IPSJ Transactions on System LSI Design Methodology Vol. 2 189–199 (Aug. 2009)

Regular Paper

Single-Cycle-Accessible Two-Level Caches and Compilation Technique for Energy Reducion

Seiichiro Yamaguchi,[†1] Yuriko Ishitobi,[†1]
Tohru Ishihara[†1] and Hiroto Yasuura[†1]

A small L0-cache located between an MPU core and an L1-cache is widely used in embedded processors for reducing the energy consumption of memory subsystems. Since the L0-cache is small, if there is a hit, the energy consumption will be reduced. On the other hand, if there is a miss, at least one extra cycle is needed to access the L1-cache. This degrades the processor performance. Single-cycle-accessible Two-level Cache (STC) architecture proposed in this paper can resolve the problem in the conventional L0-cache based approach. Both a small L0 and a large L1 caches in our STC architecture can be accessed from an MPU core within a single cycle. A compilation technique for effectively utilizing the STC architecture is also presented in this paper. Experiments using several benchmark programs demonstrate that our approach reduces the energy consumption of memory subsystems by 64% in the best case and by 45% on an average without any performance degradation compared to the conventional L0-cache based approach.

1. Introduction

Energy consumption is one of the most important criteria for not only mobile systems but also every computer systems. These systems also require ever-increasing performance of microprocessors for integrating multiple functionalities into a single system. Today's microprocessors used in embedded systems have on-chip caches in order to improve the performance. The on-chip caches also improve the energy efficiency of memory subsystems through decreasing the number of accesses to off-chip memories which involve huge energy dissipation. Therefore, increasing the size of the on-chip cache reduces the energy dissipated for the off-chip accesses. However integrating too large cache on a chip results in an increase of the total energy consumption since the energy consumption of the cache becomes dominant as the size of the on-chip cache increases. For example, ARM920T microprocessor dissipates 44% (25% in instruction cache, 19% in data cache) of the power in its caches [1]. StrongARM SA-110 microprocessor, which specifically targets low power applications, dissipates 43% (27% in instruction cache, 16% in data cache) of the power in its caches [2]. Therefore, a cache architecture which reduces the number of off-chip accesses without increasing the energy consumption of the cache is highly desired. This paper proposes a cache architecture which reduces the energy consumption of on-chip caches and the number of off-chip accesses [3].

The proposed cache architecture, named Single-cycle-accessible Two-level Cache (STC) architecture, has one small and one normal size caches at the same level of memory hierarchy. Only one of the caches is activated at a time. Since both of them can be accessed from an MPU core within a single cycle, there is no performance degradation compared to the conventional level-1 caches. If the small cache is more frequently accessed, the total cache energy consumption can be reduced. Although the two caches are located at the same level of memory hierarchy, they also have a property of hierarchical caches. This paper also presents a compilation technique which selectively places code and data objects in a memory address space so that they are mapped to the small cache preferentially. This concentrates memory accesses to the small cache and, as a result, reduces the total energy consumption of memory subsystems.

The rest of this paper is organized as follows. In section 2, related work is summarized. Section 3 presents a motivational example. The STC architecture is explained in section 4. Section 5 presents a compiler framework and an algorithm for finding code and data placement which minimizes the energy consumption of memory subsystem. Section 6 shows experimental results. Finally, section 7 concludes this paper.

2. Related Work

In the past, many researchers have proposed techniques exploiting a small L0-cache between an MPU core and an L1-cache, e.g., Block Buffering [4),5)], Loop Cache [6)], Filter Cache [7)], and S-Cache [8)]. **Figure 1** shows a memory subsystem of the conventional L0-cache based approach. Since the L0-cache is small, it

†1 Kyushu University

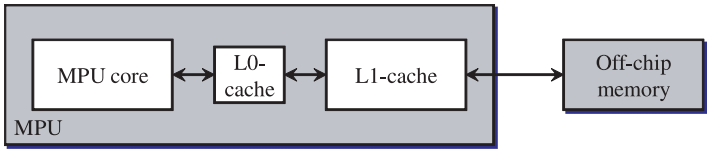


Fig. 1 Memory subsystem of conventional L0-cache based approach.

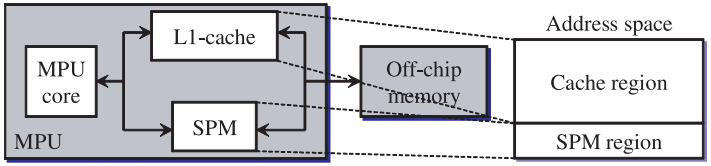


Fig. 2 Memory subsystem of SPM based approach and its address mapping.

consumes less energy per access. Therefore, if there is a hit in the L0-cache, the energy consumption will be reduced. On the other hand, if there is a miss, at least one extra cycle is required to access the L1-cache. This leads a degradation of microprocessor performance.

A software controlled memory called scratchpad memory (SPM) is used with an L1-cache for resolving the problem of the conventional L0-cache based approach. Since the SPM and the L1-cache are located at the same level of memory hierarchy, both of them can be accessed from an MPU core in a single cycle. Only one of them is activated at a time. **Figure 2** shows a memory subsystem of SPM based approach and its address mapping. An address space called SPM region is statically mapped to the SPM. The code and data allocation is done during a system boot and is unchanged after the boot. Therefore, no miss occurs in the SPM. The SPM consumes less energy per access compared to that of the L1-cache since the SPM does not need tag search operations which are need for caches. Therefore, the SPM is more energy efficient if programmers or compilers can optimally allocate code and data to the SPM. Several algorithms for allocating code and data are proposed in Refs. 9)–12). Dynamic SPM management techniques presented in Refs. 13)–16) can further reduce the energy consumption of memory subsystem if overlaying the SPM is done with a small energy overhead. However, if the overlay of the SPM occurs frequently the energy overhead is large. In such a case, a mechanism for overlaying the SPM with a small overhead is preferred.

