IPSJ Transactions on Advanced Computing Systems Vol.7 No.3 37–46 (Aug. 2014)

Regular Paper

Design Aid of Multi-core Embedded Systems with Energy Model

Takashi Nakada[1,a)] Kazuya Okamoto[1,†1] Toshiya Komoda[1,†2] Shinobu Miwa[1] Yohei Sato[2]
Hiroshi Ueki[2] Masanori Hayashikoshi[2] Toru Shimizu[2] Hiroshi Nakamura[1]

Received: December 6, 2013, Accepted: April 2, 2014

Abstract: Shifting to multi-core designs is so pervasive a trend to overcome the power wall and it is a necessary move for embedded systems in our rapidly evolving information society. Meanwhile, the need to increase the battery life and reduce maintenance costs for such embedded systems is very critical. Therefore, a wide variety of power reduction techniques have been proposed and realized, including Clock Gating, DVFS and Power Gating. To maximize the effectiveness of these techniques, task scheduling is a key but for multi-core systems it is very complicated due to the huge exploration space. This problem is a major obstacle for further power reduction. To cope with it, we propose a design method for embedded systems to minimize their energy consumption under performance constraints. This method is based on the clarification of properties of the above mentioned low power techniques and their interactions. In more details, we firstly establish energy models for these low power techniques and our target systems. We then explore for the best configuration by constructing an optimization problem especially for applications which have a longer deadline than the execution interval. Finally, we propose an approximate solution using dynamic programming with a lower computation complexity and compare it to a brute force explicit solution. We confirm with our evaluations that the proposed method successfully found a better configuration which reduces the total energy consumption by 32% if compared to the manually optimized configuration, which utilizes only one core.

Keywords: low power, multi-core embedded systems, energy model

1. Introduction

For developing multi-core embedded systems, it is challenging to find an optimal design from their extensive design space. The optimal design must meet deadline constraints and achieve the lowest energy consumption. Our approach here is to firstly construct energy models for low power techniques and target systems. Then we set up and solve an optimization problem by dynamic programming.

As the energy consumption of embedded systems is dominated by VLSIs, low power techniques for VLSIs are highly necessary. This energy consumption is classified into dynamic and static energy. The former is caused by switching activities of transistors and essentially consumed by computing. On the other hand, the latter is caused by leakage current and always consumed whenever power is supplied.

As technology advances static power increases more rapidly than dynamic power [1], and now it gets comparable to dynamic power consumption. As static power is consumed without any contribution to computing, its reduction is strongly required. A wide variety of power reduction techniques has been proposed and realized, including clock gating, power gating, DVFS, and so on.

To make full usage of these techniques, task scheduling is key but for multi-core systems it is very complicated due to the huge exploration space. Most previous work assumes that the deadline equals to the execution interval to simplify this problem. Under this assumption, only one execution interval should be considered and use its solution repeatedly. This strategy still satisfies the application requirements. On the other hand, this strategy limits the effectiveness of the low power techniques, due to a smaller design space. For further power reduction, it is necessary to make full usage of the original deadline. For example, sensing or video/audio streaming applications allow longer deadlines by buffering, though the throughput requirements are strict.

In this paper, we realize a method to find an optimal design that achieves the best energy efficiency under constraints such as the deadline. Our ultimate goal is to find the most energy efficient design for a periodically executed embedded system under deadline and other constraints with a reasonable cost. The design includes not only a hardware design such as core selection but also a software design such as scheduling and power management. Especially when the deadline is longer than the execution interval, we search a larger design space to make full use of the long deadline.

The remaining of this paper is organized as follows. Section 2 summarizes the background. Section 3 presents the description of the target system and the constraints of delay and energy. Then section 4 presents the design exploration strategy. Experimental

[1] The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] Renesas Electronics Corporation, Chiyoda, Tokyo 100–0004, Japan
[†1] Presently with NoConsulting
[†2] Presently with DeNA Co., Ltd.
[a)] nakada@hal.ipc.i.u-tokyo.ac.jp

© 2014 Information Processing Society of Japan

results appear in Section 5. Section 6 reviews the related work. Finally, Section 7 concludes the paper.

