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

Hideki Takase, Hiroyuki Tomiyama and Hiroaki Takada
Graduated School of Information Science, Nagoya University
C3-1(631), Furo-cho, Chikusa-ku, Nagoya 464-8603 Japan

Abstract— This paper proposes three approaches for allocation of scratch-pad memory in non-preemptive fixed-priority multi-task systems. These approaches can reduce energy consumption of instruction memory. Each approach is formulated as an integer programming problem which simultaneously determines (1) partitioning of scratch-pad memory spaces for the tasks, and (2) allocation of functions to the scratch-pad memory space for each task. The experimental results show the effectiveness of the proposed approaches.

I. 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 in order to improve their performance. Cache also contributes 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, ARM920T processor dissipates 43 % of the power in its cache [1]. 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. 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 on one access than cache. On the other hand, programmers or compilers need to decide the allocation of program code or data variable on SPM since hardware units for allocation management do not exist.

Recently embedded processors are typically required to execute two or more tasks concurrently. The task scheduling algorithm under the priority is generally employed because fast response is important in real-time systems. However, almost all of previous techniques have assumed the single-task environment. This paper proposes three approaches (*i.e.* spatial, temporal, and hybrid approaches) which enable energy efficient usage of SPM for non-preemptive fixed-priority multi-task systems. Each approach is formulated as an integer programming problem which simultaneously determines optimal SPM partitioning into the tasks and code allocation to SPM for each task in terms of the energy efficiency. These approaches can minimize energy consumption of instruction memory subsystems.

The rest of this paper is organized as follows. In Section II., a brief survey on related works is provided. Section III. describes our approaches for partitioning and allocating to SPM in detail. Section IV. presents experimental setup and results. Finally, Section V. summarizes the contributions of this paper.

II. Related Works and Our Strategy

Banakar et al. indicated that the energy consumption of SPM-based systems is less than that of cache-based systems by 40 % on average [2]. In [3], a compiler-oriented optimization technique to allocate data variables to SPM was proposed. The authors of [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. A dynamic programming algorithm for assigning code/data to SPM with a polynomial-time complexity was proposed in [5]. However, these previous techniques are only applicable to single-task systems.

The authors of [6] introduced a hardware mechanism for efficient overlay of SPM at runtime. In [7], the use of SPM in multiprocessor systems was studied.

Verma et al. proposed the SPM region management technique which can be applied to the multi-task environment [8]. Each task shares the SPM region in a given way for the purpose of energy minimization. However, the proposed approach in [8] targets on the time sharing system, and 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 [8] 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 achieve energy efficient utilization of SPM in multi-task environments whose tasks are scheduled under the fixed-priority. Moreover, the integer programming problems we formulate can obtain the optimal solution at a practical computation time.

III. SPM Partitioning and Allocation approaches

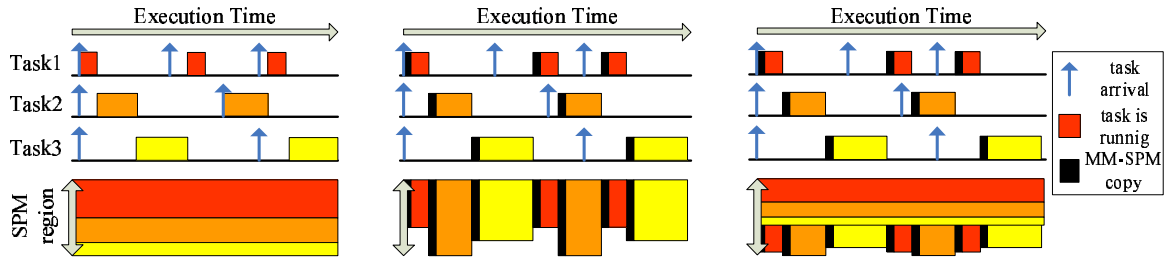
In this section, details of the proposed approaches which can be applied to priority based multi-task systems are described. We present three approaches: spatial, temporal, and hybrid approaches. It is noted that this work focuses on the energy reduction for instruction access and code allocation is performed at a function-level granularity.

A. 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 Figure 1. There are neither an inter-task communication nor synchronization, that is, each task executes independently. All tasks are cyclically activated and the task period is statically decided.

