Trends in High-Performance, Low-Power Cache Memory Architectures

Inoue, Koji
Department of Electonics and Computer Science Fukuoka University

Moshnyaga, Vasily G.
Department of Electonics and Computer Science Fukuoka University

Murakami, Kazuaki
Department of Informatics, Kyushu University

https://doi.org/10.15017/3538

出版情報：IEICE Transactions on Electronics. E85-C (2), pp.304-314, 2002-02. 電子情報通信学会
バージョン：
権利関係：

PAPER

Trends in High-Performance, Low-Power Cache Memory Architectures

Koji INOUE[†], *Nonmember*, **Vasily G. Moshnyaga**[†], *and* **Kazuaki MURAKAMI**[††], *Members*

SUMMARY

One of uncompromising requirements from portable computing is energy efficiency, because that affects directly the battery life. On the other hand, portable computing will target more demanding applications, for example moving pictures, so that higher performance is still required. Cache memories have been employed as one of the most important components of computer systems. In this paper, we briefly survey architectural techniques for high performance, low power cache memories.

key words: cache, low power, high performance, microprocessor, survey

1. Introduction

Cache memories have been playing an important role in bridging the performance gap between high-speed microprocessors and low-speed main memory. Although many cache architectures for improving computer-system performance have been proposed, the processor–memory performance gap is still growing.

One of the most straightforward approaches to improving the memory-system performance is to increase cache size. In order to alleviate the inability of memory systems, the trend is to invest the increasing transistor budget in cache capacity. From power point of view, this approach seems to be useful because the power dissipated for driving external I/O pins can be reduced. However, it increases the power consumed for cache accesses. The power consumption of on-chip caches for StrongARM SA110 occupies 43% of the total chip power [52]. In the 300 MHz bipolar CPU reported by Jouppi et al. [27], 50 % of power is dissipated by caches. Recent growing mobile-market strongly requires not only high performance but also low-power consumption. Therefore, we believe that considering high-performance, low-power cache architectures is a worthwhile work for future processor systems.

There are many levels where we can consider for improving memory-system performance and power consumption: device level, circuit level, architecture level, algorithm level, and so on. In this paper, we briefly survey architectural techniques for high performance, low power memory systems. Before presenting these techniques, we define the performance and power for a memory system in Section 2. Then, high-performance techniques and low-power techniques are introduced in Section 3 and Section 4, respectively. In Section 5, we discuss the effect of high-performance techniques on power consumption. Section 6 gives some concluding remarks.

Manuscript received

[†]Department of Electonics and Computer Science, Fukuoka University, 8–19–1 Nanakuma, Jonan-Ku, Fukuoka 814-0180 Japan.

[††]Department of Informatics, Kyushu University, 6–1 Kasuga-koen, Kasuga, Fukuoka 816-8580 Japan.

2. Performance and Power Consumption

We consider a memory hierarchy which consists of a cache memory implemented by a Static RAM and a main memory implemented by a Dynamic RAM. Note that the lowest level of the memory hierarchy is 2. We can approximate memory-system performance by *average memory-access time (AMAT)* which is the average latency per memory reference [16]. AMAT can be expressed by the following equations:

$$\begin{aligned} AMAT &= T_{Cache} + CMR \times 2 \times T_{MainMemory}, \\ T_{MainMemory} &= T_{DRAMarray} + \frac{LineSize}{BandWidth}, \end{aligned}$$

where CMR is the cache-miss rate. T_{Cache} and $T_{MainMemory}$ are the cache-access time and the main-memory-access time, respectively. On a cache miss, two main-memory accesses take place (one for write-back and one for refill) in the worst case. The main-memory-access time ($T_{MainMemory}$) consists of two factors: the latency for accessing to the DRAM array and that for transferring a cache-line between the cache and the main memory. $LineSize$ and $BandWidth$ are the cache-line size to be replaced and the memory bandwidth between the cache and the main memory, respectively.

In this paper, we consider not power (the amount of energy consumed per unit time) but energy. One of uncompromising requirements of portable computing is energy efficiency, because that affects directly the battery life. *Average memory-access energy* ($AMAE$) is the average energy dissipated per memory reference, and can be expressed by the following equations:

$$\begin{aligned} AMAE &= E_{Cache} + CMR \times 2 \times E_{MainMemory}, \\ E_{MainMemory} &= E_{DRAMarray} + E_{DataTransfer}, \end{aligned}$$

where, E_{Cache} and $E_{MainMemory}$ denote the cache-access energy and the main-memory-access energy. The main-memory-access energy consists of two factors: the energy for accessing to the DRAM array ($E_{DRAMarray}$) and that for transferring a cache-line between the cache and the main memory ($E_{DataTransfer}$).

3. High Performance Techniques

From the performance definition explained in Section 2, it can be understood that there are at least three approaches to improving memory-system performance as follows:

- Reducing the cache-access time (T_{Cache}), and maintaining the cache-miss rate and the miss penalty as can as possible.
- Reducing the cache-miss rate (CMR), and maintaining the cache-access time and the miss penalty as can as possible.
- Reducing the miss penalty ($T_{MainMemory}$), and maintaining the cache-access time and the cache-miss rate as can as possible.

In this section, we introduce techniques to satisfy the above requirements for high-performance memory systems.

3.1 Making Cache Access Faster

The most significant disadvantage of set-associative caches is to suffer from longer access time due to way selection. Since the way selection can be performed after tag-comparison results are available, the critical path becomes long. The key of techniques introduced in this section is to complete the way selection as soon as possible.

3.1.1 Speculative Way Selection: Exploiting Locality

There are two methods to search the hit way which includes the referenced data in set-associative caches: parallel search and sequential search. The parallel search examines all ways in parallel. Thus, the delay for the way selection based on tag comparison makes cache-access time longer. The sequential search examines one by one until it finds the hit way. Therefore, the way-selection overhead can be eliminated if the first examine finds the hit way. In this case, the cache-access time is as fast as direct-mapped caches. However, in the worst case, the sequential search may require the same number of clock cycles as the associativity. Namely, the cache-access time depends on how fast the cache can find the hit way.

Kessler et al. [33] proposed a set-associative *MRU cache*. The MRU cache employs MRU-order-based sequential search. The MRU information is stored in a mapping table. Chang et al. [9] proposed another MRU cache, which is employed in System/370, to improve the access time of parallel-search set-associative caches. Chang et al. reported that where a 128 KB cache has 64 associativity (i.e., 64-way set-associative cache), more than 80 % of the overall memory references hit the MRU region, even if the size of which is only 2 KB (128 KB / 64 way). The MRU information for each set is used to select the hit way before tag comparison is completed. When a cache access is issued, the way designated by the corresponding MRU information is selected. If the cache selects an wrong way, two cycles are required due to accessing to the remaining ways.

