IPSJ Transactions on System LSI Design Methodology Vol. 2 180–188 (Aug. 2009)

Regular Paper

Partitioning and Allocation of Scratch-Pad Memory in Priority-Based Multi-Task Systems

Hideki Takase,[†1] Hiroyuki Tomiyama[†1]
and Hiroaki Takada[†1]

Energy consumption has become one of the major concerns in modern embedded systems. Recently memory subsystems have become consumed large amount of total energy in the embedded processors. This paper proposes partitioning and allocation approaches of scratch-pad memory in non-preemptive fixed-priority multi-task systems. We propose three approaches (i.e., spatial, temporal, and hybrid approaches) which enable energy efficient usage of the scratch-pad region. These approaches can reduce energy consumption of instruction memory. Each approach is formulated as an integer programming problem that simultaneously determines (1) partitioning of the scratch-pad memory space for the tasks, and (2) allocation of functions to the scratch-pad memory space for each task. Our formulations pay attention to the task periods for the purpose of energy minimization. The experimental results show up to 47% of energy reduction in the instruction memory subsystems can be achieved by the proposed approaches.

1. Introduction

Energy minimization has become one of the primary goals in the design of embedded systems. Reducing energy consumption can extend battery lifetime of portable systems, decrease chip cooling costs, and increase system reliability. These days, cache memory is used not only in general-purpose processors but also in embedded processors. Caches improve performance by exploiting the temporal and spatial access locality in a program. Caches also contribute to energy reduction because of decreased accesses to off-chip memory. However, cache has become one of the most energy-hungry components in embedded processors. For example, the ARM920T processor dissipates 43% of the power in its cache [1)]. Thus, large amounts of studies have addressed cache energy minimization. More recently, scratch-pad memory (SPM) has attracted attention due to its energy efficiency.

A number of techniques have been proposed for efficient usage of SPM in terms of energy consumption and performance, for example in Refs. 2)–11). SPM only consists of decoding circuits, data arrays and output units. Unlike cache, no tag comparison on SPM access is necessary. Therefore, SPM consumes less energy than cache on one access. SPM also contributes real-time predictability because unanticipated access miss like in cache does not occur. On the other hand, programmers or compilers need to decide the allocation of program code or data on SPM since hardware units for allocation management do not exist.

Recently, the scale and the complexity in embedded systems are going to increase. Embedded processors are typically required to execute two or more tasks concurrently. The task scheduling algorithm under the priority is generally employed because high responsiveness is important in real-time systems. However, almost all of previous approaches have focused on the single-task environment. Applying the previous techniques to the multi-task system will result in non-optimal energy savings.

This paper proposes three approaches (i.e., spatial, temporal, and hybrid approaches) which enable energy efficient usage of SPM for multi-task systems. We target on the real-time systems where tasks are scheduled by non-preemptive fixed-priority policy. Each approach is formulated as an integer programming problem. Our approaches solve two problems simultaneously. One is SPM partitioning, which partitions the SPM address space into small regions to be assigned to the tasks. The other is code allocation, which decides, for each task, the program code to be placed in the SPM region. By finding values of decision variables in proposed formulas, optimal SPM partitioning and code allocation are simultaneously determined at static phase. Energy minimization of instruction memory subsystems can be achieved by proposed approaches.

The rest of this paper is organized as follows. In Section 2, a brief survey on related works is provided. Section 3 describes our approaches for SPM partitioning and code allocation in detail. In Section 4, experimental procedure and results are presented. Finally, Section 5 summarizes the contributions of this paper.

†1 Graduate School of Information Science, Nagoya University

2. Related Works and Our Strategy

A considerable amount of research on SPM has been conducted so far for energy or performance optimization. Banakar et al. proposed a technique for selecting an on-chip memory configuration from various size of cache and SPM [2]. They indicated that the energy consumption of SPM-based systems is less than that of cache-based systems by 40% on average. In Ref. 3), a compiler-oriented optimization technique to allocate data variables to SPM for performance improvement was proposed. The authors of Ref. 4) formulated the energy optimal code/data allocation to SPM as a 0/1 integer programming problem based on the size of SPM and the energy consumption on each function. Since 0/1 integer programming model is an NP problem in general, several heuristic algorithms were proposed in Refs. 5), 6). The authors of Ref. 5) proposed a data allocation method for SPM based on the value of total conflict factor (TCF). TCF indicates the possibility of data cache conflicts, and a data variable with high TCF value is allocated to SPM in order to minimize cache conflicts. In Ref. 6), a dynamic programming algorithm for allocating code/data to SPM with a polynomial-time complexity was proposed.