2. Background

2.1 Low Power Techniques

In this section, we briefly introduce major power reduction techniques including clock gating, power gating, DVFS, and so on.

Clock Gating (CG) simply cuts clock delivery from a clock oscillator. In general, the oscillator keeps running for a quick restart. Since this technique has no performance penalty, when there is no ready task, processor cores should switch to clock gating mode unless other low power techniques are applicable.

Power Gating (PG) is a promising way to reduce the static power and is used mainly in embedded systems. In modern computer system, all the components need not work all the time during computation. So far, power gating is applied in a coarse-grain manner. Recently, however, fine-grain power gating receives much attention because finer granularity increases the chances of PG. For example, Geyser-2 [2] and Geyser-3 [3] implement a fine-grained run-time PG. In these processors, PG is applied to function units (FUs). Each FU can be powered on or off instruction by instruction. In other words, instruction-level power gating is implemented in these processors. Based on this observation, there exist many chances for PG in a wide range of idle periods.

Generally speaking, coarser-grain PG can reduce more static power but have a larger transition overhead. Thus coarse-grain PG should be applied only for long idle period. For a short idle period, finer-grain PG is preferable. Therefore, there exists certain idle time between PGs. The boundary times are called Brake Even Time (BET). Dynamic Power Management (DPM) [4] manages power states of the components based on the BETs. In general, for a short idle time, the component that has a short BET should be power gated, and for a longer idle time, as much components as possible should be power gated.

DVFS has been around for more than a decade [5]. DVFS allows the voltage and the clock frequency to be decreased dynamically to trade time for energy. A lot of research is done in this area. By considering the consumed energy as a cost function, while considering deadlines as constraints, a mathematical problem can be defined and the optimal clock frequencies can be found for many kinds of real-time systems [6], [7].

For the combination of DPM and DVFS, tradeoffs between the two techniques should be considered [8]. When DVFS is used, the clock frequency is decreased to reduce the energy consumption during the execution of tasks, while the execution time increases and the idle time decreases.

Since the DVFS requires a variable voltage generator and a clock oscillator, even for modern mobile processors, the implementation cost of these components is not negligible. Additionally, the embedded processors are integrated with an analog peripheral circuit, which is more sensitive for supply voltage variation. Thus we exclude DVFS from our target system.

2.2 Dynamic Power Management

An overview of DPM techniques is given in a survey article [4]. DPM is important to reduce the static power when the processor core is in an idle state.

An example of a typical set of power modes is shown in **Table 1**. In active mode, all components are turned on. For a short idle interval, Clock Gating (CG) is preferable. The processor core can restart from the CG state instantly. In sleep mode, PG is applied to the processor core to obtain more energy reduction. To return from this mode, several clock cycles are required and some transition energy overhead is consumed. Vcc power gating is applied for a very long idle period. In this mode, the power supply is completely cut off. The only way to recover from this mode is to resume power supply. Then the processor core should follow almost the same as a power on reset procedure. Therefore, we can get the largest power reduction but both time and energy overhead become the most costly.

As mentioned in the previous section, there exist BETs between these power modes. To determine the appropriate power state, the length of next idle period is compared with these BETs. This strategy can be modeled with a function of cost with the length of the idle period. This function turns out to be piecewise-linear, increasing and concave [9].

In embedded systems, executed tasks are fixed and periodic and their scheduling is known. So, when an idle state is encountered, the time when the next task can be invoked is definitely predictable. Then the length of the idle period is also predictable. Additionally, since the restart time is predictable, wakeup overheads are easily hidden by a pre-wakeup technique. Therefore, the optimal power management is easily determined by the strategy. Finally, DPM related parameters include hardware parameters and the length of the idle period.

In this paper, to simplify the discussion, we assume only two power modes, active and sleep. It is easy to extend our approach to more power states to make full use of all power modes. Namely, after the scheduling and the length of an idle period is calculated by our scheme, the optimal sleep mode can be chosen from the length of the idle period.

2.3 Scheduling for Multi-core System

In the case of several kinds of processors executing multiple tasks, there exist many combinations of assignments.

