Vol.2009-SLDM-140 No.8
2009/5/21

組込みリアルタイムシステムにおけるスクラッチパッドメモリ管理技術

高瀬英希[†1,†2] 冨山宏之[†1] 高田広章[†1]

本研究では，命令メモリの消費エネルギーの削減を目的とした，スクラッチパッドメモリ（SPM）管理のためのフレームワークを提案する．本フレームワークは，リアルタイム要求の高いマルチタスクシステムに適用可能である．SPM 領域のタスクへの分割と，SPM 領域に配置されるコードは，システム設計時決定される．実行時における SPM の制御は，リアルタイム OS とハードウェアの協調により実現される．本フレームワークのシミュレーション環境を構築し，評価実験を行ってその有効性を確認した．

A Scratch-Pad Memory Management Framework for Embedded Real-Time Systems

Hideki Takase,[†1,†2] Hiroyuki Tomiyama[†1]
and Hiroaki Takada[†1]

In this paper, we propose a scrath-pad memory (SPM) management framework for minimizing energy consumption in preemptive fixed-priority multi-task real-time systems. The framework achieves energy minimization in the instruction memory subsystems. At the design time, energy-optimal usage of SPM, i.e., SPM partitioning and code allocation, is determined. At runtime, SPM is managed with the cooperative support of a real-time operating system and hardware modules. The proposed framework has been implemented in our simulation environment, show the results of effectiveness of the proposed framework. and its effectiveness has been demonstrated by a set of experiments.

1. Introduction

Energy minimization has become one of the primary goals in the design of embedded real-time systems. These days, cache memory is used not only in general-purpose processors but also in embedded processors. Caches improve average performance and 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)]. Additionally, it is difficult to guarantee a real-time performance since the number of cache access miss is unpredictable statically. More recently, scratch-pad memory (SPM) has attracted attention due to its energy efficiency and real-time guarantees.

SPM only consists of decoding circuits, data arrays and output units. SPM is more efficient than cache in terms of area and energy since no tag comparison on SPM access is necessary. So far, a considerable amount of research on SPM has been conducted 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)]. In 3), a compiler-oriented optimization technique to SPM was proposed. The authors of 4) formulated the energy optimal code/data allocation to SPM as a 0/1 IP problem. An allocation method for data-SPM based on the possibility of data-cache conflicts in 5). In 6), a dynamic programming algorithm for allocating code/data to SPM was studied. SPM is a software-controlled memory, that is, hardware units which manage the contents of SPM do not exist. 7), 8) presented a customized hardware mechanism and instruction for efficient code/data transfer to SPM at runtime. A hardware/software concerted approach of managing SPM content dynamically was proposed in 9). However, these previous techniques are only applicable to single-task systems.

In embedded real-time systems, the scale and the complexity are going to increase. Embedded processors are typically required to execute two or more tasks concurrently. Verma et al. proposed the SPM region management scheme which can be applied to the multi-task environment whose tasks are scheduled by round robin manner[10)]. Each task shares the SPM region in a given way for the purpose of energy minimization. However, round robin manner is not generally adopted in real-time systems. To satisfy task deadline constraints, the priority-based scheduling policy is generally employed because high responsiveness is important in real-time systems.

We studied energy efficient usage of SPM for priority-based multi-task systems. First of all, the SPM partitioning and code allocation approaches which are applicable to non-preemptive environment were proposed[11)]. Each approach formulated as an integer programming (IP) model. Next, We extended prior works to be applied to the

†1 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†2 日本学術振興会特別研究員 DC
Research Fellow of the Japan Society for the Promotion of Science

preemptive multi-task systems in 12) since a schedulability is not enough guaranteed in non-preemptive systems. The SPM partitioning and code allocation approaches in 12) can reduce the instruction access energy in the embedded real-time systems. However, management technique that take a role of code transferring dynamically was a lack in 11), 12), and experiments were perfomed by trace-based simulation.