Dynamic SPM management techniques were proposed in Refs. 7)–9). The authors of Ref. 7) introduced a hardware mechanism for efficient overlay of SPM at runtime. In Ref. 8), the customized instructions for transferring program code between SPM and main memory were proposed. Reference 9) proposed hardware/software concerted approach of managing SPM content dynamically. In Ref. 10), the use of SPM in multiprocessor systems is studied. However, these previous techniques are only applicable to single-task systems.

Verma, et al. proposed the SPM region management scheme which can be applied to the multi-task environment [11]. Each task shares the SPM region in a given way for the purpose of energy minimization. However, the proposed approach in Ref. 11) targets on the time-sharing systems whose tasks are scheduled by the round robin manner. In real-time embedded systems, the round robin manner is not generally adopted because high real-time performance is required. The priority-based scheduling policy is employed to satisfy task deadline constraints. In addition, a phased exhaustive search algorithm was conducted in Ref. 11) for searching effective SPM partitioning and code/data allocation. Thereby, the computation time might become huge as the number of input to the algorithm increases.

Our approaches can apply to multi-task environments whose tasks are scheduled based on fixed-priority. This means that energy efficient utilization of SPM can be achieved in the real-time systems. Moreover, the integer programming problems we formulate can obtain the optimal solution at a practical computation time.

3. SPM Partitioning and Code Allocation Approaches

In this section, details of the proposed approaches that can be applied to priority-based multi-task systems are described. We present three approaches: spatial, temporal, and hybrid approaches. The spatial approach statically partitions SPM region to among tasks. In the temporal approach, the entire SPM region is assigned to the running task. The hybrid approach combines the prior two approaches. Each approach is formulated as an integer programming problem which simultaneously determines optimal SPM partitioning and code allocation in terms of the energy efficiency. It is noted that this work focuses on the energy reduction for instruction access and code allocation is performed at a function-level granularity [⋆1].

3.1 Target System Organization

We target an environment where two or more tasks are executed on a single processor. Tasks take dormant, ready and running state as shown in **Fig. 1**. There are neither an inter-task communication nor synchronization, that is, each task executes independently. All tasks are cyclically activated and the periods of tasks are statically decided.

The tasks are scheduled according to the fixed-priority based policy. The highest priority one among the ready state tasks transits to the running state when no task is running. Also, no task preemption is assumed to be occurred.

⋆1 This paper assumes that programs are written in the C language, and term "function" denotes a function of a C program. Note that, in most C programs, the size of a function is much smaller than the size of SPM.

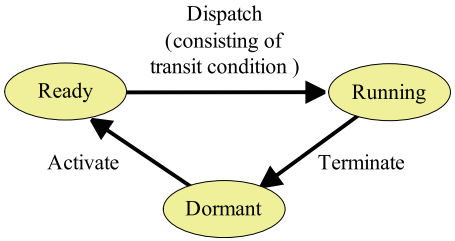


Fig. 1 Task state transition diagram.

Table 1 Definitions of symbols.

Names	Definitions
$task_i$	The i-th task. $1 \leq i \leq M$
$period_i$	The activate interval of $task_i$
$func_{i,j}$	The j-th function in $task_i$. $1 \leq j \leq N$
$fetch_{i,j}$	The total number of executed instructions in $func_{i,j}$ per execution of $task_i$
$size_{i,j}$	The code size of $func_{i,j}$
$Esaving_{i,j}$	The energy reduction if $func_{i,j}$ is allocated to SPM
$Eoverhead_{i,j}$	The energy for transferring $func_{i,j}$ from main memory to SPM
$E_{SPMgain}$	The difference of energy consumption between SPM and cache on read access
E_{Cache_read}	The energy on read access to cache
E_{SPM_read}	The energy on read access to SPM
E_{SPM_write}	The energy on write access to SPM
E_{MM_read}	The energy on read access to main memory
$SPMsize$	The total size of SPM
$hyperperiod$	The least common multiple of all task period
$x_{i,j}, y_{i,j}$	0/1 variables. 1 if $func_{i,j}$ is allocated to SPM, otherwise 0.