The tasks are scheduled according to the fixed-priority based policy. The task which has the highest priority among ready

978-1-4244-2782-6/09/$25.00 ©2009 IEEE



(a) Spatial Approach (b) Temporal Approach (c) Hybrid Approach

Figure 2. SPM partitioning and allocation approach

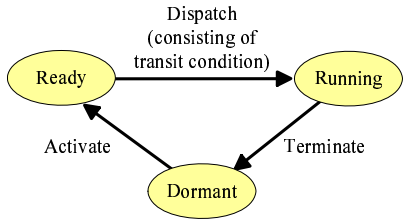


Figure 1. Task state transition diagram

TABLE I. VARIABLE DEFINITIONS

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 $func_{i,j}$
$size_{i,j}$	The code size of $func_{i,j}$
$E_{SPMgain}$	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
$Esaving_{i,j}$	The energy reduction if $func_{i,j}$ is allocated to SPM
$SPMsize$	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.

state tasks transits to the running state when no task is runnnig. Also, no task preemption is assumed to occur.

B. Definitions

Table I denotes variable definitions in our integer programming formulation.

C. Spatial Approach

The spatial approach completely partitions the SPM region into the tasks. Each task exclusively uses the given SPM region. Figure 2(a) shows the example for partitioning of SPM into three disjoint regions. The SPM partitioning for the tasks and the code allocation to each task are statically determined. The contents in SPM are not changed at runtime.

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

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

$$\textbf{Maximize}: Esaving = \sum_i \sum_j Esaving_{i,j} \times x_{i,j}$$
$$Esaving_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$
$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$
$$\textbf{s.t.}: \sum_i \sum_j size_{i,j} \times x_{i,j} \leq SPMsize$$

It aims at the maximization of the total energy reduction E_{saving}. The number of executed instruction on each function $fetch_{i,j}$ 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.

D. Temporal Approach

Figure 2(b) shows the SPM allocation in the temporal approach where whole SPM region 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 Figure 2(b)). The functions allocated to SPM are decided in consideration of the energy consumption due to this transferring. The transfer overheads is proportional to the code size of a function. For this reason, functions with not only frequently executions but also small size is likely to be allocated to SPM region.

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

$$\textbf{Maximize}: Esaving = \sum_i \sum_j Esaving_{i,j} \times y_{i,j}$$
$$Esaving_{i,j} = fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}$$
$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$
$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$
$$\textbf{s.t.}: \forall i.\ \sum_j size_{i,j} \times y_{i,j} \leq SPMsize$$

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

E. Hybrid Approach

As shown in Figure 2(c), the hybrid approach is a mixture of spatial and temporal approaches. The amount of SPM capacity used by the spatial approach and the temporal one is given by each dividing so that the total energy reduction E_{saving} becomes the largest. This achieves more frexible 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 (upper half in Figure 2(c)) and the region of temporal approach (lower half in Figure 2(c)). An integer programming problem formulated in the hybrid approach is as follows.

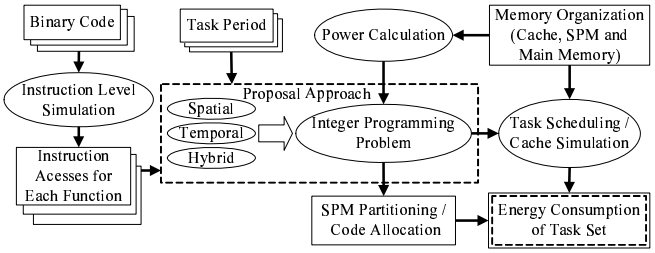


Figure 3. Experimental procedure

$$\textbf{Maximize}: Esaving = Esaving\_spt + Esaving\_tmp$$
$$Esaving\_spt = \sum_i \sum_j Esaving\_spt_{i,j} \times x_{i,j}$$
$$Esaving\_tmp = \sum_i \sum_j Esaving\_tmp_{i,j} \times y_{i,j}$$
$$Esaving\_spt_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{SPMgain}$$
$$E_{SPMgain} = E_{Cache\_read} - E_{SPM\_read}$$
$$Esaving\_tmp_{i,j} = (fetch_{i,j} \times E_{SPMgain} - Eoverhead_{i,j}) \times \frac{hyperperiod}{period_i}$$
$$Eoverhead_{i,j} = size_{i,j} \times (E_{SPM\_write} + E_{MM\_read})$$
$$\textbf{s.t.}: SPMsize\_spt + SPMsize\_tmp \leq SPMsize$$
$$\textbf{s.t.}: \sum_i \sum_j size_{i,j} \times x_{i,j} \leq SPMsize\_spt$$
$$\textbf{s.t.}: \forall i. \sum_j size_{i,j} \times y_{i,j} \leq SPMsize\_tmp$$
$$\textbf{s.t.}: \forall i, \forall j. \; x_{i,j} + y_{i,j} \leq 1$$