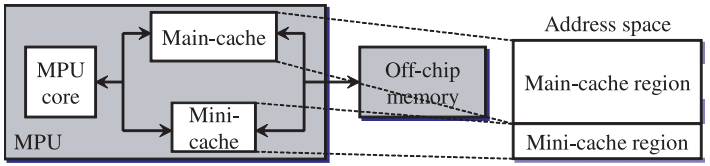


Fig. 3 Memory subsystem of HPC architecture and its address mapping.

Horizontally Partitioned Cache (HPC) architecture exploits two caches, one small and one normal size caches, at the same level of memory hierarchy [17)–19)]. A memory subsystem of the HPC architecture and its address mapping are shown in **Fig. 3**. A Main-cache and a Mini-cache respectively represent a normal size cache and a small cache. The address space is partitioned into two regions, a Main-cache region and a Mini-cache region. These regions are exclusively mapped to the Main-cache and the Mini-cache respectively. After checking several bits of a memory access address, an MPU core accesses one of the caches. The word-line of the cache which is not accessed is not activated. This mechanism is similar to the SPM based approach. However, replacement of the Mini-cache, which corresponds to the overlaying of the SPM, is done automatically in this HPC architecture. The energy consumption of the memory subsystem is also reduced by allocating frequently accessed code and data to the Mini-cache region.

3. Motivation

One of the biggest problems of the HPC architecture and the SPM overlay technique is their ineffective utilization of on-chip memory resources. For example in the HPC architecture, the code and data allocated to the Mini-cache region are mapped to the Mini-cache only. Therefore, those code and data cannot exploit large capacity of the Main-cache. On the other hand, the code and data allocated to the Main-cache region cannot exploit energy efficiency of the Mini-cache.

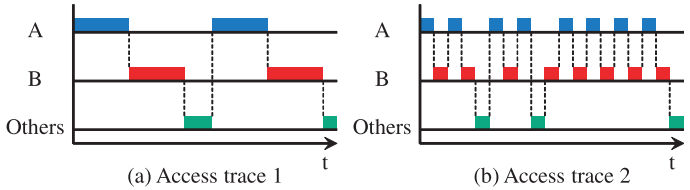
More specific example is described in **Table 1** and **Fig. 4**. In this example, subroutines A, B and the others are alternatively executed on the HPC architecture. If these subroutines are less frequently switched as shown in Fig. 4 (a), code allocation case 1 in Table 1 is better than cases 2 and 3 with respect to

Table 1 Motivational Example in the HPC Architecture.

Code	Access ratio	Size
Subroutine A	40%	1 KB
Subroutine B	40%	1 KB
Others	20%	30 KB

Cache	Size
Main-cache	8 KB
Mini-cache	1 KB

Case	Code allocation	
	Mini-cache region	Main-cache region
1	Subroutine A, Subroutine B	Others
2	Subroutine A	Subroutine B, Others
3	Subroutine B	Subroutine A, Others



(a) Access trace 1 (b) Access trace 2

Fig. 4 Access trace examples.

the energy consumption since the number of Mini-cache misses is not very large. Contrary, if the subroutines are more frequently switched as shown in Fig. 4 (b), the code allocation case 1 results in an increase of the number of Mini-cache misses. For reducing the number of Mini-cache misses, one of the subroutines can be allocated to the Main-cache region as shown in cases 2 and 3 in Table 1. However, this increases the average energy required for accessing caches.

We can resolve the problems in the HPC architecture without increasing the cache size if the frequently accessed code and data allocated to the Mini-cache region can be also mapped to the Main-cache. Our STC architecture allows the code and data allocated to the Mini-cache region to be mapped to both the small and the normal size caches.

4. STC Architecture

4.1 Overview

Our STC architecture employs a small cache called Small-cache as is employed in the L0-cache based approach and the HPC architecture. **Figure 5** shows a memory subsystem of the STC architecture and its address mapping. The

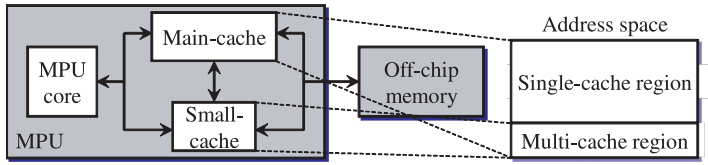


Fig. 5 Memory subsystem of STC architecture and its address mapping.

address space is partitioned into two regions, a Multi-cache region and a Single-cache region. The difference between the HPC architecture and the STC architecture is an address map of the Multi-cache region. In our STC architecture, the Multi-cache region is mapped to both the Small-cache and a normal size cache called Main-cache. The Single-cache region is mapped to the Main-cache only. The code and data allocated to the Multi-cache region exploit the Small-cache preferentially, and utilize the Main-cache depending on a replacement policy.

In the STC architecture, as is similarly done in the HPC architecture, only one of the Main-cache or the Small-cache is accessed at a time. This is done by checking several bits of the memory access address. Since the Single-cache region is mapped to the Main-cache only, only the Main-cache is activated if the target address is included in the Single-cache region. In this case, the Small-cache is inactivated. However, since the Multi-cache region can be mapped to both the Small-cache and the Main-cache, it is impossible to decide which caches should be activated before accessing a tag of the Small-cache if the target address is included in the Multi-cache region. Accessing both the Small-cache and the Main-cache in parallel leads to an increase of the energy consumption of the caches. Accessing the Small-cache first and then accessing the Main-cache if there is a miss in the Small-cache reduces the total energy consumption of the caches. However, this degrades the processor performance. To avoid these issues, the STC architecture employs a mechanism to detect Small-cache hits or misses before activating the caches. This mechanism can be implemented by a D-flip-flop based tag memory.