In this paper, SPM management framework for real-time multi-task systems are proposed. We target on the systems where tasks are scheduled by preemptive fixed-priority policy. At static system design phase, a function of profiler and compiler in the framework achieves our SPM partitioning and code allocation approaches proposed in 12). Runtime code transferring to SPM performs under the support of Real-Time Operating Systems (RTOS) and hardware modules. Our framework also contributes that there is never any need for modifying in a source code of target software. We implement a framework to the practical systems, and perform evaluation experiments where RTOS can execute. Energy minimization of instruction memory subsystems can be achieved by proposed framework.

The rest of this paper is organized as follows. In Sec. 2, a review of our prior approaches for SPM partitioning and code allocation is described. Sec. 3 presents our SPM management framework in detail. Sec. 4 shows experimental environment and results. Finally, Sec. 5 summarizes the contributions of this paper.

2. SPM Partitioning and Code Allocation Approaches

In this section, overview of our prior approaches for effective usage of SPM region is shown. We proposed three approaches: spatial, temporal, and hybrid approaches. Our prior work for SPM partitioning and code allocation approaches is presented in 12). These are able to be applied to priority-based preemptive multi-task systems. It is noted that this work focuses on the energy reduction for instruction memory access, and code allocation performs at a function-level granularity.

2.1 Target System Organization

We target an environment where two or more tasks are executed on a single processor. Tasks take dormant, ready and running states as shown in **Fig. 1**. There are neither an inter-task communication nor synchronization. All tasks are cyclically activated and periods of tasks are statically decided. The tasks are executed according to a fixed-priority based preemptive scheduling. The highest priority one among the ready state tasks transits to the running state. If higher priority task transits ready state, task preemption and context-switching are occurred even when lower priority one is runnnig.

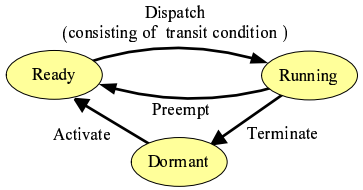


Fig. 1 Task state transition diagram

2.2 Spatial Approach

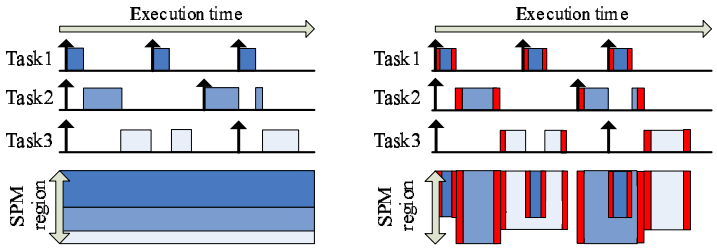
The spatial approach partitions the SPM region for the tasks statically. **Fig. 2(a)** shows the example for partitioning of SPM into three disjoint regions. The amount of SPM partitioned to the task depends on access frequency in its functions. The SPM partitioning and the code allocation are statically determined.

2.3 Temporal Approach

As shown in **Fig. 2(b)**, whole SPM address space is assigned to the current running task. It is necessary to transfer the code of task from main memory to SPM at the two timing. The reason why code transfer performs two times is the occurrence of context switching by task preemption. The situation that the code accessed by the preempted task after the restart is not allocated to SPM space can be prevented. The former transfer timing is when the scheduler dispatches the task for the first time in its period. The functions with frequently access among the function in the task are transferred. The contents of SPM is returned before it runs by code transfer again at the latter timing when the task transits to the dormant state. 'MM-SPM copy' at Fig. 2(b) refers these transfer of program code.