In general more powerful processor core consumes more energy. There exists an empirical model between them called Pollack's Rule [10]. According to this model, the performance is roughly proportional to the square root of a processor's area. The static power is proportional to the area while the dynamic power is more complex and it can be regarded as being roughly proportional to the performance since switching rates differ between functional units and other parts (in general, they switch less fre-

Table 1 An example of power mode.

	Vcc	Core	Clock
Active	ON	ON	ON
Clock Gating	ON	ON	OFF
Sleep	ON	OFF	OFF
Vcc Power Gating	OFF	OFF	OFF

© 2014 Information Processing Society of Japan

quently than FUs). Therefore, using cores that are as small as possible is the best from the viewpoint of energy efficiency.

In a real scheduling problem, there are many constraints such as deadlines. To relieve this deadline constraint, we adopt pipeline scheduling. If there is only one core in the system, the total execution time of the task should be shorter than the input interval regardless of the deadline. On the other hand, if we have multiple cores and the task can be divided into subtasks, we can assign each subtask to different cores. Then, the assigned task size of each core is smaller and we can use a smaller and more energy efficient core. Theoretically, a task can be divided into the same number of subtasks as the quotient of the deadline divided by the input interval and these subtasks can then be executed on a multi-core processor having the same number of cores. However, in real systems, we should consider a parallelizing overhead such as the communication delay.

After task division and assignment to cores, we adopt *Lumped execution*. If the deadline is longer than the input interval, we can buffer several input data that are used multiple instance. Then several subtasks are executed continuously. At the same time, idle periods also appeared continuously. In other word, though the activity ratio is not changed, the length of each idle period becomes larger and power mode transition becomes less frequent. As a result, there exists a better opportunity of energy reduction.

In this paper, to simplify the discussion, we assume that all the output of each subtask should be stored in an external shared memory. Namely, all communication between cores is done via the memory.

Finally, multi-core scheduling is expressed by the following variables. Obviously, our goal is to find the optimal values for these variables.

- Number of cores
- Core and memory selection
- Task and core mapping
- Scheduling in each core

2.4 Co-optimization

For global optimization, every variable should be considered at the same time, due to their mutual dependence. For example, an optimal number of cores depends on the type of the selected processor core. The optimal processor cores depend on the task scheduling and DPM. DPM and the scheduling depend on each other.

As a result, especially for multi-core systems, the search space to find an optimal design can easily explode. To solve this problem, an efficient search method is strongly required.

3. System Description and Constraint Modeling

In this section, we describe our target system in order to find an optimal design. First, we briefly overview our target system, then explain a detailed description for each part.

3.1 Target System

Our target multi-core embedded system is shown in **Fig. 1**. This system consists of a multi-core processor, buffer memories, input device and output device. The multi-core processor consists of heterogeneous cores that support DPM.

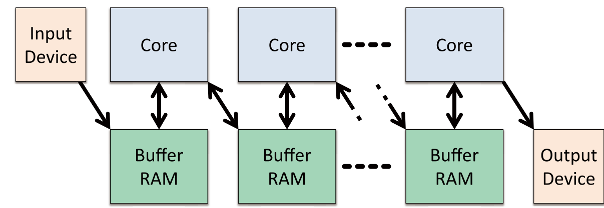


Fig. 1 Target system.

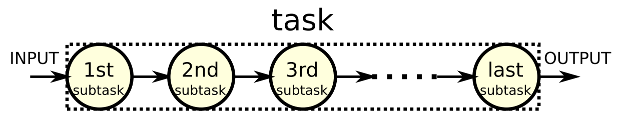


Fig. 2 Task.

For buffer memories, we assume there exists a tradeoff between the access energy and the access speed, that is, a faster memory consumes a larger energy. As mentioned in Section 1, an input device such as a sensor is integrated with a small memory controller and stores data into the buffer memory by itself. Thus the input data is periodically available in the input buffer memory.

A target application is called a task. We assume the task has already been divided into several subtasks by a programmer and the subtasks have sequential dependency from input to output. The task is invoked repeatedly. We also assume the execution time of each subtask is fixed and already measured for any candidate core.

3.2 Description

Description parameters of the target system are categorized into the following parts.

- Task
- Hardware component
- Scheduling with pipeline and lumped execution

In these parameters, the task and the hardware component parameters are given. On the other hand, to find an optimal solution of the scheduling, which is expressed with some parameters, is our goal. In other words, the task and the hardware parameters are input and the scheduling parameters are output of the multi-core system design.