4.2 D-flip-flop Based Tag Memory

Typically, cache consists of a row decoder, an SRAM tag array for status and tag fields, an SRAM data array for a data field, and sense amplifiers. **Figure 6** shows 2-bits 2-rows SRAM array. Read access to the SRAM array is done in

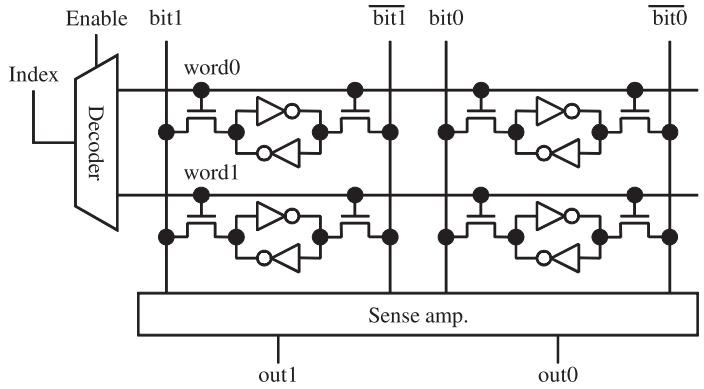


Fig. 6 2-bits 2-rows SRAM array.

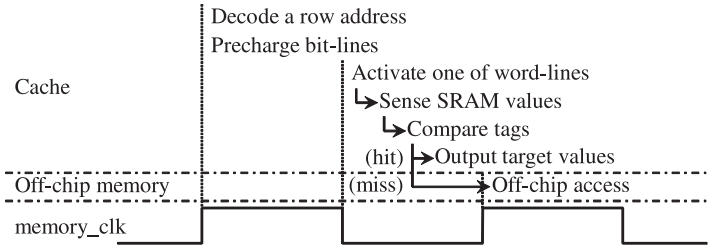


Fig. 7 Timing chart of conventional cache read access using SRAM tag array.

the following steps, 1) decode a row address and precharge bit-lines, 2) activate one of word-lines, and 3) read out data by sense amplifiers. Timing chart of the conventional cache read access using SRAM tag array is shown in **Fig. 7**. The conventional cache decodes a row address and precharges bit-lines at first half of memory clock cycle. At last half of the cycle, the conventional cache operates in the following steps, 1) activates one of word-lines, 2) senses SRAM tag and data values, 3) compares tags, and 4) output target values if there is a hit in the cache.

Meanwhile, our STC architecture employs D-flip-flop tag array in the Small-cache for detecting Small-cache hits or misses before activating word-lines of SRAM array. **Figure 8** shows 2-bits 2-rows D-flip-flop array. A value in the D-

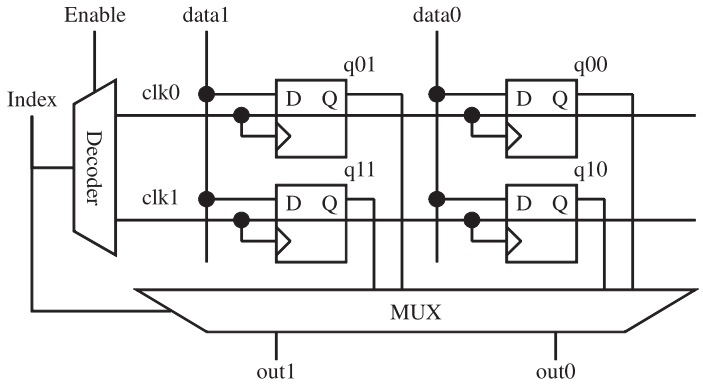


Fig. 8 2-bits 2-rows D-flip-flop array.

flip-flop array can be accessed quickly since, unlike the SRAM array, an output signal of the D-flip-flop is always available without any read out operations. Therefore, read access to the D-flip-flop array can be done in the following one step only, 1) select output signals of D-flip-flops using a multiplexer. Hence, the delay for checking a Small-cache hit or a miss depends on delays required for the multiplexer and a comparator. The tag search operation of the Small-cache has to be completed before activating a word-line of the SRAM array. In other words, the delays of the multiplexer and the comparator have to be shorter than the half of memory clock cycle. Otherwise, an access to the Main-cache needs more than one cycle. Since the tag search operation of the Small-cache in our STC architecture is performed every cycles in parallel with checking the region of access address, the average energy consumption per a Small-cache access is larger than that of the HPC architecture. The energy consumption of the tag search operation of the Small-cache is the energy consumptions of the multiplexer and the comparator. Note that clock-lines are always inactivated during the read access. If the size of the Multi-cache region is N times larger than the Small-cache size, $\lceil log_2(N) \rceil$-bit is needed for implementing the tag memory.

4.3 Architecture and Operation of STC

The memory access address consists of a Region-tag, a Small-tag, an Index, and an Offset fields as shown in **Fig. 9**. The Region-tag field specifies whether

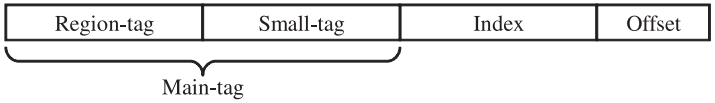


Fig. 9 Memory access address of the STC architecture.