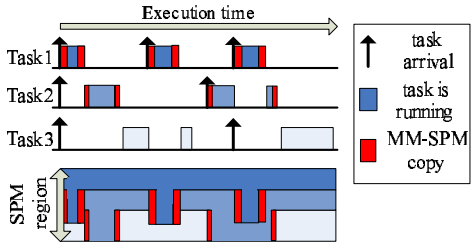
2.4 Hybrid Approach

As shown in **Fig. 2(c)**, the hybrid approach is a mixture of previous two approaches. The amount of SPM capacity a task can use is sum of the region partitioned to itself



(a) Spatial Approach

(b) Temporal Approach



(c) Hybrid Approach

Fig. 2 SPM partitioning and code allocation approach

by spatial approach and the temporal region where the lower priority tasks use. In other words, the higher priority task can *preempt* and temporarily utilize SPM space partitioned to the lower priority task. For example, Task1 in Fig. 2(c) uses its own spatial region and preempted temporal region where Task2 and Task3 use.

2.5 Integer Programming Models

In each approach, the IP models are formulated for the maximization of energy reduction on the instruction memory subsystems in 12). Each IP model simultaneously determines optimal SPM partitioning and code allocation in terms of the energy efficiency. However, we introduce only an IP formulation of hybrid approach in this paper due to lack of space. Please refer our previous work in 12) for definitions of symbols and details of the IP formulations.

An IP model formulated in the hybrid approach is as follows.

$$\textbf{Maximize}: Esaving = Esaving\_spt + Esaving\_tmp \tag{1}$$

$$Esaving\_spt = \sum_i \sum_j Esaving\_spt_{i,j} \times x_{i,j} \tag{2}$$

$$Esaving\_spt_{i,j} = fetch_{i,j} \times \frac{hyperperiod}{period_i} \times E_{gain} \tag{3}$$

$$Esaving\_tmp = \sum_i \sum_j Esaving\_tmp_{i,j} \times y_{i,j} \tag{4}$$

$$Esaving\_tmp_{i,j} = (fetch_{i,j} \times E_{gain} - Eoverhead_{i,j} \times 2) \times \frac{hyperperiod}{period_i} \tag{5}$$

$$E_{gain} = E_{C\_read} - E_{S\_read} \tag{6}$$

$$Eoverhead_{i,j} = size_{i,j} \times (E_{S\_write} + E_{MM\_read}) \tag{7}$$

$$\textbf{s.t.}: \sum_i SPMsize\_spt_i \leq SPMsize \tag{8}$$

$$SPMsize\_spt_i = \sum_j size_{i,j} \times x_{i,j} \tag{9}$$

$$\textbf{s.t.}: \forall i. \ \sum_j size_{i,j} \times y_{i,j} \leq SPMsize\_tmp_i \tag{10}$$

$$\exists k, \ period_k > period_i \ . \quad SPMsize\_tmp_i = SPMsize - \sum_k SPMsize\_spt_k \tag{11}$$

$$\textbf{s.t.}: \forall i, \ \forall j. \ x_{i,j} + y_{i,j} \leq 1 \tag{12}$$

Here, the decision variables are $SPMsize\_spt_i$, $SPMsize\_tmp_i$, $x_{i,j}$, and $y_{i,j}$. By utilizing task period $period_i$ as information. a lot of codes with more frequently access become to be allocated to SPM. In formulas on hybrid approach, the partitioning of the spatial region for the tasks, the temporal region used by the higher priority task, and the allocation of the function to the SPM region for each task are simultaneously determined by finding these values.

3. The SPM Management Framework

We propose the whole SPM management framework which can enable dynamic management to SPM space. Our framework consists of statical system design flow and runtime SPM management units depicted in **Fig. 3**. In the system design phase, SPM partitioning and code allocation are performed by a part of function of profiler and compiler. The SPM management units support dynamic code transfer to SPM with RTOS and hardware modules. It is particularly worth noting that any modification to target software is needed on our framework.

