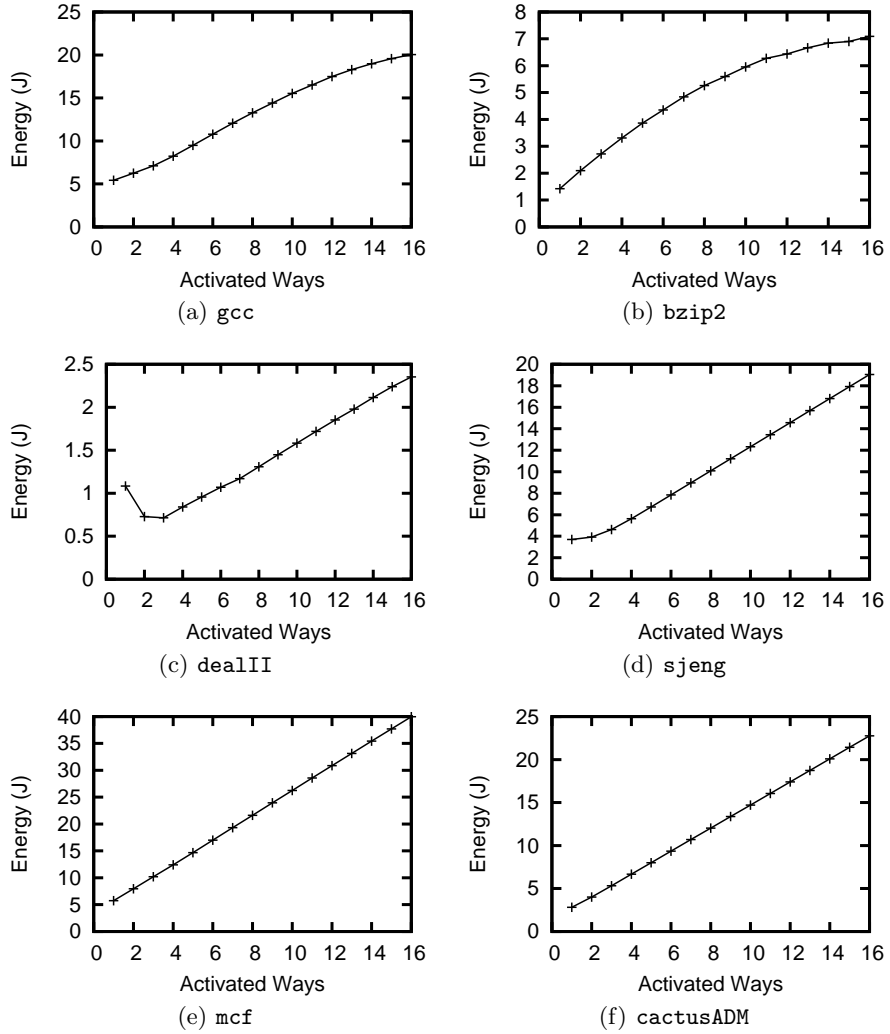


Fig. 10. Activated Ways vs. Energy Consumption

4.4 Evaluation of Hardware Overhead

To evaluate hardware overheads of the proposed cache control mechanism, we design a circuit as shown in Figure 12. The designed circuit consists of two dividers (DIV), three comparators (D_COMP and T_COMP), and two state machines (STATE). Four configurations ($N = 8, 12, 16, 20$) were designed by Rohm $0.18 \mu\text{m}$ CMOS Technology, using Synopsys EDA tools.

Table 3 shows the design results of the four configurations. The delay time is enough small compared with the sampling interval; therefore, the delay of

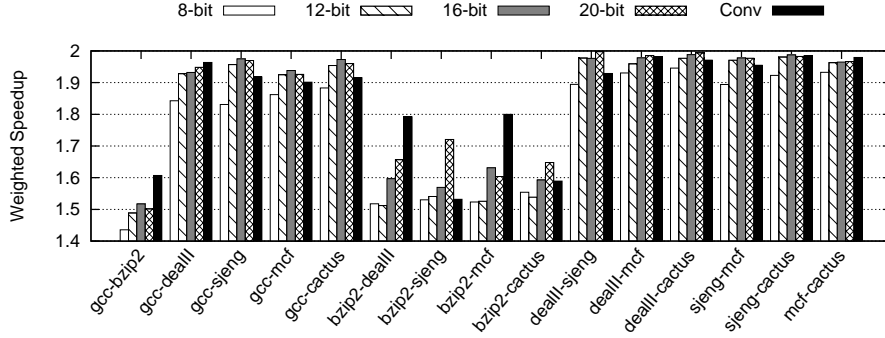


Fig. 11. Effects of Interval Length

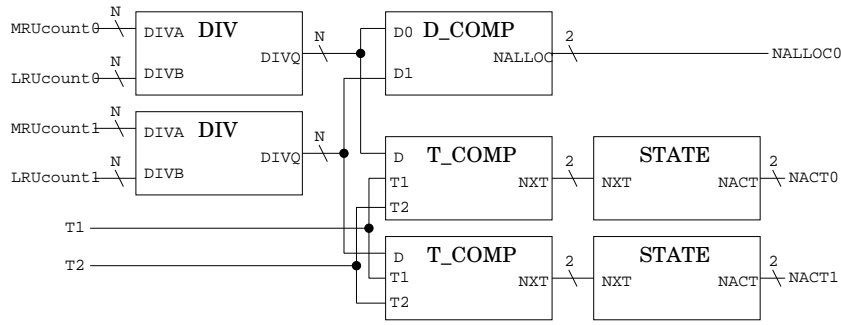


Fig. 12. Cache Control Block Diagram

Table 3. Hardware Overhead

	8-bit	12-bit	16-bit	20-bit
Delay (ns)	21.16	35.38	52.61	84.96
Area(μm^2)	27221	56314	92100	149831

our mechanism hardly affects our way-allocation and power saving capability. Moreover, the circuit areas are extremely small compared with cache area. For comparison, we estimate the area of a 1MB 32-way cache by 0.18 μm CMOS Technology using CACTI [19]. Our proposed mechanism consumes only 0.1% of the cache memory area. The hardware overheads of our cache control mechanism, therefore, have an extremely small effect on the chip design.