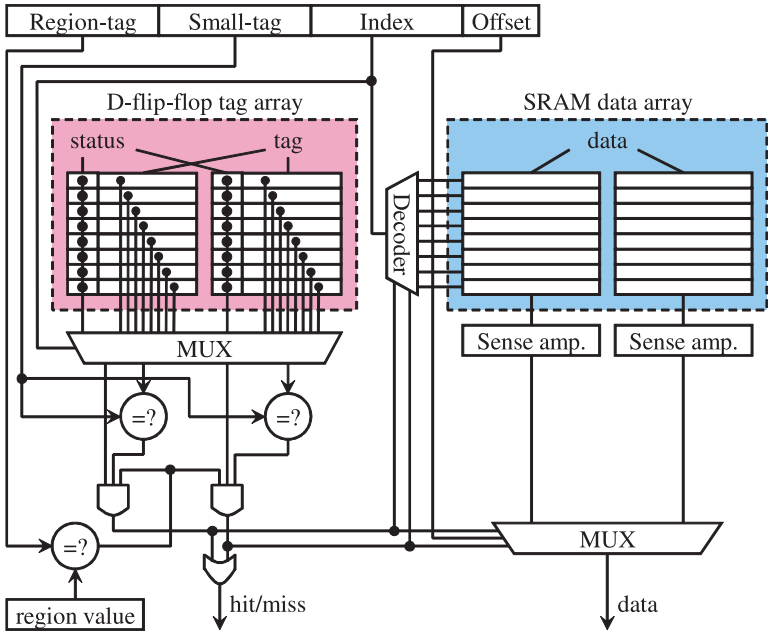


Fig. 10 2-way set-associative Small-cache architecture.

the access address resides in the Single-cache region or the Multi-cache region. The Small-tag field is used for detecting a Small-cache hit or a miss. The Main-tag field which consists of the Region-tag and the Small-tag fields is used for checking a Main-cache hit or a miss. The Index and the Offset fields are used as conventional. **Figure 10** and **Fig. 11** shows architectures of 2-way set-associative Small-cache and 4-way set-associative Main-cache.

At first half of memory clock cycle, Region-tag comparison, Small-tag comparison of D-flip-flop tag array, and bit-lines precharge of SRAM arrays are performed. The rest of cache read procedures depends on the results of Region-tag comparison and Small-tag comparison. There are three types of procedures as follows.

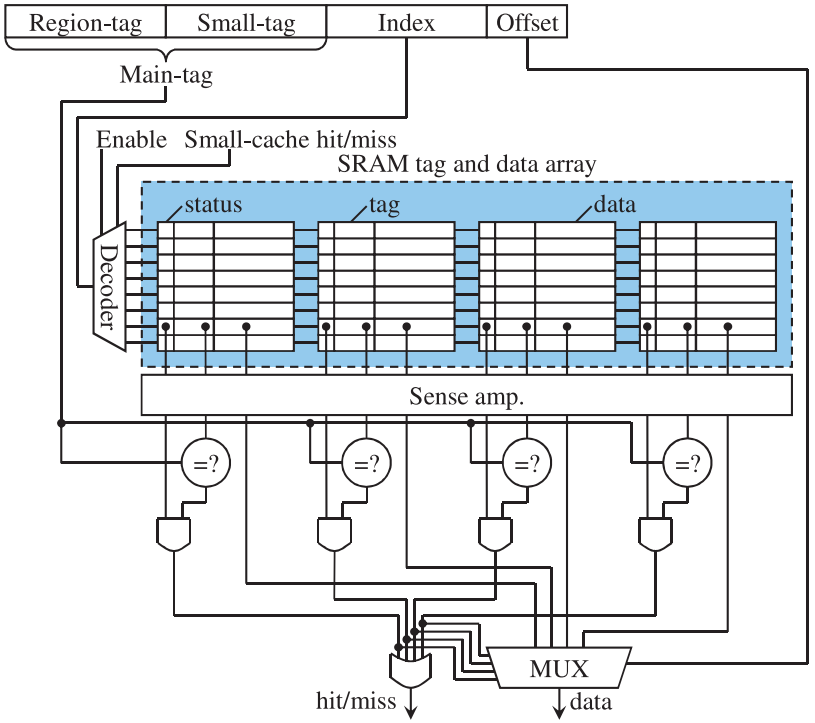


Fig. 11 4-way set-associative Main-cache architecture.

(1) **The access address resides in the Single-cache region.**
In this case, a word-line of the Main-cache is activated at the last half of the cycle as shown in **Fig. 12**. The word-line of the Small-cache is inactivated.

(2) **The access address resides in the Multi-cache region and there is a Small-cache hit.**
In this case, a hit way's word-line of the Small-cache is activated at the last half of the cycle as shown in **Fig. 13**. The word-line of the Main-cache is inactivated.

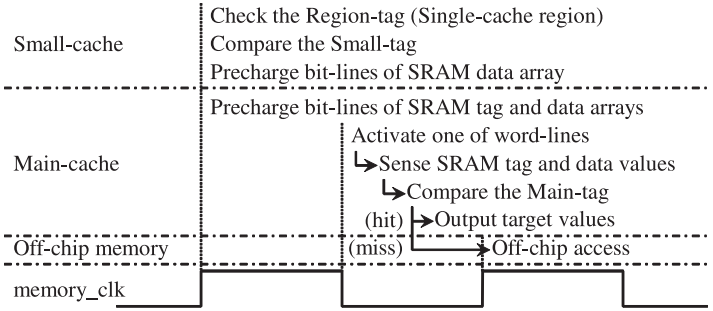


Fig. 12 Timing chart of STC if the access address resides in the Single-cache region.