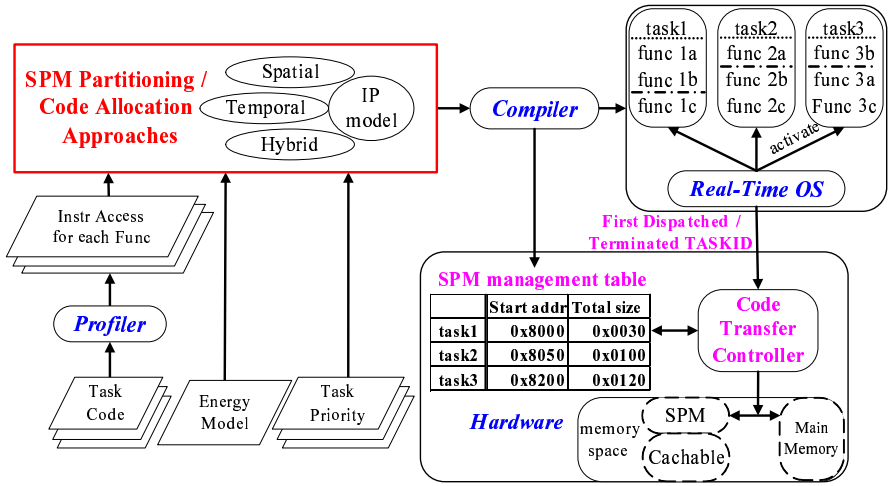


Fig. 3 The framework of statical design and runtime SPM management

3.1 Statical Analysis and System Design

Profiler collects the total number of executed instructions for each function on task once execution by the instruction-level simulation. These pieces of information are used to the input values to IP model. Then, the optimal SPM partitioning and code allocation by our proposed approach at compile time described in Sec. 2. Compiler also generates the SPM management table with the code group to be allocated to SPM space. The key of this table is a task ID, and values of key are the start address of code group and the total size of functions.

3.2 Real-Time OS Support

The RTOS shedules task execution based on the priority as a basic capability. And more, RTOS enhances capability to hardware support for SPM management. The dispatcher notifies ID number of target task to the code transfer contoroller at two timing described in Sec. 2.3. The purpose of this is hardware have no manner to know which task executed on the system.

3.3 Runtime Hardware Support

The SPM management units on hardware consists of SPM management table and code transfer controller shown in Fig. 3. SPM management table generated by the compiler is specially implemented on a part of hardware. Code transfer controller is the dedicated module to be performed in the temporal or hybrid approach. When RTOS notifies the task ID, the controller refers information on the table and transfer required program code from main memory to SPM.

4. Evaluation and Experimental Results

4.1 Experimental Procedure and Tools

We performed evaluation experiments to assess the effectiveness of the framework. SkyEye–1.2.6_rc1[13] was used for the instruction-level hardware simulation. We chose ARM920T[14] as target core. We employed TOPPERS/ASP kernel (Release 1.3.2)[15] as RTOS. GNU arm-elf[16] was used for cross development environment.

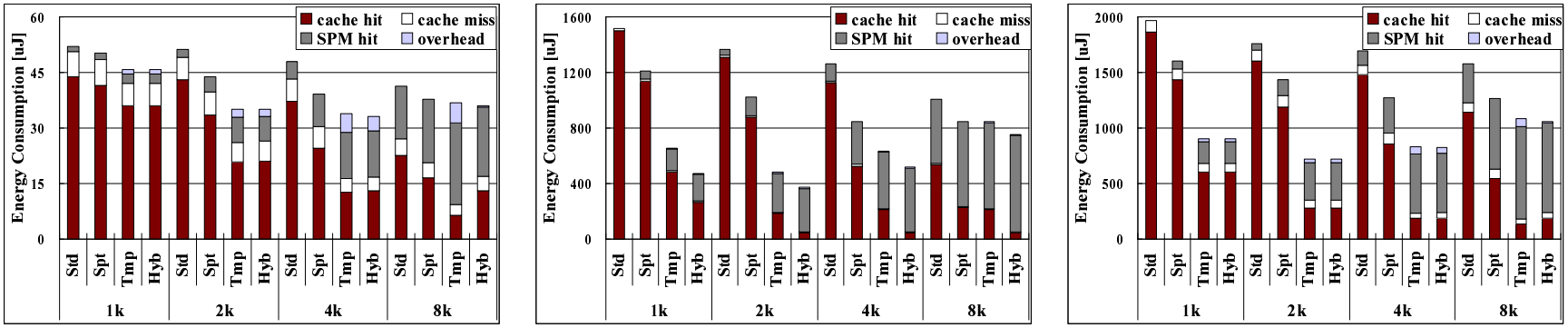
The instruction memory subsystem consists of cache and SPM as on-chip memory, and SDRAM as off-chip main memory. The cache organization is 8 KB in size and 4-way in associativity. The size of SPM is selected from 1, 2, 4, and 8 KB. We assumed that access to SDRAM is performed by 4-words burst access. Memory access energy model is based on the CACTI 5.3[17]. Also, we do not consider static energy consumptions.