3.1.2 Speculative Way Selection: Partial Tag Comparison

Another approach to improving the access time of set-associative caches is to obtain the tag-comparison results as soon as possible. If the control signals for the way selection are available before cache-lines are completely read out, the cache-access-time overhead caused by the way selection can be hidden.

Partial address matching proposed by Liu [39] is one of approaches to reducing the tag-comparison time. Their cache has two memory arrays: MD (for "Main Directory") and PAD (for "Partial Address Directory"). MD contains complete tag information, whereas PAD contains a part of tag bits (e.g., 5 bits). First, tag-comparison results given by the PAD are used for the way selection. The complete tag comparison with the MD tag information is also performed in parallel, but it is for the verification of the partial address tag-comparison. The timing advantage for partial address matching comes from the simpler comparators and fast data read of small number of bits.

Juan et al. [28] proposed the *difference-bit cache* for 2-way set-associative caches. The idea is based on the fact that the two stored tags corresponding to a set have to differ in at least one bit. By using the 1-bit (difference-bit) comparison result, the way selection can be performed. *Diff memory* is employed to record the position and the value of the difference-bit. Note that the difference-bit comparison can be used for the way selection, but not for tag checking. In the case of two-way set-associative caches, complete two tags are read in parallel. After that, one of the tags is selected based on the difference-bit comparison result, and the selected tag is compared with the tag portion in the reference address in order to determine whether the memory reference hits the cache.

3.2 Making Cache-Miss Rate Lower

Memory access behavior varies within and among program executions. However, conventional caches expect that all memory references have high degree of tempo-

ral and spatial locality. Thus, conventional organization has fixed hardware parameters: cache size, cache-line size, associativity, mapping function, replacement policy, and so on. Therefore, it is difficult for the conventional caches to follow the various behavior of memory references. To improve cache-hit rates, many researchers have proposed cache architectures which attempt to adapt dynamically or statically the cache parameters to the varying memory-access behavior.

3.2.1 Making Good Use of Cache Space

Unfortunately, conventional caches have only one mapping function for data placement. The mapping function determines that which set the data designated by a memory address should be placed in. In particular, each set in direct-mapped caches can include only one cache line. Therefore, many data which compete in a set cause a large number of conflict misses. The key of techniques introduced in this section is to use several mapping functions in order to make good use of limited cache space, thereby reducing the conflict misses.

(1) Employing Different Mapping Functions

The direct-mapped *hash-rehash cache* proposed by Agarwal et al. [1] attempts to avoid conflict misses by using two different mapping functions. Conflicting data can be located in the different sets. When a cache access is issued, the first mapping function which is the same function as conventional direct-mapped caches is used to search the first entry. If the first search finds a hit (i.e., first hit), the cache behaves as a direct-mapped cache. Otherwise, the other mapping function is used to search the second entry. *Column-associative cache* proposed by Agarwal et al.[2] has the same configuration as the hash-rehash cache except for a *rehash bit* in each set. The rehash bit inhibits the rehash access in order to avoid secondary thrashing.

The hash-rehash cache and the column-associative cache worsen the cache-miss rates from conventional two-way set-associative caches with LRU replacement strategy. Because the mechanism for hash and rehash operations can not implement a true LRU replacement. Calder et al.[8] proposed *predictive sequential associative cache* which has the *steering bit table* in order to indicate which entry has to be searched first. In addition, the MRU information is used for the complete LRU replacement strategy. The other studies are *sequential multi-column cache* and *parallel multi-column cache* [64]. The detail of this kind of caches has been summarized in [64].

(2) Employing an Adaptive Mapping Function

Adaptive group-associative cache proposed by Peir et al. [49] attempts to intelligently use the cache space. In conventional caches, a number of empty frames, or *holes*, exist in the cache. The authors measured the average percentage of holes in various cache configurations during the execution of a program, and it was observed that between 37.5 % and 42.6 % of the cache are holes. In fact, these holes will be filled by rarely reused data. The idea of the adaptive group-associative cache is to identify the existing holes and allocate the holes to frequently reused data. On a cache miss, frequently reused data to be evicted from the cache is moved into a hole, instead of the main memory. In other words, the cache attempts to optimize the mapping function by detecting the holes at run-time.

3.2.2 Inhibiting Rarely Reused Data from Polluting Cache Space

Since conventional caches attempt to load every data into the cache regardless of its reuse behavior, rarely reused data pollute the limited cache space. Cache bypassing is one of the approaches to solving the pollution problem. The missed data which has poor temporal locality is provided directly from the main memory to the processor without loading into the cache.

Johnson et al. [25] proposed a run-time adaptive cache management technique to improve the cache-hit rates. Their cache employs a *memory address table (MAT)*, in which the memory reference behavior is recorded at run-time. Each entry in the MAT contains a counter in order to identify the amount of temporal locality corresponding to a memory block. When the value of the counter is smaller than a threshold, the reference data in the corresponding memory block bypasses the cache. McFarling [40] proposed *dynamic exclusion replacement* policy in order to reduce the number of conflict misses in direct-mapped instruction caches. The cache presented in the paper monitors reference patterns. When two instructions compete for the same cache line, the dynamic exclusion approach attempts to prohibit loading one instruction into the cache, so that the other instruction can be kept in the cache.

Another approach to avoiding the cache pollution is to secure a part of cache space for frequently reused data. *Scratch-pad memory* has been proposed to realize this kind of memory management. The scratch-pad memory consists of a part of the main-memory space, and is located at level-1 memory hierarchy. Panda et al. [45] presented a technique for exploiting effectively the scratch-pad memory. Chiou et al. [10] proposed *column caching* strategy that allows to restrict the data replacement in way by way. Nakamura et al. [43] proposed a software technique, called *SCIMA: Software Controlled Integrated Memory Architecture*, for high performance computing. In the SCIMA, an on-chip memory space is divided into two portions: cache and on-chip memory. The cache is under the control of hardware as conventional caches, while the data replacement of on-chip memory is controlled by software. Since the cache and

the on-chip memory share the hardware memory structure, software can attempt to change and optimize the ratio of their sizes.