3.2 Definitions

Table 1 describes definitions of symbols used in our integer programming formulation. In our work, the values of $fetch_{i,j}$ are obtained by profiling based on instruction-level simulation. However, $fetch_{i,j}$ may be varied depending on input data to the task. In that case, their expected values are obtained by giving a set of representative input data and running the task multiple times at the profiling phase.

3.3 Spatial Approach

The spatial approach partitions the SPM region for the tasks. Each task exclusively uses the given SPM region. **Figure 2** shows the example for partitioning

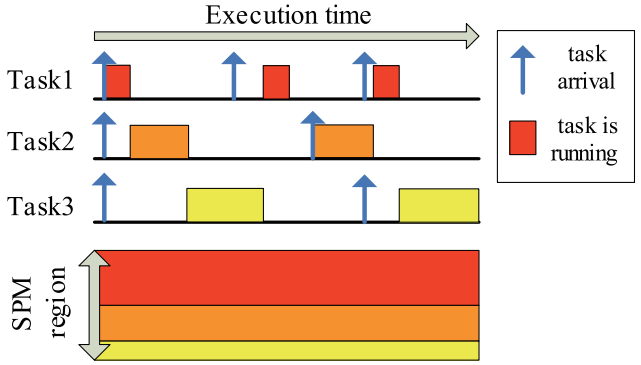


Fig. 2 The spatial approach.

of SPM into three disjoint regions. The amount of SPM region partitioning to the task depends on access frequency in its functions. The SPM partitioning for the tasks and the code allocation to SPM are statically determined. The contents of SPM are not changed at runtime.

In the spatial approach, many codes with high frequently access become to be allocated to SPM by using the task periods as information. As a result, more efficient usage of SPM space is conducted. Since the functions in a short period task raise frequent execution, the given SPM space to task becomes large. For example, Task1 which takes the shortest period in Fig. 2 occupies large SPM region.

The partitioning of SPM region for the tasks and the allocation of function to the SPM for each task in the spatial approach are formulated as a following integer programming problem.

$$\textbf{Maximize} : Esaving = \sum_i \sum_j Esaving_{i,j} \times x_{i,j}$$

$$Esaving_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$\textbf{s.t.} : \sum_i \sum_j size_{i,j} \times x_{i,j} \leq SPMsize$$

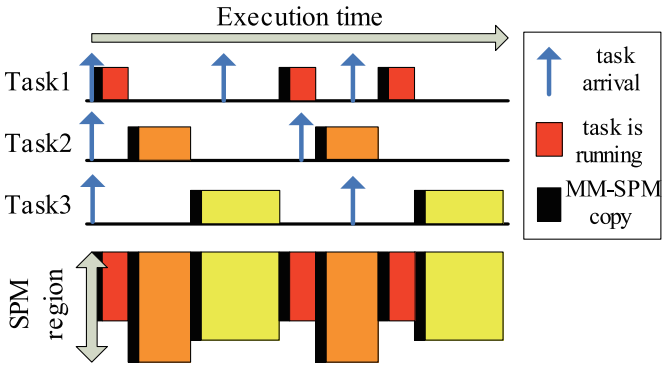


Fig. 3 The temporal approach.

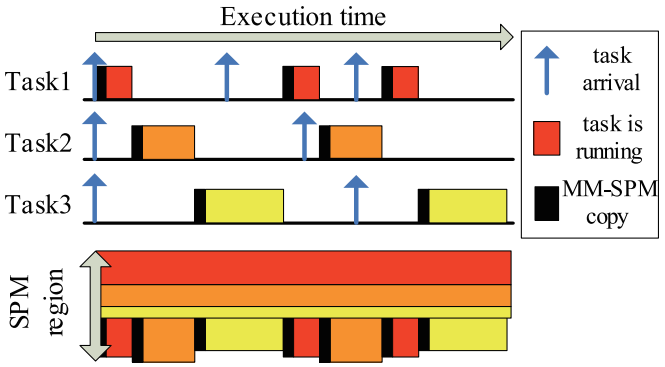


Fig. 4 The hybrid approach.