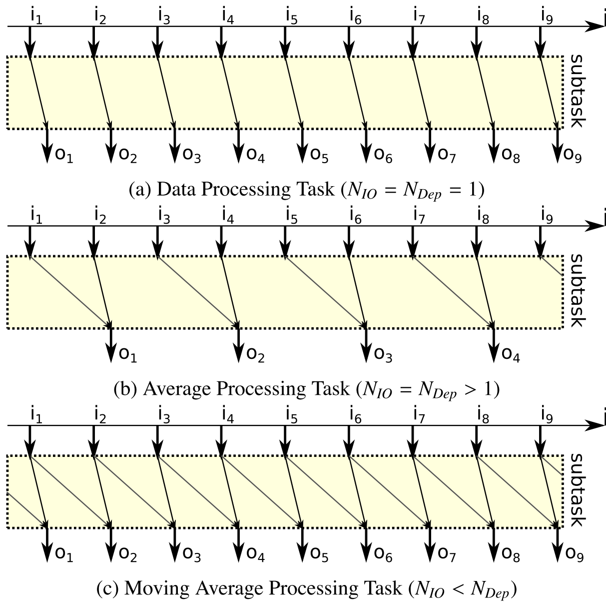
Each parameter is described in the following sections.

3.2.1 Task Description

As previously mentioned, the target task has been divided into several subtasks. The divided task is shown in **Fig. 2**. Each subtask has an ID s. The first subtask has $s = 1$, and the last subtask has $s = s_{max}$. Additionally, we defined N_{IO} and N_{Dep} to cover a wide variety of task types. N_{IO} stands for ratio of input and output throughput. N_{Dep} stands for how many sets of input data depend on an output. Thus, when both N_{IO} and N_{Dep} are equal to 1, we call this type of task *Data Processing Task* (**Fig. 3**-(a)). When both N_{IO} and N_{Dep} are equal but larger than 1, we call this type of task *Average Processing Task* (Fig. 3-(b)). When N_{Dep} is larger than N_{IO}, we call this type of task *Moving Average Processing Task* (Fig. 3-(c)).

All parameters related to the task are as follows.

© 2014 Information Processing Society of Japan



(a) Data Processing Task ($N_{IO} = N_{Dep} = 1$)

(b) Average Processing Task ($N_{IO} = N_{Dep} > 1$)

(c) Moving Average Processing Task ($N_{IO} < N_{Dep}$)

Fig. 3 Variation of task type.

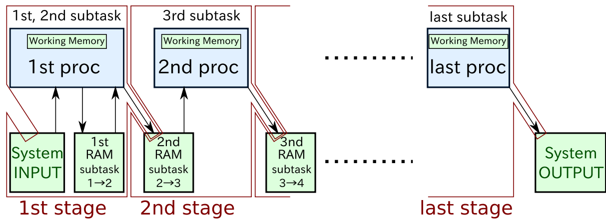


Fig. 4 Hardware model.

$\{s_n\}$ Set of subtask IDs in task

T_{gIn} Input interval of task

T_{gDl} Total deadline of task

$N_{sIO}[s]$ N_{IO} of subtask s

$N_{sDep}[s]$ N_{Dep} of subtask s

3.2.2 Hardware Description

The outline of our target hardware is shown in **Fig. 4**. We divide the total execution into several stages. Each stage is executed by one processor core. The core executes one or more than one successive subtasks. For communication between subtasks, a dedicated buffer area is assigned. We assume each stage has its own buffer memory.

All parameters related to the hardware are as follows.