3.2.3 Exploiting Different-Type Memories

Researchers have proposed many cache architectures which consist of several memory modules for improving cache-hit rates. The memory modules are used for different purposes in order to follow the various behavior of memory references.

(1) Keeping and Filtering: Attaching a High-Associative Cache

There are many approaches employing a small high-associative cache. The roles of the attached set-associative cache are 1) to keep frequently reused data at close to the level-1 cache instead of the next-level memory and 2) to filter rarely reused data which pollute the cache. If a data has rich temporal locality, it should not be evicted from the cache. In contrast, a data having poor temporal locality should not be loaded into the cache.

Jouppi [26] proposed the *victim cache* which is a small full-associative cache located between the direct-mapped level-1 cache (main cache) and the next-level memory (main memory). When a cache line in the main cache is evicted, it is moved to the victim cache. In the case of a miss in the main cache that hits in the victim cache, the cache lines are swapped between the main cache and the victim cache.

Theobald et al. [57] discussed the design space of hybrid-access caches (the combination of a direct-mapped main cache and a set-associative cache like the victim cache), and proposed the *half-and-half cache*. The main-cache access can be completed in one cycle, while the associative cache requires two cycles: one for normal access and one for swapping between the main cache and the set-associative cache. If we increase the associative cache size, many memory references can be confine on-chip because of increase in the associative-cache-hit rates. Thus, there is a trade-off between the cache-access time and the cache-hit rate when we consider the cache resource distribution to the direct-mapped region and the set-associative region.

Against to the victim cache and the half-and-half cache, the *annex cache* proposed by John et al. [23] and the *pollution control cache* proposed by Walsh et al. [62] attempt to filter the data to be loaded into the main cache. Both the annex cache and the pollution control cache are small high-associative caches attached to the main cache. On a cache miss, the missed data is loaded into the small associative cache, instead of the main cache. Then the cache lines in the main cache and the small associative cache are swapped when the filled data in the small associative cache is referenced again.

(2) Exploiting Different Types of Locality

Spatial locality can be exploited by increasing the cache-line size. On the other hand, decreasing the cache-line size is a good approach to exploiting the temporal locality, because the total number of entries, or cache lines, in the cache is increased. Unfortunately, conventional caches have a fixed cache-line size, so that it is impossible to satisfy effectively both the above mentioned requirements. The most straightforward approach to solving the problem is to employ two types of caches: one has a small cache-line size and the other has a large cache-line size.

Dual Data Cache proposed by González et al. [14] consists of two memory modules: *spatial cache* and *temporal cache*. These caches have the same organization, but the cache-line sizes are different. The spatial cache has a larger cache-line size, whereas the temporal cache has a smaller cache-line size. The *locality prediction table* in the dual data cache determines where the missed data should be loaded. Each entry in the table corresponds to a recently executed load/store instruction. Against to the dynamic optimization, Sánchez et al. [51] discussed a static analysis of locality for the dual data cache. Park [47] proposed the *co-operative cache* which consists of the *spatial-oriented cache (SOC)* having a larger cache-line size and the *temporal-oriented cache (TOC)* having a smaller cache-line size.

(3) Prohibiting Non-Critical Data from Polluting Cache Space

So far, we have introduced many techniques to improve the cache-hit rates. However, improving the cache-hit rates may not be able to produce an advantage for total system performance. When we consider the total execution time of a program, the most important thing is to reduce the total number of clock cycles required, if we assume that the clock frequency is a constant. Therefore, we need to consider the total number of processor stalls caused by the poor performance of the memory system. Recent processors exploit increased instruction level parallelism (ILP), thereby achieving higher performance.

The cache-hit rate may not be an appropriate metric to evaluate total memory system performance, because it does not include how much each load/store operation affects the total number of processor stalls. The processor stall depends on data dependency in the target program, so that some cache misses which affect the data dependency are more critical than others. In addition, if there are enough instructions which can be issued, the cache miss might not affect ILP. Actually, Srinivasan et al. [55] showed that not all data accesses need to occur immediately if there are enough ready instructions to execute.

Non-Critical Buffer proposed by Fisk et al. [12] is a small associative buffer (for example 16-entry) which

works in parallel with level-1 data cache (main cache). The Non-Critical Buffer is used to prohibit *non-critical* data from polluting the cache space. As a result, the large main cache can be used for *critical* data which give significant damage to the processor performance (i.e., ILP). Two mechanisms to identify the non-critical data at run-time were proposed. One of the mechanisms tracks the processor performance by monitoring issue rate or functional unit usage, and the other mechanism uses the Load/Store Queue (LSQ). Fisk et al. reported that the non-critical buffer can achieve a processor performance improvement even if it worsens the cache-hit rate.

3.2.4 Data Prefetching by Larger Cache-Line Sizes

If we can perform perfect prefetching, the ideal memory system can be realized because all main-memory accesses are overlapped with other computations. Increasing the cache-line size is one of the methods to perform the data prefetching. If memory references have rich spatial locality, larger cache-line sizes give the prefetching effect. Otherwise, a number of conflict misses will take place. For cache-line-size optimization, we have to detect the amount of spatial locality inherent in programs, and then the cache-line size should be modified to an appropriate size.

With hardware support, we can perform the detection and the modification for cache-line-size optimization at run-time. In this method, a mechanism to record memory-reference history will be required. The cache-line size is modified according to the memory-reference history [11], [24], [18], [61], [37]. We can also consider static approaches. The amount of spatial locality is analyzed at compile-time. In this method, loop structures inherent in programs will be exploited. Special instructions are inserted in program codes by compiler in order to change the cache-line size [60], [61].

3.2.5 Optimizing Data Placement

Conflict misses take place when two data compete for a cache location. If we can re-allocate one of the competing data address, the conflict miss can be avoided. Data-placement optimization is a static approach to reducing the conflict misses [58], [44], [17], [29].

3.3 Making Cache-Miss Penalty Smaller

As explained in Section 2, there are at least three approaches to minimizing the miss penalty: 1) improving the DRAM access time, 2) reducing the amount of data to be replaced, and 3) increasing the memory bandwidth. The DRAM access time can be improved by advanced process technology, and is out of scope of this paper.

Cache bypassing explained in Section 3.2.2 reduces the amount of data to be transferred between the cache and the main memory. The idea of Tyson et al. [59] is based on the fact that almost all cache misses are caused by a small number of instructions, called *troublesome instructions*. The authors reported that less than 5 % of the total load instructions are responsible for causing over 99 % of all data-cache misses. When a *troublesome instruction* causes a data-cache miss, the referenced data bypasses the cache, instead of loading into the cache. In this case, only the troublesome instruction is transferred from the main memory to the processor, thereby reducing the memory traffic required.