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 approach and the temporal one, 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 division of SPM for the spatial region and the temporal one, the partitioning of the spatial region into the tasks, and the allocation of the function to the SPM region for each task are simultaneously determined by finding these values.

IV. Evaluation

A. Experimental Procedure and Tools

Our experimental procedure is depicted in Figure 3. 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 [9] 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 associativity. The size of SPM is selected from 4, 8, 12, and 16 KB. Access power for cache and SPM is calculated by using CACTI 4.2 [10], and that for the main memory is on Micron System Power Calculator [11]. Also, we do not consider static energy consumptions.

Based on these data, SPM partitioning and code allocation are decided by the proposed approach described in Section III.. In our experiments, the integer programming problem is solved with an open-source ILP solver glpsol 4.23 [12]. In our experimental environment, optimal solutions in our integer programming problem are able to derived up to 1 second. Task scheduling and cache simulation 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 when the task set is performed from these pieces of information. Also, we do not consider leakage energy.

TABLE II. 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

B. Results and Discussion

The proposed allocation approaches are evaluated on a synthetic task set from MiBench suite [13] as presented in Table II. The task period is set for CPU utilization when the task set was performed become to about 60 % in portion to the total number of fetched instructions on task.

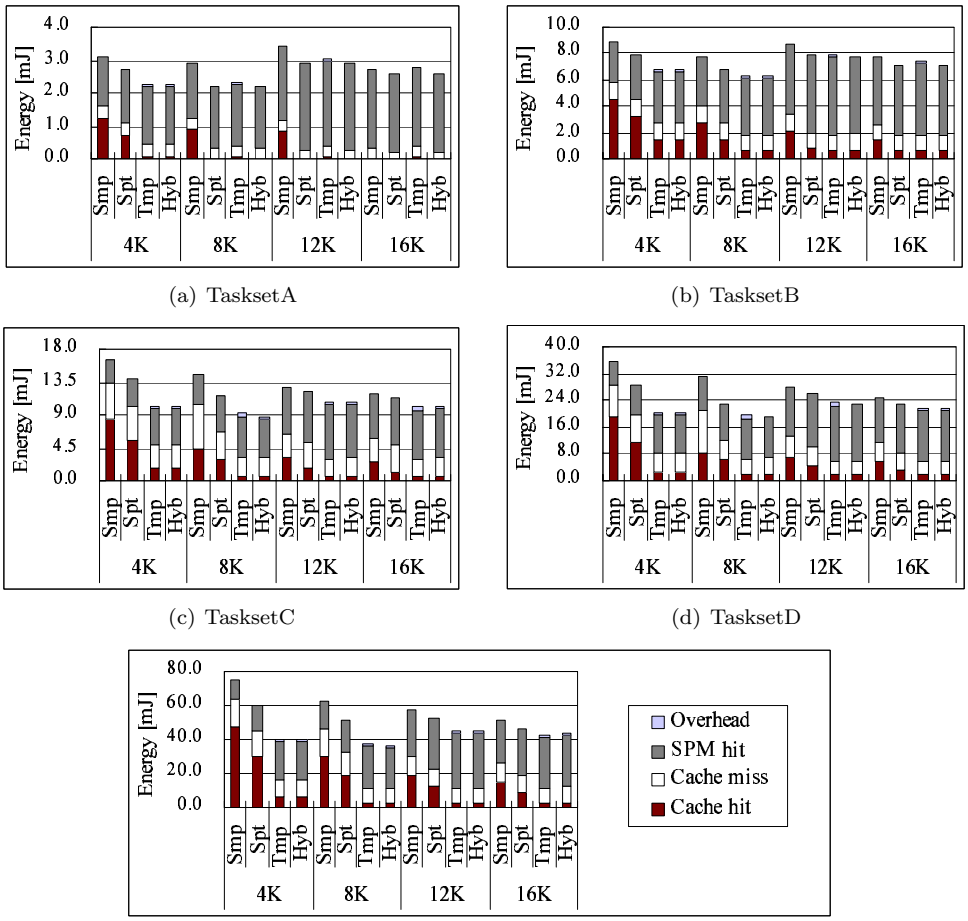
Lacking a previous approach for a priority based multi-task system, we brought a simple approach as baseline to evaluate and compare the benefits of proposed approaches. In this approach, the capacity of SPM is partitioned equally for each task at first, and then code allocation to SPM about one-by-one task is decided by a knapsack problem presented in [4].