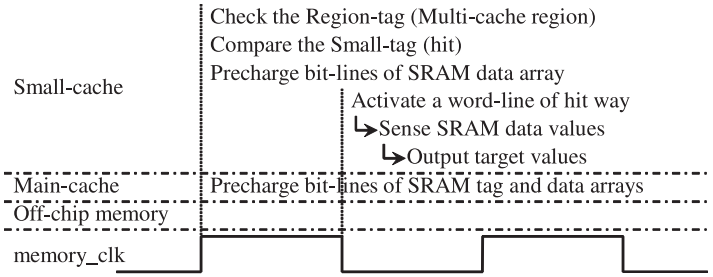


Fig. 13 Timing chart of STC if the access address resides in the Multi-cache region and there is a Small-cache hit.

(3) **The access address resides in the Multi-cache region and there is a Small-cache miss.**

In this case, a word-line of the Main-cache is activated at the last half of the cycle as shown in **Fig. 14**. The word-line of the Small-cache is inactivated. In case (3), if there is a Main-cache hit, operated cache line is stored to a Small-cache replacement buffer. Otherwise, get the target data from an off-chip memory and store it to the Small-cache replacement buffer. And then performs Small-cache replacement flow as shown in the following.

Small-cache replacement flow

SR1 Make backup copy

Make a copy of data in a cache line having the lowest priority in the target set of the Small-cache and store it to a Main-cache replacement buffer. Go to the next step. Note that the lowest priority line can be determined based on replacement policies like the least recently used (LRU) policy or the least frequently used (LFU) policy.

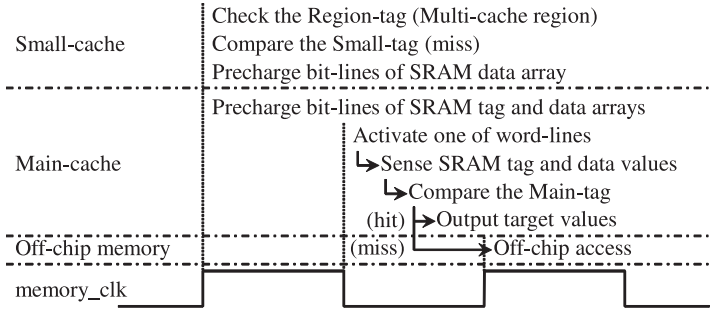


Fig. 14 Timing chart of STC if the access address resides in the Multi-cache region and there is a Small-cache miss.

SR2 Update Small-cache

Move the data in the Small-cache replacement buffer to the lowest priority line of the Small-cache. Update the Small-cache tag and status fields simultaneously. Go to the next step.

SR3 Update Main-cache

Move the data in the Main-cache replacement buffer to the lowest priority line of the Main-cache. Update the Main-cache tag and status fields simultaneously. This step is operated in parallel with the step SR2.

An invalid flag of a cache line is set when the cache line resided in the Small-cache is replaced to the Main-cache or the cache line resided in the Main-cache is copied to the Small-cache. This preserves coherence between the Main-cache and the Small-cache. For always achieving the single cycle access to both the Small-cache and the Main-cache, the Small-cache replacement procedure is not performed if the replacement buffers are full. Otherwise, one extra cycle is needed for waiting for completing the Small-cache replacement procedure before accessing them. This is one of the Small-cache replacement policies. Another policy is that new cache lines for replacement overwrite the replacement buffers if the MPU core accesses a new cache line before completing the Small-cache replacement procedure.

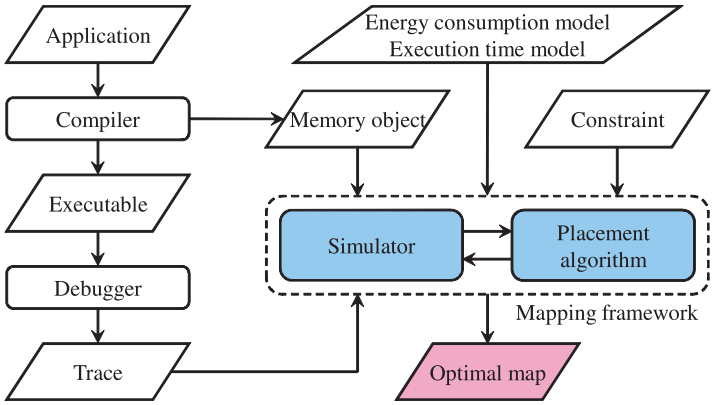


Fig. 15 Compiler framework.

5. Code and Data Placement

5.1 Compiler Framework

Optimization of the energy consumption for memory subsystem can be converted into a memory object placement problem since the energy consumption of memory subsystem strongly depends on code and data placement. **Figure 15** shows our compiler framework for effectively utilizing the STC architecture. A compiler generates an executable file and a list of memory objects with its original mapping address from a target application program. An address trace can be obtained through exploiting a debugger. We input the list of memory objects, the address trace and some constraints to a mapping framework for finding the optimal code and data mapping. A simulator in the mapping framework estimates the energy consumption and the execution time using an energy consumption model and an execution time model formulated in the following subsection. A placement algorithm is also presented in the following subsection. Finally, the mapping framework outputs the optimal memory mapping.