Improving the memory bandwidth can be achieved by integrating the cache and the main memory into the same chip, or *merged DRAM/logic LSI*. Eliminating the chip boundary between the cache and the main memory solves the I/O-pin bottleneck problem, thereby improving dramatically the memory bandwidth [42], [48], [53].

4. Low-Energy Memory-Access Techniques

From the energy definition explained in Section 2, it can be understood that there are at least three approaches to reducing the average memory-access energy ($AMAE$) as follows:

- Reducing the cache-access energy (E_{Cache}), and maintaining the cache-miss rate and the main-memory-access energy as can as possible.
- Reducing the cache-miss rate (CMR), and maintaining the cache-access energy and the main-memory-access energy as can as possible.
- Reducing the main-memory-access energy ($E_{MainMemory}$), and maintaining the cache-access energy and the cache-miss rate as can as possible.

We can employ the techniques to improve the cache-miss rate introduced in Section 3.2 for the second approach. Therefore, in this section, we focus on the first and the third approaches.

4.1 Reducing Cache-Access Energy

Energy dissipation in CMOS technology circuits is mainly due to charging and discharging gates (and wires). While a cache access is performed, the following energy is dissipated:

$$E_{Cache} = 0.5 \times C \times Vdd^2,$$

where Vdd is the supply voltage as well as the output voltage swing. C is the total capacitive load of all cache components (bit-lines, word-lines, memory cells, and so on). It can be understood that we can reduce the energy dissipation by making a small value of C, or Vdd.

Basically, the energy dissipation for a memory-array access depends on the memory-array size [22]. Here, we refer to a fraction of the cache space activated for the cache access as an *activated-area*. In the case of a conventional organization, the activated-area is equal to the whole cache space. However, if we can divide the memory-array into several subarrays, and if it is possible to activate only one subarray, the activated-area becomes smaller. In other words, the switched load capacitance (C) is reduced, thereby saving the energy.

The key of techniques introduced in this section is to make the activated-area small as can as possible, so that the following processes are required:

1. Module Partitioning: Partition the cache into at least two modules, or attach at least one small cache module. As a result, a small area, which is a candidate of the activated-area, is generated.
2. Selective Activation: Activate only the small area for performing the cache access.

We classify energy reduction techniques for cache memories into two approaches: structural approach and behavioral approach. The structural approach changes the memory organization (the cache is divided), but the cache-access operation is not modified. On the other hand, the behavioral approach attempts to optimize the cache-access procedure for low energy consumption, but the memory organization is maintained (caches originally have a multi-module organization).

4.1.1 Structural Approaches

(1) Horizontal Partitioning

The techniques introduced in this section partition the cache module horizontally. An well known technique for memory-array partitioning is the *word-line partitioning* [50]. In conventional memory arrays, a number of transfer gates are connected to a word-line. The word-line partitioning reduces the total number of memory cells connected to a word-line.

Cache subbanking [56], [30] is a horizontal partitioning scheme for low-energy caches. Usually, a cache line consists of several words (for example 8 words) in order to exploit the spatial locality of memory references. In conventional caches, the referenced data is selected from the cache line. Thus, the remaining contents in the cache line are unused. In the cache subbanking, the data memory is partitioned into subarrays horizontally. Only the subbank including the referenced data is activated. Since the subbank address can be generated directly from the offset-field in the memory-reference address, no cache-access-time overhead occurs.

Region-Based Caching proposed by Lee et al. [38] is another implementation of the horizontal partitioning. The region-based caching exploits the different characteristics of data types, and consists of three cache modules: a small module for stack data, a small module for global data, and a larger main module for others. For example, a 4 KB direct-mapped stack-cache, a 4 KB direct-mapped global-cache, and a 32 KB direct-mapped main-cache are implemented. Which module has to be searched is determined based on the reference address generated by the processor. When the stack-cache or global-cache is activated, the energy dissipated for the cache access can be reduced due to the small value of C, compared with conventional cache organization. Lee et al. reported that about 70 % of memory references hit the stack-cache or the global-cache.

(2) Vertical Partitioning

Bit-line partitioning is another well known technique for reducing energy consumption [50]. Ghose et al. [13] evaluated the effects of the bit-line partitioning on cache energy.

Employing a level-1 cache reduces the energy consumed for memory accesses because of the small activated-area (i.e., accessing not to the large main memory but to the small level-1 cache). Similarly, adding a small level-0 cache between the level-1 cache and the processor can make a significant energy reduction. Su et at. [56] and Kamble et al. [30] evaluated the energy efficiency of *cache-line buffering*, or *block buffering*. The previously accessed cache-line is loaded into the cache-line buffer. When a memory access is issued, the cache-line buffer is searched first. If the memory access hits the cache-line buffer, the referenced data is provided from the cache-line buffer to the processor. In this case, only the cache-line buffer is activated instead of the main cache. When memory references have rich temporal (and spatial) locality, the buffer-hit rate is improved, thereby saving more energy. Ghose et al. [13] proposed the *multiple line buffer* for superscalar processors, in which there are several (for example four) entries. Kin et al. [35] proposed the *filter cache* which is a very small L0-cache located between the processor and the L1-cache. The level-1 cache accesses occur only when filter-cache misses take place. The authors reported that the filter cache reduces 51 % of energy-delay product across a set of multimedia and communication applications compared with a conventional cache organization. Another study for this kind of approach is demonstrated by Bajwa et al. [4], in which a level-0 small cache is employed for reducing the energy consumed for the instruction cache.

The effectiveness of vertical partitioning depends largely on how many the memory references can be concentrated on the small level-0 cache. Bellas et al. [6] proposed a dynamic cache management to allocate the most frequently executed instruction blocks to the small level-0 cache. A branch prediction unit is exploited for detecting the frequently executed blocks.

(3) Horizontal and Vertical Partitioning

Ko et al. [36] proposed the *MDM (multi-divided module) cache* architecture. The cache is divided horizontally and vertically into small modules. Each small module includes own peripheral circuits, so that it can operate as a stand-alone cache. Only a single small module designated by the memory-reference address is activated. When the MDM cache has M independently selectable modules, the average load capacitance becomes almost $1/M$ compared with a non-divided conventional organization.