We selected 16 programs as task code from EEMBC[18] benchmark suite. For each task, the same input data is used for both profiling and evaluation phases. The proposed framework are evaluated on 3 synthetic task sets as presented in **Table 1**. In our experiments, the periods of tasks are set according to be proportional to their execution times, and the task with shorter period is given to higher priority. The total CPU utilization is set about 50 %. 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, proposed SPM management framework are applied. We constructed proposed framework described in Sec. 3. GNU ILP solver glpsol 4.23[19] was

Table 1 Task sets

	# of task	Tasks included
TasksetA	5	aifftr, basefp, bitmnp, cacheb, idctrn
TasksetB	11	bezier, conven, dither, ospf, pktflow, rgbcmy, rgbyiq, rotate, routelookup, text, viterb
TasksetC	16	the combination of TasksetA and TasksetB



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

Fig. 4 Experimental results

employed to solve IP models we have proposed in prior work 12) at compile time. Each task code was cross-compiled into a binary code and linked to TOPPERS/ASP kernel, which is able to execute on SkyEye. For the purpose of runtime SPM support described in Sec. 3.2 and Sec. 3.3, we expanded the functionality of TOPPERS/ASP kernel and SkyEye. To measure the number of each memory access, each task set was executed under the scheduling by TOPPERS/ASP kernel. We derived 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 framework because of lacking a previous approach for priority-based preemptive 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 4).

Fig. 4 shows the experimental results. The amount of energy consumption in the memory subsystem shown as bars is 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 on code transferring from main memory to SPM. 'xK' in the x-axis denotes the size of SPM. 'Std', 'Spt', 'Tmp', and 'Hyb' denote the energy consumption of the simple approach, the spatial approach, the temporal approach, and the hybrid approach, respectively.

From these figures, the effectiveness of the proposed framework in this paper is confirmed. Energy savings in the memory subsystems can achieve compared with the simple approach. Up to 73 % of energy reduction was achived by hybrid approach in the case of 2K SPM of TasksetB. On average of each task set and each SPM capacity, 17 % of energy by spatial, 36 % by temporal, and 39 % by hybrid was reduced.

Next, we focus on the comparison among proposed three approaches. Experimental results show that both the temporal and hybrid approach where parts of SPM region is occupied by the runnning task can minimize the total energy consumption than spatial approach. This tendency indicates the effectiveness of our proposed SPM management framework. Note that total energy consumption of code transferring is not trivial. An appropriate SPM management is better way for reducing energy in the instruction memory subsystems. Moreover, hybrid approach becomes the most effective in allmost all situations, $^{\star 1}$. As described in Sec. 2.4, hybrid approach have a feature that the higher

$\star$1 In the case of 1K SPM of TasksetC, temporal approach results slightly smaller energy consumption than hybrid one. This is due to the difference of cache miss energy consumption, and the point of view of energy reduction in on-chip memory by hybrid approach is larger.

priority task can preempt not only CPU citizenship but also spatial SPM region of the lower priority ones. We insist on hybrid approach is the most suitable SPM partitioning and code allocation approach for the preemptive real-time systems.