$fetch_{i,j}$, that is, the total number of executed instructions on each function per one execution of $task_i$, is weighted by the task period $period_i$. By finding $x_{i,j}$'s value, the optimal SPM partitioning and code allocation can be determined at the same time.

3.4 Temporal Approach

As shown in **Fig. 3**, whole SPM address space is assigned to the currently running task. It is necessary to transfer the program code from main memory to SPM whenever a task transits to the running state ('MM-SPM copy' at Fig. 3). The functions allocated to SPM are decided in consideration of the energy consumption on this transferring. The transfer overheads are proportional to the code size of a function. For this reason, a function with not only larger executions but also smaller size is likely to be allocated to SPM space.

An integer programming formulation to decide code allocation in the temporal approach is as follows.

$$\textbf{Maximize}: Esaving = \sum_i \sum_j Esaving_{i,j} \times y_{i,j}$$

$$Esaving_{i,j} = fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$

$$\textbf{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leq SPMsize$$

$Eoverhead_{i,j}$ denotes the energy consumption for transferring $func_{i,j}$. The maximization of E_{saving} is performed under the consideration of $Eoverhead_{i,j}$.

3.5 Hybrid Approach

As shown in **Fig. 4**, the hybrid approach is a mixture of previous approaches. The amount of SPM capacity used by the spatial approach and the temporal one is given by each partitioning so that the total energy reduction E_{saving} becomes the largest. This can achieve more flexible use of the SPM region and more reduction of energy consumption. SPM region where a certain task can use becomes a total of the region partitioned from the spatial region and the region of the temporal approach. An integer programming problem formulated in the hybrid approach is as follows.

$$\textbf{Maximize}: Esaving = Esaving\_spt + Esaving\_tmp$$

$$Esaving\_spt = \sum_i \sum_j Esaving\_spt_{i,j} \times x_{i,j}$$

$$Esaving\_tmp = \sum_i \sum_j Esaving\_tmp_{i,j} \times y_{i,j}$$

$$Esaving\_spt_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$

$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$

$$Esaving\_tmp_{i,j} = (fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}) \times \frac{hyperperiod}{period_i}$$

$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$

$$\textbf{s.t.}: SPMsize\_spt + SPMsize\_tmp \leq SPMsize$$

$$\textbf{s.t.}: \sum_i \sum_j size_{i,j} \times x_{i,j} \leq SPMsize\_spt$$

$$\textbf{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leq SPMsize\_tmp$$

$$\textbf{s.t.}: \forall i, \forall j. \; x_{i,j} + y_{i,j} \leq 1$$

Here, the decision variables are $SPMsize\_spt$, $SPMsize\_tmp$, $x_{i,j}$, and $y_{i,j}$. $SPMsize\_spt$ and $SPMsize\_tmp$ denote the SPM capacity used by the spatial and temporal approaches, respectively. $x_{i,j}$ denotes a 0/1 variable whose value is 1 if $func_{i,j}$ is allocated to the spatial region, and $y_{i,j}$ denotes that of the temporal region. In formulas on the hybrid approach, the partitioning of SPM into the spatial region and the temporal one, the partitioning of the spatial region for the tasks, and the allocation of the function to the SPM region for each task are simultaneously determined by finding these values.

4. Evaluation and Experimental Results

4.1 Experimental Procedure and Tools

Our experimental procedure is depicted in **Fig. 5**. Each task code is cross-compiled into a binary code, which is fed to an instruction-set simulator to generate instruction access traces. We assumed a single-issue ARM processor as our target CPU, and the SimpleScalar/ARM simulator [12] is used for the instruction-level simulation. The instruction memory subsystem consists of cache and SPM as on-chip memory, and SDRAM as off-chip main memory. The cache organization is 16 KB in size and 4-way in associatively. The size of SPM is selected from 4, 8, 12, and 16 KB. Access power for cache and SPM is calculated based on the CACTI 4.2[13], and that for SDRAM is on the Micron System Power Calculator [14]. The amount of energy consumption on one read/write access to each memory is presented in **Table 2**. Also, we do not consider static energy consumptions.