5.2 Energy Consumption and Execution Time Models

Energy consumption of memory subsystem E_{total} and execution time of target application T_{total} are formulated as follows;

$$\begin{aligned} E_{total} = {} & N_{RStag} \cdot E_{RStag} + N_{RSdata} \cdot E_{RSdata} + N_{RM} \cdot E_{RM} \\ & + N_{Roff} \cdot E_{Roff} + N_{WStag} \cdot E_{WStag} + N_{WSdata} \cdot E_{WSdata} \\ & + N_{WM} \cdot E_{WM} + N_{Woff} \cdot E_{Woff} \end{aligned} \quad (1)$$

$$\begin{aligned} T_{total} = {} & N_{RSdata} \cdot T_{RSdata} + N_{RM} \cdot T_{RM} + N_{Roff} \cdot T_{Roff} \\ & + N_{WSdata} \cdot T_{WSdata} + N_{WM} \cdot T_{WM} + N_{Woff} \cdot T_{Woff} \end{aligned} \quad (2)$$

where E_{RStag}, E_{RSdata}, E_{RM}, E_{Roff}, E_{WStag}, E_{WSdata}, E_{WM} and E_{Woff} represent the energy consumption of the Small-cache tag read, Small-cache data read, Main-cache read, off-chip read, Small-cache tag write, Small-cache data write, Main-cache write and off-chip write operations, respectively. N_{RStag}, N_{RSdata}, N_{RM}, N_{Roff}, N_{WStag}, N_{WSdata}, N_{WM} and N_{Woff} denote the event counter for each operation. T_{RSdata}, T_{RM}, T_{Roff}, T_{WSdata}, T_{WM} and T_{Woff} indicate the execution time for each operation. The values of N_x can be obtained from the trace data using the simulator in the mapping framework. The values of E_x and T_x are provided from each memory module.

5.3 Algorithm

As an input of our algorithm shown in **Fig. 16**, a list of memory objects F is used. The memory object includes functions, global variables and constants. An address trace TR of a target application program is also used as an input of our algorithm. The address trace TR is a sequence of instruction and data addresses accessed by the MPU core. An execution time constraint t_{const} is given as a constraint of the problem. The energy consumption of memory subsystem and the execution time of the target application are calculated using TR, the energy consumption model and the execution time model presented in the previous subsection. The main loop of the algorithm starts from the original code where all the memory objects are placed in the Single-cache region. Then the optimal set F' of memory objects which is relocated to the Multi-cache region is found. This is done by choosing a single memory object o from top of F and calculating the energy reduction obtained by relocating o into the Multi-cache region. This calculation is performed for every elements in F. After every elements are evaluated, a memory object o_{best} which reduces the energy consumption the most is selected and is moved form F to F'. This iteration is repeated until the energy consumption stops decreasing. Finally, the algorithm outputs F'.

Code_and_Data_Placement
Input: TR,F,t_{const}
Output: F'
E_{min}=infinity;
repeat
  for$(t = 0; t < |F|; t++)$**do**
    $o = F[t]$;
    tentatively place memory object o to the multi-cache region;
    TR' = modified TR according to the relocation of o;
    Calculate $E_{tmp} = Energy(TR')$;
    Calculate $t_{tmp} = time(TR')$;
    if($E_{tmp} \leq E_{min}$ and $t_{tmp} \leq t_{const}$)
      $E_{min} = E_{tmp}$;
      $o_{best} = o$;
    end if
  end for
  Remove o_{best} from F;
  Append o_{best} into F';
  Update TR according to F and F';
until E_{min} **stops decreasing**
Output F'

Fig. 16 Code and data placement algorithm.

6. Experimental Results

6.1 Experimental Setup

We use several benchmarks in EEMBC DENBench 1.0 suite [20] for our experiments. Benchmarks we chose are shown in **Table 2**. The GNU C compiler and debugger for Toshiba MeP architecture are used for generating lists of memory objects and address traces. The length of the address trace for each single-task benchmark program is 10 million instructions after skipping the initial 1 million instructions. Active code size in Table 2 represents a size of code which are appeared in the address traces obtained. We also use our original address traces, *tasksetA*, *tasksetB*, *tasksetC*, *tasksetC2* and *tasksetC3*, which are generated by combining the address traces of several single-task benchmarks. For example, *tasksetA* consists of *aes*, *cjpeg*, *des*, *djpeg* and *huffde*. We suppose that 1 million instructions of each application program are executed by turns in *tasksetA*, *tasksetB* and *tasksetC*. After executing *huffde*, *tasksetA* executes *aes* continuously. *tasksetC2* executes 4 million instructions of *mpeg2enc* and *mpeg4dec* first, and then 1 million instructions of *rgbyiq* and *rsa* are executed. *tasksetC3* executes 5 million, 1 million, 1 million, 1 million instructions of *mpeg2enc*, *mpeg4dec*, *rgbyiq*, and *rsa* are executed respectively. The length of the address traces for each multi-task benchmark program is 30 million instructions after skipping the initial 5 million instructions.

Table 2 Benchmark Applications.

Benchmarks	Description	Code size	Active code size
aes	AES	55.10 KB	3.47 KB
cjpeg	JPEG Compression	71.34 KB	10.53 KB
des	DES	56.76 KB	7.91 KB
djpeg	JPEG Decompression	75.95 KB	6.50 KB
huffde	Huffman Decoder	49.23 KB	0.88 KB
mpeg2dec	MPEG-2 Decoder	84.76 KB	9.66 KB
mpeg2enc	MPEG-2 Encoder	112.30 KB	33.38 KB
mp3player	MP3 Player	64.64 KB	7.31 KB
mpeg4dec	MPEG-4 Decoder	256.05 KB	27.47 KB
rgbcmyk	RGB to CMYK Converter	49.18 KB	2.03 KB
rgbhpg	High-Pass Gray-Scale Filter	49.45 KB	2.19 KB
rgbyiq	RGB to YIQ Converter	49.38 KB	2.09 KB
rsa	RSA	100.23 KB	10.91 KB
tasksetA	aes, cjpeg, des, djpeg, huffde	308.38 KB	27.34 KB
tasksetB	mpeg2dec, mp3player, rgbcmyk, rgbhpg	248.02 KB	21.19 KB
tasksetC	mpeg2enc, mpeg4dec, rgbyiq, rsa	517.96 KB	62.66 KB
tasksetC2	mpeg2enc, mpeg4dec, rgbyiq, rsa	517.96 KB	52.59 KB
tasksetC3	mpeg2enc, mpeg4dec, rgbyiq, rsa	517.96 KB	33.50 KB

We have developed the code and data mapping framework shown in Fig. 15 which estimates energy consumption of memory subsystem and execution time of the target benchmark program exploiting the energy consumption model and execution time model presented in the previous section. The code and data placement algorithm is embedded in the developed framework. Execution time constraint is supposed as the execution time of the conventional L0-cache based approach.