$P_{pStby}[p]$ Static power of processor core p in stand-by mode

$P_{pStat}[p]$ Static power when processor core p is in active mode

$E_{pDpm}[p]$ Overhead energy when PG is applied to processor core p

$E_{pDyn}[p][s]$ Dynamic energy of a subtask s on processor core p

$T_{proc}[p][s]$ Processing time of a subtask s on processor core p

$E_{read}[r][s]$ Dynamic energy when subtask s reads its input data from memory r

$T_{read}[r][s]$ Data read latency for input data of subtask s from memory r

$E_{wrt}[r][s]$ Dynamic energy when subtask s writes its output data to memory r

$T_{wrt}[r][s]$ Data write latency for output data of subtask s to memory r

$P_{rStat}[r][s][n]$ Static power of memory r that can hold n input data of subtask s

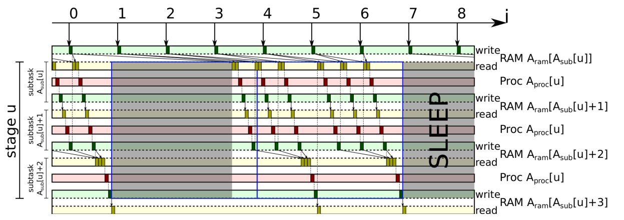


Fig. 5 Lumped scheduling.

As we mentioned, all power and execution time related parameters have to be measured in advance.

3.2.3 Scheduling Description

As we mentioned in Section 2.3, the target task is divided into several tasks and some of them are assigned for one processor core. In this section, we describe a scheduling within a core.

All parameters related to the scheduling are as follows.

u_{max} # of stages

$A_{sub}[u]$ The first subtask ID of stage u

$A_{proc}[u]$ Core ID of stage u

$A_{ram}[s]$ Memory ID for subtask s's input

$B_{eSlp}[u]$ 1: enter stand-by mode after stage u, 0: stay in active mode

$N_{Lmp}[u]$ # of instances of lumped execution in stage u

Here $A_{sub}[u]$ represents task assignment. Namely, from subtask $A_{sub}[u]$ to $A_{sub}[u+1]-1$ are assigned to stage u. $N_{Lmp}[u]$ represents the number of lumped instances. **Figure 5** shows an example scheduling when $N_{Lmp}[u] = 2$. In this example, 7 inputs are required for the lumped 2 instances. When the last input (i = 6) is available, every subtasks related to the input are executed continuously. Other preceding tasks should be scheduled earlier properly.

Since our design space exploration is done offline and its result is static, when the best configuration of these parameters are found, hardware and software configurations are statically determined. The hardware configuration is directly determined from u_{max}, $A_{proc}[u]$ and $A_{ram}[s]$. The task assignment is determined from $A_{sub}[u]$. Finally, the task scheduling is determined from $N_{Lmp}[u]$ and $B_{eSlp}[u]$. When each core wakes up and starts execution is easily and statically calculated from these scheduling parameters and the task parameters.

3.3 Modeling

In this section, we model delay constraints to guarantee a proper execution and energy constraints to find the minimum energy consumption.

Firstly, we define following intermediate variables, which can be calculated from the above parameters, for convenience.

$N_{uIO}[u]$ N_{IO} of stage u

$T_{uIn}[u]$ Input interval of stage u

$T_{uOut}[u]$ Output interval of stage u

$N_{Ram}[s]$ Required memory size of subtask s's input

Here, $N_{uIO}[u]$ is simply given by

$$N_{uIO}[u] = \prod_{k=A_{sub}[u]}^{A_{sub}[u+1]-1} N_{sIO}[k]$$

$T_{uIn}[u]$ and $T_{uOut}[u]$ are given by

© 2014 Information Processing Society of Japan

$$T_{uIn}[u] = T_{sIn}[A_{sub}[u]]$$
$$T_{uOut}[u] = T_{sOut}[A_{sub}[u+1]-1] = T_{sIn}[A_{sub}[u+1]]$$
$$\text{where} \quad T_{sIn}[s] = T_{gIn} \prod_{k=1}^{s-1} N_{sIO}[k]$$
$$T_{sOut}[s] = T_{gIn} \prod_{k=1}^{s} N_{sIO}[k] = N_{uIO}[s] \cdot T_{sIn}[s]$$

3.3.1 Delay Model

The total execution time T_{all} is given by

$$T_{all} = \sum_{u=1}^{u_{max}} T_{stg}[u].$$

Here, $T_{stg}[u]$ is the total execution time of stage u and given by

$$T_{stg}[u] = T_{wait}[u] + T_{exec}[u]$$
$$\text{where} \quad T_{wait}[u] = (N_{uIO}[u] \cdot N_{Lmp}[u] + N_{sDep}[A_{sub}[u]] - 1) \cdot T_{uIn}[u]$$
$$T_{exec}[u] = T_{pLat}[u][N_{uIO}[u]] + T_{rLat}[u][N_{uIO}[u]].$$