(4) Static/Dynamic Region Partitioning

Adding a small level-0 cache between the level-1 cache and the processor seems to be an extension of memory hierarchy. Because data replacements between the level-0 and the level-1 cache are required on level-0 cache misses. Another approach to reducing the cache-access energy is to partition the cache module into a small static module and a large dynamic module. Data allocation to the static module is determined based on prior program analysis and that works as the scratch-pad memory explained in Section 3.2.2, whereas the dynamic module behaves as a normal cache. If we can concentrate memory accesses on the small static module, a lot of energy can be reduced due to the small value of C.

Many techniques for data allocation to the static module have been proposed. These techniques are based on profile data from the execution of programs. Panwar et al. [46] proposed *S-cache*. Frequently executed basic-blocks are placed in the S-cache (a small static module). Jump instructions which control the execution flow between the S-cache and the level-1 main cache are inserted in program codes. Bellas et al. [5], [7] proposed the *loop cache (L-cache)* for instruction caches. The compiler lays out the target program for maximizing the number of accesses to the L-cache, and inserts special instructions to identify the boundary between the placed and non-placed code into the L-cache.

Increasing the static-module size improves the static-module-hit rate. However, it increases the energy dissipation for accessing to the static module. Ishihara et al. [22] discussed the trade-off between the size of the main level-1 cache and that of the static module. Although the authors focused on the instruction ROM for embedded systems, we can apply their method to the cache memories. Kawabe et al. [31] presented an implementation example of the static and the dynamic region partitioning.

4.1.2 Behavioral Approaches

All ways in set-associative caches are searched in parallel because the cache-access time is critical. A way consists of one tag-subarray and one data-subarray. Thus, the energy consumed for a tag-subarray access and that for a data-subarray access are consumed in each way. Since only one way has the data referenced by the processor on a cache hit, conventional set-associative caches waste a lot of energy. Some techniques have been proposed for alleviating the negative effect of the set-associative caches by optimizing cache-access behavior.

(1) Selective Way Activation

The activated-area in conventional set-associative caches consists of all of ways. One of the approaches to achieving energy reduction for set-associative caches is to make the activated-area close to a single way including the referenced data.

Hitachi SH microprocessor employs a *phased cache* in order to avoid the unnecessary data-subarray accesses [15]. In the phased cache, tag comparison and cache-line access are performed sequentially. First, the tag comparison is performed at each way without activating the data-subarray. Then, only a single data-subarray which includes the referenced data is accessed if at most one tag matches. Otherwise, a cache-line replacement is performed without any data-subarray access.

Although the phased cache reduces the energy consumed for data-subarray accesses (cache-line accesses), the cache-access time will be increased due to the sequential operation. If we know which way includes the referenced data before starting the cache access (i.e., without performing tag comparison), the unnecessary way-accesses can be eliminated without cache-access-time overhead. Inoue et al. proposed *way-predicting cache* [19] which attempts to avoid the unnecessary way activation for set-associative caches. The basic organization of the way-predicting cache is the same as that of MRU cache [9]. Only one way is speculatively selected by using set-base MRU information before starting normal cache access. The selective way-activation makes a significant energy reduction. If the way prediction is incorrect, the cache then searches the other remaining ways. Kim et al. evaluated the energy efficiency of several selective-way caches [34].

Conventional set-associative caches consist of several memory-subarrays (i.e., ways). Albonesi [3] proposed the *selective cache ways* which allows software to optimize the cache size and the cache associativity. Each way can be enabled/disenabled by the *Cache Way Select Register (CWSR)*. For example, a 32 KB four-way set-associative cache can operate as an 8 KB direct-mapped cache, a 16 KB two-way set-associative cache, or a 32 KB four-way set-associative cache. Namely, the activated-area corresponds to the cache size specified by the CWSR. A software (e.g., operating system) can determine the trade-off between the performance and the energy dissipation by modifying the CWSR.

(2) Omitting Tag Comparison

In conventional caches, tag comparison is performed in every access to determine whether the current access hits the cache. Panwar et al. [46] proposed a conditional tag-comparison scheme which attempts to reduce the total count of tag comparison required. If two successive instructions i and j reside in the same cache-line, the tag comparison for j can be omitted. Another approach to omitting the tag comparison is *history-based tag-comparison cache* proposed in [21]. The history-based tag-comparison cache exploits execution footprints, which are recorded in an extended branch target buffer. The condition for performing the tag comparison is determined based on the execution footprint. In this approach, the tag comparison for instruction j can be omitted, even if instruction i and j are reside in different cache lines.

4.2 Reducing Main-Memory-Access Energy

As introduced in Section 4.1.1(1), cache-access energy can be reduced by employing the cache subbanking. The idea of subbanking comes from that the referenced data is only one word, instead of a whole cache line. Since the offset of the referenced data in the cache line is determined by the memory-reference address, the selective subbank activation can be implemented. However, this kind of technique can not be employed for main memory. Since the cache-line size is fixed, a DRAM-array access with the fixed cache-line size occurs on every main-memory access. The main-memory subbanking can be achieved by reducing the size of data to be replaced between the cache and the main memory. Therefore, the techniques for cache bypassing introduced in Section 3.2.2, reducing the cache-line size introduced in Section 3.2.4, and low memory traffic introduced in Section 3.3 are useful for the main-memory subbanking. Inoue et al. evaluated the energy efficiency of variable cache-line size [20].

5. Effects of High-Performance Techniques on Energy Consumption

The cache performance depends on how fast the memory reference can be performed, while the cache energy depends on how many unnecessary operations can be eliminated. Before, the cache designers had attempted to employ lower associativity for improving T_{Cache}. To avoid the conflict misses, the transistor budget had been invested to increase the cache size. Although decreasing the cache size worsens the cache-miss rate (CMR), this approach is the most straightforward way to reduce both T_{Cache} and E_{Cache} at the same time. Recent low-power microprocessors tend to alleviate the negative effect of the small cache size by employing high associativity [54]. In this section, we discuss effects of the high-performance techniques introduced in Section 3 on energy consumption.