The energy consumption and execution time for each on-chip memory module are estimated using a commercial 65 nm CMOS technology. We use a model of Micron mobile SDRAM MT48H16M32LFCM-75 IT [21] as an off-chip memory. In our experiments, we also suppose several on-chip memory configurations as

Table 3 On-chip Memory Configurations.

On-chip Memory	Configurations
L1-cache, Main-cache	4 way 8 KB or 2 way 8 KB or 2 way 4 KB, line size:32 byte
L0-cache, Mini-cache Small-cache	2 way 1 KB or Direct map 1 KB or Direct map 512 B line size:32 byte
SPM	1 KB or 512 B

shown in **Table 3**. Target memory subsystems are assumed only instruction memory subsystem and have 8-byte instruction buffer. The size of the Mini-cache region and the Multi-cache region are assumed 4 MB.

6.2 Delay and Area of D-flip-flop Array

The delay for checking a Small-cache hit or a miss depends on delays required for the multiplexer and the comparator. If the size of Multi-cache region is 4 MB, the size of the Small-cache is 1 KB, and cache line size is 32 B, 12-bits 32-rows D-flip-flop array is needed. We designed the multiplexer and the comparator for the D-flip-flop array, and estimated these delays using the commercial 65 nm CMOS technology. The delays of the multiplexer and the comparator are approximately 550 ns and 420 ns, respectively. In this case, our STC architecture performs over 500 MHz. This performance is enough for embedded processors.

The area of 12-bits 32-rows SRAM array is approximately $5.3 \times 10^{-15} m^2$. The area of 12-bits 32-rows D-flip-flop array is approximately $1.0 \times 10^{-13} m^2$, which are about 19 times of the SRAM array.

6.3 Simulation Results

Figure 17 shows the normalized energy consumption results of the STC architecture and those obtained by previous techniques presented in section 2. The energy consumptions are normalized by the energy consumption of the conventional L0-cache based approach for each benchmark. For the HPC and the STC architectures, functions are placed to the large cache region and the small cache region so that the total energy consumption can be minimized. This code placement for the HPC and the STC architecture is performed based on our developed mapping framework. The code placement for the SPM-based approach, we placed funcions order by the number of accesses of the functions.

Our STC architecture reduces the energy consumption of memory subsystem approximately by 64% in the best case and by 45% on an average without any performance degradation compared to the L0-cache based memory subsystem. As one can see from Fig. 17, SPM-based memory subsystem is the most suitable for almost single-task applications. The one of the reasons is that active code size for each benchmark is not so large. On the other hand, our STC architecture is suitable for multi-task applications.

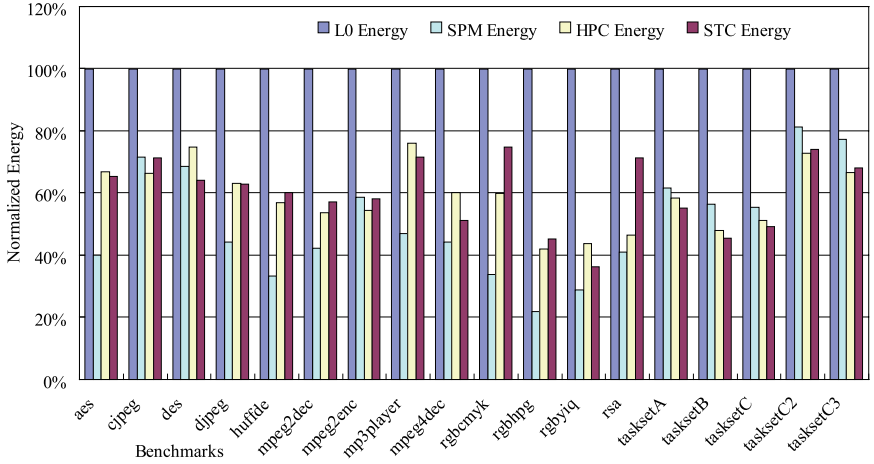


Fig. 17 Normalized energy consumption.

7. Conclusions

Single-cycle-accessible Two-level Cache (STC) architecture and compiler framework for effectively utilizing the STC architecture are proposed. Experiments using EEMBC DENBench 1.0 benchmark suite demonstrate that our STC architecture reduces the energy consumption of the memory subsystems by 64% at the best case and by 45% on an average compared to the conventional L0-cache based approach without any performance degradation.

Acknowledgments This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with STARC, e-Shuttle, Inc., Fujitsu Ltd., Synopsys, Inc. and Cadence Design Systems, Inc. This work is also supported by CREST ULP program of JST and JSPS Grant-in-Aid for Young Scientists (B) (20700049).