Figure 4 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. Energy savings can be achieved in almost all situations. Compared with the simple approach, 12.4 % of energy consumption by the spatial approach, 21,6 % by the temporal approach, and, 23.0 % by the hybrid approach was reduced on average of each task set and each SPM capacity.

Next, we focus on an influence exerted by the size of SPM. In TasksetE which includes 10 tasks, the effectiveness of the temporal and hybrid approaches rises when the size of SPM is limited small because of occupation to SPM by the currently runnning task. In addition, the larger SPM size is, the less energy consumption of hybrid approach that can combine use of the spatial and temporal approaches. A similar tendency is appeared in the rest figures. This tendency means that the hybrid approach where the runnning task can occupy both the spatial region and the temporal one becomes the most effective. Thus, the hybrid approach achieves optimal in energy consumption among proposed approaches.

Additionally, when the proposed approaches applied, SPM size in 8 KB achieves the least energy consumption in any task set. In other words, increasing SPM size is not always the best way to reduce energy consumption. Note that increasing SPM size introduces an extra energy on not only MM–SPM copy but also each read access to SPM if code allocation was enough to perform on small SPM size, This implies the possibility that SPM size should be decided appropriately for the purpose of energy minimization.



(a) TasksetA

(b) TasksetB

(c) TasksetC

(d) TasksetD

(e) TasksetE

Figure 4. Experimental results

V. Conclusions

In this paper, three approaches for energy efficient usage of SPM in the fixed-priority multi-task systems are proposed. For each approach, we formulated the integer programming problem. By finding optimal solutions, SPM partitioning and code allocation can be simultaneously determined. Experimental results show that our approach can achieve energy reduction in the instruction memory. Additionally, the hybrid approach which exploits both the spatial and temporal techniques together obtains the stable result in all situations. In future, 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] S. Segars, "Low Power Design Techniques for Microprocessors," In *Proc. of IEEE Int'l Solid-State Circuits Conference*, tutorial, Feb, 2001.

[2] R. Banakar, et al., "Scratchpad Memory: A Design Alternative for Cache On-Chip Memory in Embedded Systems," In *Proc. of Int'l Sympo. on CODES*, pp. 73–78, May, 2002.

[3] O. Avissar, et al., "An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems," *ACM Trans. on Embedded Computing Systems*, Vol. 1, No. 1, pp. 6–26, 2002.

[4] S. Steinke, et al., "Assigning Program and Data Objects to Scratchpad for Energy Reduction," In *Proc. of the conference on DATE*, pp. 409–415, Mar, 2002.

[5] F. Angiolini, et al., "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.

[6] A. Janapsatya, et al., "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.

[7] M. Kandemir, et al., "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.

[8] M. Verma, et al., "Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach," In *Proc. of IEEE 3rd Workshop on ESTIMedia*, pp. 115–120, Sep, 2005.

[9] SimpleScalar LLC, `http://www.simplescalar.com`.

[10] S. J. E. Wilton and N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid State Circuit*, Vol. 31, No. 5, pp. 677–688, 1996.

[11] The Micron System Power Calculator, `http://www.micron.com/support/designsupport/tools/powercalc/powercalc`.

[12] GLPK (GNU Linear Programming Kit), `http://www.gnu.org/software/glpk/`.

[13] M. R. Guthaus, et al., "MiBench: A free, commercially representative embedded benchmark suite," In *Proc. of IEEE Int'l Workshop on Workload Characterization*, pp.3–14, Dec, 2001.