Here $T_{wait}[u]$ corresponds to the delay between the first input and the last input in the lumped execution in stage u. Thus, if $N_{Lmp}[u]$ is larger, $T_{wait}[u]$ also becomes longer. After the last input data is available, the final instance, which depends on the last input, is executed sequentially. Note that earlier instances have already been invoked as shown in Fig. 5. This execution time corresponds to $T_{exec}[u]$. $T_{pLat}[u][N_{uIO}[u]]$ and $T_{rLat}[u][N_{uIO}[u]]$ represent the execution times by the core and the memory access latency respectively, namely they are given by

$$T_{pLat}[u][N_{uIO}[u]] = \sum_{s=A_{sub}[u]}^{A_{sub}[u+1]-1} T_{proc}[A_{proc}[u]][s]$$
$$T_{rLat}[u][N_{uIO}[u]] = \sum_{s=A_{sub}[u]}^{A_{sub}[u+1]-1} (T_{read}[A_{ram}[u]][s] + T_{wrt}[A_{ram}[u]][s]).$$

The total deadline constraint is simply given by

$$T_{gDl} \geq T_{all} = \sum_{u=1}^{u_{max}} T_{stg}[u]. \quad (1)$$

Here, $T_{stg}[u]$ is the worst case delay of stage u. This is a necessary condition.

For each stage, the deadline constraint of stage u is given by

$$T_{uOut}[u] \geq \sum_{s=A_{sub}[u]}^{A_{sub}[u+1]-1} T_{proc}[A_{proc}[u]][s]. \quad (2)$$

If this constraint is violated, an additional delay occurs in this stage u.

3.3.2 Energy Model

The total energy consumption E_{all} is the sum of each stage's energy consumption and is given by

$$E_{all} = \sum_{u=1}^{u_{max}} E_{stg}[u] \cdot \prod_{u'=u+1}^{u_{max}} N_{uIO}[u']. \quad (3)$$

The product represents that earlier stages are executed for more instances than later stages when N_{IO} is more than 1.

Here, $E_{stg}[u]$ is the total energy consumption of stage u and is given by

$$E_{stg}[u] = E_{dyn}[u] + E_{stat}[u] + E_{dpm}[u].$$

$E_{dyn}[u]$, $E_{stat}[u]$, $E_{dpm}[u]$ correspond to the dynamic energy, the static energy and the energy overhead of DPM respectively.

$E_{dyn}[u]$ is given by

$$E_{dyn}[u] = \sum_{s=A_{sub}[u]}^{A_{sub}[u+1]-1} \Big(N_{sDep}[s] \cdot E_{read}[A_{ram}[s]][s] + E_{pDyn}[A_{proc}[u]][s] + E_{wrt}[A_{ram}[s+1]][s]\Big).$$

These clauses represent the dynamic energy of the memory read, the processor and the memory write respectively and $E_{dyn}[u]$ is their sum for all subtasks in stage u. Note that the output data is written to the next stage's memory.

$E_{stat}[u]$ is given by

$$E_{stat}[u] = \sum_{s=A_{sub}[u]}^{A_{sub}[u+1]-1} \Big(T_{uOut}[u] \cdot P_{rStat}[A_{ram}[s]][s][N_{Ram}[s]]\Big) + \Big((1 - B_{eSlp}[u]) \cdot P_{pStat}[A_{proc}[u]] \cdot T_{uOut}[u] + B_{eSlp}[u] \cdot \Big(P_{pStat}[A_{proc}[u]] \cdot T_{proc}[A_{proc}[u]][s] + P_{pStby}[A_{proc}[u]] \cdot (T_{uOut}[u] - T_{proc}[A_{proc}[u]][s])\Big)\Big).$$

The first clause represents the static energy of the memory, which is proportional to the capacity ($N_{Ram}[s]$). The second clause represents the sum of the static energy when PG is not applied and the static and the stand-by energy when PG is applied. If PG is not applied in stage u, the static power $P_{pStat}[A_{proc}[u]]$ is consumed in both active and idle periods. On the other hand, if PG is applied, the static power $P_{pStat}[A_{proc}[u]]$ is consumed only in the active period and the smaller static power $P_{pStby}[A_{proc}[u]]$ is consumed in the idle period.