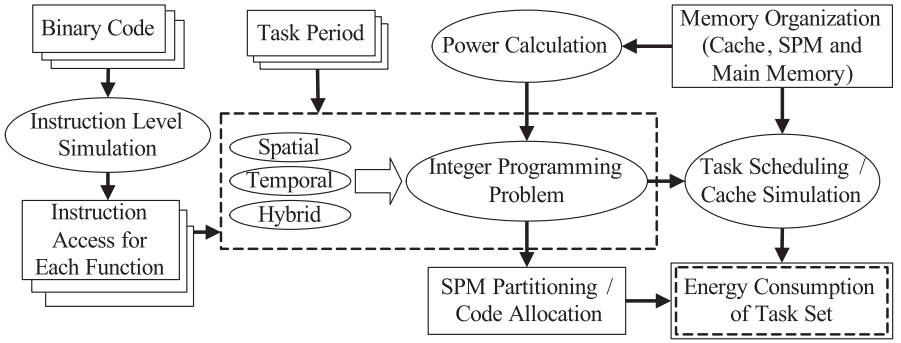


Fig. 5 Experimental procedure.

Table 2 Energy consumption on a memory access.

Memory		Read access [pJ]	Write access [pJ]
cache		5.881	2.279
SPM	4 KB	1.545	0.435
	8 KB	1.617	0.606
	12 KB	2.231	0.638
	16 KB	2.033	0.778
SDRAM		1.002×10^4	—

We selected ten benchmark programs from MiBench suite [15] as task code shown in **Table 3**. The second and third rows denote the number of functions and total code size in bytes within each task, respectively. The fourth row denotes the total number of executed instructions per one execution of the task. Note that only the functions which is actually executed are included in Table 3. For each task, the same input data is used for both profiling and evaluation phases. The proposed approaches are evaluated on a synthetic task set as presented in **Table 4**.

In our experiments, the periods of tasks are set according to the following two

Table 3 Task features.

Name	# of func.	Code size [bytes]	# of executed instr. [$\times 10^4$]
bitcnts	123	44476	4966
bf	103	17912	5240
cjpeg	248	52008	2811
crc	112	33092	6166
dijkstra	112	50204	6492
ispell	162	60044	837
qsort	113	48312	4360
rawcaudio	82	28616	3769
sha	113	33972	1354
tiff2rgba	882	42452	3694

Table 4 Task sets.

	# of task	Tasks included
TasksetA	2	bf, tiff2rgba
TasksetB	4	cjpeg, crc, qsort, tiff2rgba
TasksetC	6	bitcnts, cjpeg, dijkstra, ispell, rawcaudio, sha
TasksetD	8	bitcnts, bf, crc, dijkstra, ispell, qsort, rawcaudio, sha
TasksetE	10	bitcnts, bf, cjpeg, crc, dijkstra, ispell, qsort, rawcaudio, sha, tiff2rgba

rules; The periods are proportional to their execution times; The total CPU utilization is 60%. The reason for the latter rule is to guarantee the schedulability of the tasks. It is noted that the total CPU utilization does not affect the effectiveness (in terms of energy saving ratio) of our proposed approaches.

Based on these data, SPM partitioning and code allocation are decided by the proposed approach described in Section 3. The integer programming problems proposed in this paper are solved with an ILP solver glpsol 4.23 [15)⋆1]. Task scheduling and cache simulations are performed to measure the number of cache hits and misses. In this paper, the task with shorter period is given to higher priority. We derive the total energy consumption of the task sets from these pieces of information.

4.2 Results and Discussion

We brought a simple approach as baseline to evaluate and compare the benefits of proposed approaches because of lacking a previous approach for priority-based multi-task systems. In this approach, the capacity of SPM is partitioned evenly for each task at first, and then code allocation to SPM about each task is decided by a knapsack problem presented in Ref. 4).