In Section 3.1, we have introduced the techniques to improve the cache-access time T_{Cache}. However, their techniques will not improve the energy efficiency. Since they are based on speculation techniques, the verification processes which also consume the energy are required. On the other hand, almost all techniques introduced in Section 3.2 may be able to produce the energy reductions by improving the cache-miss rate (CMR). That is, the energy consumed for cache misses (not E_{Cache}) is reduced. By concentrating frequently referenced data into the cache, unnecessary cache-line evictions are eliminated, thereby saving the energy. These techniques can be adapted to the high-associative small caches employed in recent low-power microprocessors, because they can compensate the negative effect of small cache sizes. The prefetching techniques explained in Section 3.2.4 will make no energy reductions for cache accesses (E_{Cache}). Prefetching means that a data to be referenced is loaded into the cache in advance. The energy reduction can be done not by performing operations in advance but by eliminating unnecessary operations. However, adapting the cache-line size, or adapting the amount of data transferred between the cache and the main memory, will eliminate the unnecessary data transfer, as explained in Section 3.3. Therefore, it can reduce the energy consumed for cache-line replacements (i.e., $E_{MainMemory}$).

6. Conclusions

In this paper, we have surveyed the techniques for high speed, low power cache memories. Although many cache architectures for improving memory-system performance have been proposed, one of the most important goals in future memory systems is to achieve high performance and low power consumption at the same time.

The high-performance techniques introduced in this paper basically rely on prediction techniques (e.g., the prediction for cache look-up, that for the amount of locality of memory references, and so on). The common low-power microprocessors tend to employ steady approaches for low power, e.g., cache subbanking, cache-line buffering (explained in Section 4.1.1), and so on. However, we believe that accurate predictions for cache accesses have a potential to achieve significant energy reductions. Therefore, we conclude that exploiting prediction techniques not only for high-performance but also for low-power is one of the promising approaches.

Next, we consider the methodology for cache optimization. There are some parameters to be determined in cache design, cache size, cache-line size, associativity, replacement policy, hierarchy, and so on. The techniques introduced in this paper attempt to optimize these parameters dynamically or statically. The

effectiveness of these techniques depends on the characteristics of target programs, which will change intra- and inter-programs. One of disadvantages of the dynamic approach is that the hardware components which consume power works in every clock cycle. Therefore, the performance/power efficiency will be degraded if the dynamic approach is not suitable to the varying characteristics of target programs. On the other hand, although the static approach does not require extra hardware components, the cache designer has to assume a input set for optimizing the cache parameters in advance. That is, one of disadvantages of the static approach is input dependency. It is hard to analyze the memory-reference behavior in detail at compile time. Thus, we believe that a combination of the static and the dynamic approach, or dynamic optimization with software support, is promising for future high-performance, low-power cache memories.

More space in future processor chips will be invested for the cache memories (not only level-1 but also level-2, -3, and so on). Thus, the cache memories will be an important component in processor chips. The followings are other future challenges.

- An effective approach to reducing power dissipation is to reduce the supply voltage [41]. However, low supply voltage will produce a leakage power which is consumed at whole cache space. Challenging to reduce the leakage power consumption is an attractive problem [32][63].

- Combining the structural approach and the behavioral approach is a promising way to achieve high-performance and low-power consumption at the same time. However, the clever cache behavior will complex the cache control logic and its verification. The maturity of verification techniques for cache memories is very important.

- Increasing cache area may result in undesirable effects of reducing manufacturing yield. Although the addition of redundancy circuits (and memory cells) increases manufacturing yield, it also leads to a performance degradation (i.e., cache-access time will become longer). The cache-access time affects directly the memory-access latency. Thus, fault-tolerant techniques suitable for high-speed cache memories are very important.

- In the future social environment in world-wide network systems, one of the most serious problems is the security of information, for example credit-card number, phone number, other personal data, and so on. This kind of information will be memorized and treated via memory systems (disk, main memory, cache memory, and so on). Accordingly, next challenge is to develop high security memory systems for the society in twenty-first century.

Acknowledgments

We thank Hiroto Yasuura who gave us advice on laboratory seminar. This research was supportd in part by the Grant-in-Aid for Scientific Research (A) contracts 09358005, 11308011, 12358002, and 13308015.

References

[1] A. Agarwal, J. Hennesy, and M. Horowitz,"Cache Performance of Operating Systems and Multiprogramming," *ACM Transactions on Computer Systems*, pp. 393–431, Nov. 1988.

[2] A. Agarwal and S. D. Pudar,"Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches," *Proc. of the 20th International Symposium on Computer Architecture*, pp. 179–180, May 1993.

[3] D. H. Albonesi,"Selective Cache Ways: On-Demand Cache Resource Allocation," *Proc. of the International Symposium on Microarchitecture*, pp. 248–259, Nov. 1999.

[4] R. S. Bajwa, M. Hiraki, H. Kojima, D. J. Gorny, K. Nitta, A. Shridhar, K. Seki, and K. Sasaki, "Instruction Buffering to Reduce Power in Processors for Signal Processing," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 5, no. 4, pp.417–424, Dec. 1997.

[5] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis,"Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors," *Proc. of the International Symposium on Low Power Electronics and Design*, pp.70–75, Aug. 1998.

[6] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using dynamic cache management techniques to reduce energy in a high-performance processor," *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, pp.64–69, Aug. 1999.

[7] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis,"Energy and Performance Improvements in Microprocessor Design using a Loop Cache," *Proc. of the 1999 International Conference on Computer Design: VLSI in Computers & Processors*, pp.378–383, Oct. 1999.

[8] B. Calder, D. Grunwald, and J. Emer,"Predictive Sequential Associative Cache," *Proc. of the 2nd International Symposium on High-Performance Computer Architecture*, pp. 244–253, Feb. 1996.

[9] J. H. Chang, H. Chao, and K. So,"Cache Design of A Sub-Micron CMOS System370," *Proc. of the 14th International Symposium on Computer Architecture*, pp. 208–213, June 1987.

[10] D. Chiou, P. Jain, L. Rudolph, and S. Devadas,"Application-Specific Memory Management for Embedded Systems Using Software-Controlled Caches," *Proc. of 37th Design Automation Conference*, pp.416–419, June 2000.

[11] C. Dubnicki and T. J. LeBlanc,"Adjustable Block Size Coherent Caches," *Proc. of the 19th International Symposium on Computer Architecture*, pp. 170–180, May 1992.

[12] B. R. Fisk and R. I. Bahar,"The Non-Critical Buffer: Using Load Latency Tolerance to Improve Data Cache Efficiency," *Proc. of the International Conference on Computer Design: VLSI in Computers & Processors*, pp. 538–545, Oct. 1999.

[13] K. Ghose and M. B. Kamble,"Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line buffers and Bit-Line Segmentation," *Proc. of the 1999 International Symposium on Low Power Electronics and Design*, pp.70–75, Aug. 1999.

[14] A. González, C. Aliagas, and M. Valero,"A Data Cache with

Multiple Caching Strategies Tuned to Different Types of Locality," *Proc. of the International Conference on Supercomputing*, pp. 338–347, July 1995.