References

1) Segars, S.: Low-Power Design Techniques for Microprocessor, *International Solid-State Circuits Conference Tutorial* (2001).
2) Montanaro, J., et al.: A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor, *IEEE Journal of Solid-State Circuits*, Vol.31, No.11, pp.1703–1714 (1996).
3) Yamaguchi, S., Ishihara, T. and Yasuura, H.: A Single Cycle Accessible Two-Level Cache Architecture for Reducing the Energy Consumption of Embedded Systems, *Proc. International SoC Design Conference*, pp.188–191 (2008).
4) Su, C.-L. and Despain, A.M.: Cache Design Trade-offs for Power and Performance Optimization: A Case Study, *Proc. International Symposium on Low Power Design*, pp.63–68 (1995).
5) Kamble, M. B. and Ghose, K.: Analytical Energy Dissipation Models for Low Power Caches, *Proc. International Symposium on Low Power Electronics and Design*, pp.143–148 (1997).
6) Bellas, N., Hajj, I., Polychronopoulos, C. and Stamoulis, G.: Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors, *Proc. International Symposium on Low Power Electronics and Design*, pp.70–75 (1998).
7) Kin, J., Gupta, M. and Mangione-Smith, W. H.: The Filter Cache: An Energy Efficient Memory Structure, *Proc. International Symposium on Microarchitecture*, pp.184–193 (1997).
8) Panwar, R. and Rennels, D.: Reducing the Frequency of Tag Compares for Low Power I-cache Design, *Proc. International Symposium on Low Power Design*, pp.57–62 (1995).
9) Avissar, O., Barua, R. and Stewart, D.: An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems, *ACM Trans. Embedded Computing Systems*, Vol.1, No.1, pp.6–26 (2002).
10) Steinke, S., Wehmeyer, L., Lee, B.-S. and Marwedel, P.: Assigning Program and Data Objects to Scratchpad for Energy Reduction, *Proc. Design, Automation and Test in Europe*, pp.409–415 (2002).
11) Verma, M., Wehmeyer, L. and Marwedel, P.: Cache-Aware Scratchpad Allocation Algorithm, *Proc. Design, Automation and Test in Europe*, Vol.2, pp. 1264–1269 (2004).
12) Ishitobi, Y., Ishihara, T. and Yasuura, H.: Code Placement for Reducing the Energy Consumption of Embedded Processors with Scratchpad and Cache Memories, *Proc. IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia*, pp.13–18 (2007).
13) Francesco, P., Marchal, P., Atienza, D., Benini, L., Catthoor, F. and Mendias, J.M.: An Integrated Hardware/Software Approach for Run-Time Scratchpad Manegement, *Proc. Design Automation Conference*, pp.238–243 (2004).
14) Kandemir, M., Ramanujam, J., Irwin, M.J., Vijaykrishnan, N., Kadayif, I. and Parikh, A.: Dynamic Management of Scratch-Pad Memory Space, *Proc. Design Automation Conference*, pp.690–695 (2001).
15) Udayakumaran, S., Dominguez, A. and Barua, R.: Dynamic allocation for scratch-pad memory using compile-time decisions, *ACM Trans. Embedded Computing Systems*, Vol.5, No.2, pp.472–511 (2006).
16) Verma, M., Wehmeyer, L. and Marwedel, P.: Dynamic Overlay of Scratchpad Memory for Energy Minimization, *Proc. International Conference on Hardware/Software Codesign and System Synthesis*, pp.104–109 (2004).
17) González, A., Aliagas, C. and Valero, M.: A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality, *Proc. International Conference on Supercomputing*, pp.338–347 (1995).
18) Shrivastava, A., Issenin, I. and Dutt, N.: Compilation Techniques for Energy Reduction in Horizontally Partitioned Cache Architectures, *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp.90–96 (2005).
19) Shrivastava, A., Issenin, I. and Dutt, N.: A Compiler-in-the-Loop Framework to Explore Horizontally Partitioned Cache Architectures, *Proc. Asia and South Pacific Design Automation Conference*, pp.328–333 (2008).
20) EEMBC: *DENBench 1.0*, http://www.eembc.org/benchmark/digital_entertainment_sl.php.
21) Micron Technology, Inc.: *MICRON 512 Mb Mobile SDRAM : MT48H16M32LFCM-75 IT Data Sheet*, http://www.micron.com/products/dram/mobilesdram/.

(Received November 17, 2008)
(Revised February 20, 2009)
(Accepted April 13, 2009)
(Released August 14, 2009)

(Recommended by Associate Editor: *Shigetoshi Nakatake*)

Seiichiro Yamaguchi received his B.E. and M.E. degrees in computer science from Kyushu University in 2004 and 2006 respectively. His research interest includes low power SoC design.

Yuriko Ishitobi received her B.E. and M.E. degrees in computer science from Kyushu University in 2007 and 2009 respectively. Her research interest includes low power SoC design.

Tohru Ishihara received his B.S., M.S. and Ph.D. degrees in computer science from Kyushu University in 1995, 1997 and 2000 respectively. From 1997 to 2000, he was a research fellow of the Japan Society for the promotion of science. For the next three years he worked as a research associate in VLSI Design and Education Center, the University of Tokyo. From 2003 to 2005, he worked at Fujitsu Laboratories of America as a research staff of an advanced CAD technology group. In 2005, he joined System LSI Research Center, Kyushu University as an associate professor. His research interests include low power SoC design and hardware/software co-design. He is a member of IPSJ, IEEE and ACM.

Hiroto Yasuura received his B.E., M.E. and Ph.D. degrees in computer science from Kyoto University. He was an associate professor in Kyoto University and moved to Kyushu University in 1991. He was a professor of Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering of Kyushu University. He is currently a trustee/vice president of Kyushu University. His research interests include design methodology for VLSI system, CAD and hardware algorithm. He was a recipient of the Achievement Award from IEICE and awards for persons of merit in Industry-Academia-Government Collaboration/Ministry of Education, Culture, Sports, Science and Technology Award in 2001 and 2007 respectively. He served as general chair of ICCAD and ASP-DAC, and a vice president of IEEE CAS Society. He also served as director of IPSJ/IEICE. He is a fellow of IPSJ.