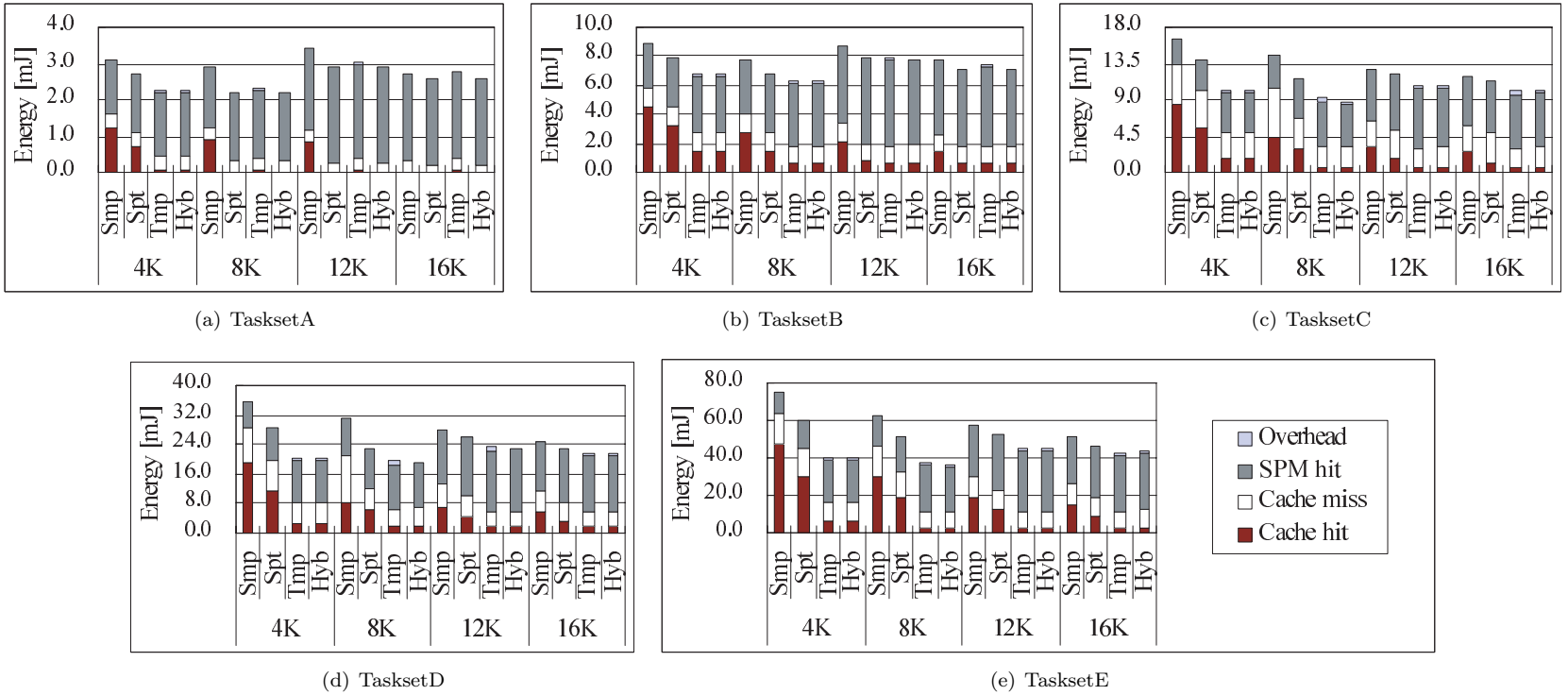
Figure 6 shows the experimental results. The bars show the energy consumption in mJ. The amount of energy consumed in the memory subsystem are analyzed into four factors; access energy of cache hits, that of cache misses (including access energy on the main memory), that of SPM hits, and energy overhead on MM–SPM copy, respectively. 'xK' in the x-axis denotes the size of SPM. 'Smp', 'Spt', 'Tmp', and 'Hyb' denote the energy consumption of the simple approach, that of the spatial approach, that of the temporal approach, and that of the hybrid approach, respectively.

From these figures, the effectiveness of the proposed approaches is confirmed. Compared with the simple approach, energy savings can be achieved in all situations. At maximum of each task set and each SPM capacity, 28% of energy reduction by the spatial approach, 47% by the temporal and hybrid one was achieved. On average, 12% of energy by the spatial approach, 22% by the temporal one, and, 23% by the hybrid one was reduced.