Additionally, when the proposed framework applied, enlarging SPM capacity does not lead to the reduction in energy consumption. For example, 2 KB in SPM size achieve the least energy consumption in TasksetB and TasksetC. In other words, increasing SPM size is not always the best way to reduce energy consumption. This implies the possibility that SPM size should be decided appropriately for the purpose of energy minimization.

5. Conclusions

In this paper, the SPM management framework for the embedded real-time systems was proposed. Our framework gives the benefit on energy reduction in the instruction memory, and can apply to a fixed-priority based preemptive multi-task systems. In our framework, our prior approaches for SPM partitioning and code allocation are performed at the statical system design phase. Deriving optimal energy efficient usage of SPM is determined by profiler and compiler statically. RTOS and hardware modules assume a role of coordination to support runtime SPM management.

We implemented the framework to experimental environment. Experimental results showed the effectiveness of our framework. It is striking that the hybrid approach which higher priority task can preempt the spatial region of lower priority task and utilize as temporal region obtained the best result in allmost all situations. In future, we intend to extend the framework for data memory subsystems.

Acknowledgments 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., et al.: Scratchpad Memory : A Design Alternative for Cache On-chip memory in Embedded Systems, *Proceedings of the International Symposium on Hardware/Software Codesign (CODES)*, Estes Park, Colorado (2002).
3) Avissar, O., et al.: An Optimal Memory Allocation Scheme for Scratch-Pad-Based Embedded Systems, *ACM Transaction on Embedded Computing Systems (TECS)*, Vol.1, No.1, pp.6–26 (2002).
4) Steinke, S., et al.: Assigning Program and Data Objects to Scratchpad for Energy Reduction, *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, Washington, DC, pp.409–415 (2002).
5) Panda, P.R., et al.: *Memory Issues in Embedded Systems-on-Chip: Optimizations and Exploration*, Kluwer Academic Publishers, Norwell, MA, USA (1998).
6) Angiolini, F., et al.: An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning, *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.11, pp.1660–1676 (2005).
7) Janapsatya, A., et al.: Exploiting Statistical Information for Implementation of Instruction Scratchpad Memory in Embedded System, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol.14, No.8, pp.816–829 (2006).
8) Steinke, S., et al.: Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory, *Proceedings of the 15th International Symposium on System Synthesis (ISSS)*, Kyoto, Japan (2002).
9) Janapsatya, A., et al.: Hardware/Software Managed Scratchpad Memory for Embedded System, *Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '04)*, Washington, DC, pp.370–377 (2004).
10) Verma, M., et al.: Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach, *Proceedings of IEEE 3rd Workshop on Embedded System for Real-Time Multimedia (ESTIMedia)*, Jersey City, pp.115-200 (2005).
11) 高瀬英希，他：マルチタスク環境におけるスクラッチパッドメモリ領域活用法，電子情報通信学会技術研究報告，Vol.107, No.558, 屋久島，pp.109–114 (2008).
12) 高瀬英希，他：プリエンプティブなマルチタスク環境におけるスクラッチパッドメモリ領域分割法，情報処理学会研究報告，Vol.2008, No.55, 東京，pp.57–64 (2008).
13) SkyEye - Open Source Simulator. `http://www.skyeye.org/`.
14) ARM920T. `http://www.arm.com/products/CPUs/ARM920T.html`.
15) TOPPERS プロジェクト. `http://toppers.jp/`.
16) GCC, the GNU Compiler Collection. `http://gcc.gnu.org/`.
17) Wilton, S. J.E. et al.: CACTI: An Enhanced Cache Access and Cycle Time Model, *IEEE Journal of Solid-State Circuits*, Vol.31, No.5, pp.677–688 (1996).
18) EEMBC – The Embedded Microprocessor Benchmark Consortium. `http://www.eembc.org/`.
19) GLPK (GNU Linear Programming Kit). `http://www.gnu.org/software/glpk/`.