[15] A. Hasegawa, I. Kawasaki, K. Yamada, S. Yoshioka, S. Kawasaki, and P. Biswas,"SH3: High Code Density, Low Power," *IEEE Micro*, pp. 11–19, Dec. 1995.

[16] J. L. Hennessy and D. A. Patterson,"Computer Architecture: A Quantitative Approach," *Morgan Kaufmann Publishers, Inc*, 1990.

[17] W. W. Hwu and P. P. Chang,"Achieving High Instruction Cache Performance with an Optimizing Compiler," *Proc. of the 16th International Symposium on Microarchitecture*, pp. 242–251, May 1989.

[18] K. Inoue, K. Kai, and K. Murakami,"Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs," *Proc. of the 5th International Symposium on High-Performance Computer Architecture*, pp. 218–222, Jan. 1999.

[19] K. Inoue, T. Ishihara, and K. Murakami,"Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," *Proc. of the 1999 International Symposium on Low Power Design*, pp. 273–275, Aug. 1999.

[20] K. Inoue, K. Kai, and K. Murakami, "Performance/Energy Efficiency of Variable Line-Size Caches on Intelligent Memory Systems," *Proc. of the 2nd Workshop on Intelligent Memory Systems*, Nov. 2000.

[21] K. Inoue and K. Murakami, "A Low-Power Instruction Cache Architecture Exploiting Program Execution Footprints," *Work-in-progress session in the 7th International Symposium on High-Performance Computer Architecture*, included in CD proc., Jan. 2001.

[22] T. Ishihara and H. Yasuura,"A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors," *Proc. of Design, Automation and Test in Europe Conference 2000*, pp. 617–623, Mar. 2000.

[23] L. K. John and A. Subramanian,"Design and Performance Evaluation of a Cache Assist to Implement Selective Caching," *Proc. of the International Conference on Computer Design: VLSI in Computers & Processors*, pp. 510–518, Oct. 1997.

[24] T. L. Johnson, M. C. Merten, and W. W. Hwu,"Run-time Spatial Locality Detection and Optimization," *Proc. of the 30th International Symposium on Microarchitecture*, pp. 57–64, Dec. 1997.

[25] T., L. Johnson and W. W. Hwu,"Run-time Adaptive Cache Hierarchy Management via Reference Analysis," *Proc. of the 24th International Symposium on Computer Architecture*, pp. 315–326, June 1997.

[26] N. P. Jouppi,"Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. of the 17th International Symposium on Computer Architecture*, pp. 364–373, June 1990.

[27] N. P. Jouppi, P. Boyle, J. Dion, M. J. Doherty, A. Eustace, R. W. Haddad, R. Mayo, S. Menon, L. M. Monier, D. Stark, S. Turrini, J. L. Yang, W. R. Hamburgen, J. S. Fitch, and R. Kao,"A 300-MHz 115-W 32-b Bipolar ECL Microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 11, pp.1152–1166, Nov. 1993.

[28] T. Juan, T. Lang, and J. J. Navarro,"The Difference-bit Cache," *Proc. of the 23th International Symposium on Computer Architecture*, pp. 114–119, May 1996.

[29] J. Kalamatianos and D. R. Kaeli,"Temporal-based Procedure Reordering for Improved Instruction Cache Performance," *Proc. of the 4th International Symposium on High-Performance Computer Architecture*, pp. 244–253, Jan./Feb. 1998.

[30] M. B. Kamble and K. Ghose,"Analytical Energy Dissipation Models For Low Power Caches," *Proc. of the 1997 International Symposium on Low Power Electronics and Design*, pp.143–148, Aug. 1997.

[31] N. Kawabe and K. Usami,"Low Power Technique for On-Chip Memory Using Biased Partitioning and Access Concentration (in japanese)," *IPSJ DA Symposium '00*, pp. 191–196, July 2000.

[32] S. Kaxiras, Z. Hu, G. Narlikar, and R. McLellan,"Cache-line Decay: A Mechanism to Reduce Cache Leakage Power," *Proc. of Workshop on Power-Aware Computer Systems*, Nov. 2000.

[33] R. E. Kessler, R. Jooss, A. Lebeck, and M. D. Hill,"Inexpensive Implementations of Set-Associativity," *Proc. of the 16th International Symposium on Computer Architecture*, pp. 131–139, 1989.

[34] H. S. Kim, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin,"Multiple Access Caches: Energy Implications," *Proc. of the IEEE CS Annual Workshop on VLSI*, Apr. 2000.

[35] J. Kin, M. Gupta, and W. H. Mngione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," *Proc. of the 30th Annual International Symposium on Microarchitecture*, pp.184–193, Dec. 1997.

[36] U. Ko, P. T. Balsara, and A. K. Nanda,"Energy Optimization of Multi-Level Processor Cache Architecture," *Proc. of the 1995 International Symposium on Low Power Electronics and Design*, pp.45–49, Apr. 1999.

[37] S. Kumar and C. Wilkerson,"Exploiting Spatial Locality in Data Caches using Spatial Footprints," *Proc. of the 25th International Symposium on Computer Architecture*, pp. 357–368, June 1998.

[38] H. S. Lee and G. S. Tyson,"Region-Based Caching:An Energy-Delay Efficient Memory Architecture for Embedded Processors," *Proc. of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp.120–127, Nov. 2000.

[39] L. Liu,"Cache Design with Partial Address Matching," *Proc. of the 27th International Symposium on Microarchitecture*, pp. 128–136, Nov./Dec. 1994.

[40] S. McFarling,"Cache Replacement with Dynamic Exclusion," *Proc. of the 19th International Symposium on Computer Architecture*, pp. 191–200, May 1992.

[41] V. G. Moshnyaga,"Reducing Cache Engery Through Dual Voltage Supply," *Proc. of the Asia South Pacific Design Automation Conference*, pp. 302–305, Jan.–Feb. 2001.

[42] K. Murakami, S. Shirakawa, and H. Miyajima,"Parallel Processing RAM Chip with 256Mb DRAM and Quad Processors," *Proc. of the 1997 International Solid-State Circuits Conference*, pp. 228–229, Feb. 1997.

[43] H. Nakamura, M. Kondo, and T. Boku,"Software Controlled Reconfigurable On-Chip Memory for High Performance Computing," *Proc. of the 2nd Workshop on Intelligent Memory Systems*, Nov. 2000.