Next, we focus on an influence exerted by the number of tasks. In TasksetE which includes 10 tasks, the effectiveness of the temporal and hybrid approaches rose when the size of SPM is limited small because of occupation to SPM by the currently running task. In addition, the larger SPM size was, the less energy consumption of the hybrid approach that can combine use of the spatial and temporal approach. Note that total energy consumption of MM–SPM copy is not trivial. A similar tendency was appeared in Fig. 6(c) and Fig. 6(d). Thus, if the number of task is large, both the temporal and hybrid approach where the running task can occupy parts of SPM region becomes the most effective. On the other hand, the spatial and hybrid approach was more effective in the case that the number of task is small shown in Fig. 6(a) and Fig. 6(b). This implies SPM–MM transferring is not necessarily needed when the total code size of task set is small.

Additionally, when the proposed approaches applied, 8 KB in SPM size achieve the least energy consumption in any task set. In other words, increasing SPM size

⋆1 In our experimental environment, all integer programming problems for SPM partitioning and code allocation were solved within 10 seconds.



(a) TasksetA (b) TasksetB (c) TasksetC

(d) TasksetD (e) TasksetE

Fig. 6 Experimental results.

is not always the best way to reduce energy consumption. Note that increasing SPM size introduces an extra energy on not only MM–SPM transfer but also each read access to SPM if code allocation was enough to perform on small SPM size. The proposed integer programming problems in this paper treat SPM size as an input value of constant. This implies the possibility that SPM size should be decided appropriately for the purpose of energy minimization.

Finally, we focus on the feature of the hybrid approach. As described in Section 3.5, the hybrid approach is a combination of the spatial and temporal approach. Therefore, the amount of energy reduction by hybrid approach should be the maximum among proposed three approaches. However, the hybrid approach was not always show the best result, for example in 4K SPM of **Fig. 6(d)**. We thought this is mainly attributed to the influence on access to cache. Since run-time analysis is required to calculate the number of cache misses, dynamic behavior on cache access was neglected in the integer programming problem we formulated. Nevertheless, it is remarkable contribution that the hybrid approach can achieve the stable energy reduction in all SPM size of all task sets.

5. Conclusions

In this paper, three approaches for partitioning and allocating of SPM in the fixed-priority multi-task systems were proposed. Our approaches give the benefit

on energy reduction in the instruction memory, and are applicable to real-time environments. For each approach, we formulated the integer programming problem. By deriving its solutions, optimal SPM partitioning and code allocation can be simultaneously determined. Experimental results showed the effectiveness of our approaches. Additionally, the hybrid approach that exploits both the spatial and temporal approaches together obtained the best result especially when the number of tasks is large.

SPM partitioning and code allocation are statically decided in our approaches. However, if they were changed dynamically, more energy reduction can be achieved. In future, a dynamic solution to decide SPM partitioning and code allocation will be studied. Also, we intend to extend the approaches for data memory subsystems and preemptive multi-task environments.

Acknowledgments We thank Prof. Tohru Ishihara, Dr. Gang Zeng, and Dr. Tetsuo Yokoyama for suggesting our experiments. This work is supported in part by Core Research for Evolutional Science and Technology (CREST) from Japan Science and Technology Agency.