5 Conclusions

This paper has proposed a power-aware cache partitioning mechanism for CMPs. We have defined a metric of cache reference locality based on the stack distance

profiling. The mechanism has a way-allocation function and a power control function. The way-allocation function can decide the percentage of cache resources allocated to each core. The power control function decides the cache resources necessary for keeping the current performance. That is, the former is for a relative evaluation by comparing cores' demands for cache resources, and the latter is for an absolute evaluation by estimating the magnitude of the demand by each core.

The evaluation results show that although our cache mechanism and the utility-based scheme are comparable in the cache partitioning performance, our mechanism can save the power consumption. The evaluation results also show that the power control policy of our mechanism can be adjusted from a performance-oriented configuration to an energy-oriented one. The way-allocation cache with a performance-oriented parameter setting can reduce an energy consumption by 20%, while keeping the performance in comparison with a conventional one. On the other hand, the cache with a energy-oriented parameter setting can reduce 55% energy consumption with a performance degradation of 13%. Moreover, we have designed a control mechanism to evaluate hardware overheads. We have shown that our cache control hardware has an extremely small overhead and small effect on the chip design.

The experimental results also indicated that our mechanism does not work in some cases; there was a harmful effect on the performance for lower locality applications. In addition, our cache mechanism can only handle a unit of two cores and an L2 shared cache as a building block for CMPs. Addressing these limitations is the focus of our future work.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. This research was partially supported by Grant-in-Aid for Scientific Research(B), the Ministry of Education, Culture, Sports, Science and Technology, No.18300011. This work is partially supported by VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Synopsys, Inc.

References

1. Wall, D.W.: Limits of instruction-level parallelism. *SIGARCH Comput. Archit. News* **19**(2) (1991) 176–188
2. Nayfeh, B., Olukotun, K.: A single-chip multiprocessor. *Computer* **30**(9) (Sept. 1997) 79–85
3. Suh, G.E., Rudolph, L., Devadas, S.: Dynamic partitioning of shared cache memory. *Journal of Supercomputing* **28**(1) (2004) 7–26
4. Chandra, D., Guo, F., Kim, S., Solihin, Y.: Predicting inter-thread cache contention on a chip multi-processor architecture. In: *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, Washington, DC, USA, IEEE Computer Society (2005) 340–351

5. Kim, S., Chandra, D., Solihin, Y.: Fair cache sharing and partitioning in a chip multiprocessor architecture. In: PACT '04: the 13th International Conference on Parallel Architectures and Compilation Techniques, Washington, DC, USA, IEEE Computer Society (2004) 111–122
6. Qureshi, M.K., Patt, Y.N.: Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In: MICRO 39: the 39th Annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, IEEE Computer Society (2006) 423–432
7. Butts, J.A.; Sohi, G.: A static power model for architects. In: MICRO-33: 33rd Annual IEEE/ACM International Symposium on Microarchitecture. (2000) 191–201
8. Kim, N., Austin, T., Blaauw, D., Mudge, T., Flautner, K., Hu, J., Irwin, M., Kandemir, M., Narayanan, V.: Leakage current: Moore's law meets static power. *Computer* **36**(12) (Dec. 2003) 68–75
9. International technology roadmap for semiconductors: <http://public.itrs.net>
10. Meng, Y., Sherwood, T., Kastner, R.: Exploring the limits of leakage power reduction in caches. *ACM Trans. Archit. Code Optim.* **2**(3) (2005) 221–246
11. Stan, M., Skadron, K.: Power-aware computing. *Computer* **36**(12) (Dec. 2003) 35–38
12. Albonese, D.: Selective cache ways: on-demand cache resource allocation. In: MICRO-32: 32nd Annual International Symposium on Microarchitecture. (Nov. 1999) 248–259
13. Powell, M., Yang, S.H., Falsafi, B., Roy, K., Vijaykumar, N.: Reducing leakage in a high-performance deep-submicron instruction cache. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **9**(1) (Feb 2001) 77–89
14. Powell, M., Yang, S.H., Falsafi, B., Roy, K., Vijaykumar, T.N.: Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories. In: ISLPED '00: the 2000 international symposium on Low power electronics and design, New York, NY, USA, ACM (2000) 90–95
15. Kaxiras, S., Hu, Z., Martonosi, M.: Cache decay: exploiting generational behavior to reduce cache leakage power. In: 28th Annual International Symposium on Computer Architecture. (30 June-4 July 2001) 240–251
16. Flautner, K., Kim, N.S., Martin, S., Blaauw, D., Mudge, T.: Drowsy caches: simple techniques for reducing leakage power. In: 29th Annual International Symposium on Computer Architecture. (25-29 May 2002) 148–157
17. Kobayashi, H., Kotera, I., Takizawa, H.: Locality analysis to control dynamically way-adaptable caches. *SIGARCH Comput. Archit. News* **33**(3) (2005) 25–32
18. Binkert, N., Dreslinski, R., Hsu, L., Lim, K., Saidi, A., Reinhardt, S.: The m5 simulator: Modeling networked systems. *IEEE Micro* **26**(4) (July-Aug. 2006) 52–60
19. Wilton, S., Jouppi, N.: Cacti: an enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits* **31**(5) (May 1996) 677–688
20. The Standard Performance Evaluation Corporation: <http://www.spec.org/>