[44] P. R. Panda, N. D. Dutt, and A. Nicolau,"Memory Organization for Improved Data Cache Performance in embedded Processors," *Proc. of the International Symposium on System Synthesis*, pp. 90–95, Nov. 1996.

[45] P. R. Panda, N. D. Dutt, and A. Nicolau,"Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications," *Proc. of European Design & Test Conference*, Mar. 1997.

[46] R. Panwar and D. Rennels,"Reducing the frequency of tag compares for low power I-cache design," *Proc. of the 1995 International Symposium on Low Power Electronics and Design*, Aug. 1995.

[47] Gi-Ho Park,"Design and Analysis of An Adaptive Memory System for Deep-Submicron and Processor-Memory Integration Technologies," *PhD thesis, Department of Computer*

Science The Graduate School Yonsei University, Dec. 1999.

[48] D. A. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick,"Intelligent RAM(IRAM) Chips that Remember and Compute," *Proc. of the 1997 International Solid-State Circuits Conference*, pp. 224–225, Feb. 1997.

[49] J. K. Peir, Y. Lee, and W. W. Hsu,"Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology," *Proc. of the 8th International Conference on Architectural Support for Programming Language and Operating Systems*, pp. 240–250, Oct. 1998.

[50] T. Sakurai et al.,"Low-Power High-Speed LSI Circuits & Technology (in japanese)," *Realize, Inc*, 1998.

[51] F. J. Sánchez, A. González, and M. Valero,"Static Locality Analysis for Cache Management," *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, Nov. 1997.

[52] S. Santhanam,"StrongARM SA110 -A 160MHz 32b 0.5W CMOS ARM Processor-," *Hot Chips 8: A Symposium on High-Performance Chips*, Aug. 1996.

[53] A. Saulsbury, F. Pong, and A. Nowatzyk,"Missing the Memory Wall: The Case for Processor/Memory Integration," *Proc. of the 23rd Annual International Symposium on Computer Architecture*, pp. 90–101, May 1996.

[54] S. Segars, "Low Power Design Techniques for Microprocessors," *Tutorial Note of the ISSCC*, Feb. 2001.

[55] S. T. Srinivasan and A. R. Lebeck,"Load Latency Tolerance In Dynamically Scheduled Processors," *Proc. of the 31st International Symposium on Microarchitecture*, Nov./Dec. 1998.

[56] C. L. Su and A. M. Despain,"Cache Design Trade-offs for Power and Performance Optimization:A Case Study," *Proc. of the 1995 International Symposium on Low Power Design*, pp. 69–74, Apr. 1995.

[57] K. B. Theobald, H. H. J. Hum, and G. R. Gao,"A Design Framework for Hybrid-Access Caches," *Proc. of the 1st International Symposium on High-Performance Computer Architecture*, pp. 144–153, Jan. 1995.

[58] H. Tomiyama and H. Yasuura,"Code Placement Techniques for Cache Miss Rate Reduction," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 4, pp. 410–429, Oct. 1997.

[59] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun,"A Modified Approach to Data Cache Management," *Proc. of the 28th International Symposium on Microarchitecture*, pp. 93–103, Nov./Dec. 1995.

[60] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji,"Adapting Cache Line Size to Application Behavior," *The International Conference on SuperComputing*, Nov. 1999.

[61] P. V. Vleet, E. Anderson, L. Brown, J. Baer, and A. Karlin,"Pursuing the Performance Potential of Dynamic Cache Line Sizes," *Proc. of the International Conference on Computer Design: VLSI in Computers & Processors*, pp. 528–537, Oct. 1999.

[62] S. J. Walsh and J. A. Board,"Pollution Control Caching," *Proc. of the International Conference on Computer Design: VLSI in Computers & Processors*, pp. 300–306, Oct. 1995.

[63] S. H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar,"An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches," *Proc. of the 7th International Symposium on High-Performance Computer Architecture*, pp. 147–157, Jan. 2001.

[64] C. Zhang, X. Zhang, and Y. Yan,"Two Fast and High-Associativity Cache Schemes," *IEEE Micro*, vol. 17, no. 5, pp. 40–49, Sep.Oct. 1997.

Koji Inoue was born in Fukuoka, Japan in 1971. He received the B.E. and M.E. degrees in computer science from Kyushu Institute of Technology, Japan in 1994 and 1996, respectively. He received the Ph.D. degree in Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan in 2001. In 1999, he joined Halo LSI Design & Technology, Inc., NY, as a circuit designer. He is currently a research instructor of the Department of Electronics Engineering and Computer Science, Fukuoka University. His research interests processor and cache architectures. He is a member of the ACM, the IEEE, the IEEE Computer Society, and the IPSJ (Information Processing Society of Japan).

Vasily G. Moshnyaga received computer engineering degree (Hons) from the Sevastopol Device-Making Institute, Sevastopol, USSR, in 1980 and Ph.D. degree in computer science from the Moscow Aviation Institute in 1987. Till 1992 he was with faculty of Kishinev Politechnical Institute, Kishinev, USSR. From 1992 to 1998 he was a Lecturer of Kyoto University, Japan. Since 1998 he has been with Fukuoka University, Japan, were he is currently a Professor. His research interests are in the areas of VLSI and computer architectures, low power design methodologies and video processing. He is a member of IEEE and Information Processing Society of Japan.

Kazuaki Murakami was born in Kumamoto, Japan in 1960. He received the B.E., M.E., and Ph.D. degrees in computer science and engineering from Kyoto University, Japan in 1982, 1984, and 1994, respectively. From 1984 to 1987, he worked for the Fujitsu Limited, where he was a Computer Architect of the mainframe computers. In 1987, he joined the Department of Information Systems of Kyushu University, Japan, and then was an Associate Professor of the Department. Since 1996, he was an Associate Professor of the Department of Computer Science and Communication Engineering. He is currently a Professor of the Department of Information Science. His original research area was ILP (instruction-level parallel) processors, and in 1987, he proposed one of the first superscalar architectures. His current research focuses on the area of designing and exploiting new computer systems based on advanced VLSI and parallel-processing technologies. In 1994, he started PPRAM (Parallel Processing Random Access Memory) project and is now leading the design and implementation of PPRAM chips. He is also working on another research project, called SmartCore, which aims at developing an application-domain-specific, user-customizable logic/processor core. He is a member of the ACM, the IEEE, the IEEE Computer Society, the IPSJ (Information Processing Society of Japan), and the JSIAM (Japan Society for Industrial and Applied Mathematics).