References

1) Segars, S.: Low Power Design Techniques for Microprocessors, *IEEE International Solid-State Circuits Conference* (*Tutorial*) (2001).
2) Banakar, R., Steinke, S., Lee, B.-S., Balakrishnan, M. and Marwedel, P.: Scratchpad Memory : A Design Alternative for Cache On-chip memory in Embedded Systems, *Proc. International Symposium on Hardware/Software Codesign* (*CODES*), Estes Park, Colorado (2002).
3) Avissar, O., Barua, R. and Stewart, D.: An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems, *ACM Trans. on Embedded Computing Systems* (*TECS*), Vol.1, No.1, pp.6–26 (2002).
4) Steinke, S., Wehmeyer, L., Lee, B. and Marwedel, P.: Assigning Program and Data Objects to Scratchpad for Energy Reduction, *Proc. Conference on Design, Automation and Test in Europe* (*DATE*), Washington, DC, USA, pp.409–415, IEEE Computer Society (2002).
5) Panda, P.R., Nicolau, A. and Dutt, N.: *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*, Kluwer Academic Publishers, Norwell, MA, USA (1998).
6) Angiolini, F., Benini, L., and Caprara, A.: An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.11, pp.1660–1676 (2005).
7) Janapsatya, A., Ignjatovic, A. and Parameswaran, S.: Exploiting Statistical Information for Implementation of Instruction Scratchpad Memory in Embedded System, *IEEE Trans. on Very Large Scale Integration* (*VLSI*) Systems, Vol.14, No.8, pp.816–829 (2006).
8) Steinke, S., Grunwald, N., Wehmeyer, L., Banakar, R., Balakrishnan, M. and Marwedel, P.: Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory, *Proc. 15th International Symposium on System Synthesis* (*ISSS*), Kyoto, Japan (2002).
9) Janapsatya, A., Parameswaran, S. and Ignjatovic, A.: Hardware/Software Managed Scratchpad Memory for Embedded System, *Proc. 2004 IEEE/ACM International Conference on Computer-Aided Design* (*ICCAD '04*), Washington, DC, USA, pp.370–377, IEEE Computer Society (2004).
10) Kandemir, M., Kadayif, I., Choudhary, A., Ramanujam, J. and Kolcu, I.: Compiler-Directed Scratch Pad Memory Optimization for Embedded Multiprocessors, *IEEE Trans. on Very Large Scale Integration* (*VLSI*) Systems, Vol.12, No.3, pp.281–287 (2004).
11) Verma, M., Petzold, K., Wehmeyer, L., Falk, H. and Marwedel, P.: Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach, *Proc. IEEE 3rd Workshop on Embedded System for Real-Time Multimedia* (*ESTIMedia*), Jersey City, USA, pp.115–200 (2005).
12) SimpleScalar LLC. http//www.simplescalar.com/ (accessed 2009-01-27)
13) Wilton, S.J.E. and Jouppi, N.P.: CACTI: An Enhanced Cache Access and Cycle Time Model, *IEEE Journal of Solid-State Circuits*, Vol.31, No.5, pp.677–688 (1996).
14) The Micron System Power Calculator. http://www.micron.com/support/designsupport/tools/powercalc/powercalc (accessed 2009-01-27)
15) Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T. and Brown, R.B.: MiBench: A Free Commercially Representative Embedded Benchmark Suite, *Proc. IEEE International Workshop on the Workload Characterization* (*WWC*), Washington, DC, USA, pp.3–14 (2001).
16) GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/ (accessed 2009-01-27)

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

(Recommended by Associate Editor: *Yuichi Nakamura*)

Hideki Takase received the M.S. in Information Science from Nagoya University in 2009. Currently he is a Ph.D. candidate at the Information Science from Nagoya University. His research interests include compilers, real-time operating systems, and low energy design for embedded systems. He received the Incentive Award from Computer Science group of IPSJ in 2008.

Hiroyuki Tomiyama received his Ph.D. degree in computer science from Kyushu University in 1999. From 1999 to 2001, he was a visiting postdoctoral researcher with the Center of Embedded Computer Systems, University of California, Irvine. From 2001 to 2003, he was a researcher at the Institute of Systems & Information Technologies/KYUSHU. In 2003, he joined the Graduate School of Information Science, Nagoya University, as an assistant professor, where he is now an associate professor. His research interests include design automation, architectures and compilers for embedded systems and systems-on-chip. He currently serves as an editorial board member of IPSJ Transactions on SLDM, IEEE Embedded Systems Letters, and International Journal on Embedded Systems. He has also served on the organizing and program committees of several premier conferences including ICCAD, ASP-DAC, DATE, CODES+ISSS, and so on. He is a member of ACM, IEEE and IEICE.

Hiroaki Takada is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in Information Science from the University of Tokyo in 1996. He was a Research Associate at the University of Tokyo from 1989 to 1997, and was an Assistant Professor and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IEICE, and JSSST.