$E_{dpm}[u]$ is given by

$$E_{dpm}[u] = B_{eSlp}[u] \cdot E_{pDpm}[A_{proc}[u]]/N_{Lmp}[u].$$

When PG is applied, the DPM overhead ($E_{pDpm}[A_{proc}[u]]$) is consumed every $N_{Lmp}[u]$ executions.

Our goal is to find the best configuration that minimizes E_{all} under delay constraints.

4. System Design Aid

4.1 Design Space

In the previous section, we have listed all input parameters and search variables. The search variables are u_{max}, $A_{sub}[u]$, $A_{proc}[u]$, $A_{ram}[s]$, $B_{eSlp}[u]$, $N_{Lmp}[u]$.

Finding an optimal design is equal to finding the best solution of these variables.

4.2 Brute Force Search

Since all variables are integers, it is possible to search all combinations by a brute force approach, namely, to find the best solution, which required the minimum energy while satisfying delay

© 2014 Information Processing Society of Japan

constraints, from all combinations in the search space.

When a combination of values is chosen, the total energy consumption is calculated by Eq. (3) and the delay constraints are verified by Eqs. (1) and (2).

This algorithm definitely has an exponential computation complexity. This complexity is given by

$$O(S \cdot M^{u_{max}}).$$

Here S is a number of the combination of stage selection, namely $S = (s_{max}-1)!/((s_{max}-1-u_{max})!u_{max}!)$, M is a number of the possible designs of each stage namely, M is the product of the numbers of candidate core, memory, sleep mode and upper limit of the $N_{Lmp}[u]$. The upper limit of $N_{Lmp}[u]$ is smaller than $T_{gDl}/T_{uIn}[u]$.

When s_{max} and u_{max}, which correspond to the maximum number of subtasks and cores respectively, are increased, the computation complexity is exponentially increased.

4.3 Dynamic Programming

To alleviate the huge computation complexity problem, we propose a sophisticated algorithm using dynamic programming.

First we quantize the execution time t, namely $t = n \cdot \Delta t$ $(n \in \mathbb{N})$. In this equation, Δt is the quantization step. We call each possible assignment of the variables *design* of stage u and assign ID $1, 2, \ldots, c_{max}(u)$. When searching the best design from every possible design for stage $1, \ldots, u-1$ and design $1, \ldots, c$ for stage u and the total latency is shorter than t, take the lowest energy consumption $E_c(t, u, c)$. Also, when using design c for stage u, take the total delay $T(c)$ and the energy consumption $E(c)$. Here, $T(c)$ is also quantized, namely $T(c)$ is rounded up to the nearest $n \cdot \Delta t$. Then, the following equation is established.

$$\begin{aligned} E_c(t, 0, 0) &:= 0 \\ E_c(t, u, c) &= \min(E_{cAdd}, E_{cOther}) \\ \text{where} \quad & E_{cOther} := \min_{c' < c} E_c(t, u, c') \\ & E_{cAdd} := E(c) + E_c(t - T(c), u-1, c_{max}(u-1)) \end{aligned}$$

Here, E_{cOther} corresponds to the energy when the design c is not used, and E_{cAdd} corresponds to the energy when the design c is used. Based on this equation, the optimal design of the target system is found by dynamic programming.

Specifically, from the first stage ($u = 1$), calculate $T_{stg}[u]$, $E_{stg}[u]$ for design $1, \ldots, c$. There is a tradeoff between delay and energy. From the designs that $T_{stg}[u]$ is shorter than t, the smallest $E_{stg}[u]$ is stored in $E_c(t, 1, c)$. Then move to next stage. When $E_c(T_{gDl}, u_{max}, c_{max})$ is determined, obviously, it is the minimum energy consumption with the best configuration.

As we mentioned, t is quantized, the total number of E_c is a polynomial of $T_{gDl}/\Delta t$, u_{max}, c_{max}. This restriction helps to reduce the computation complexity by providing a solution close to the optimal.

As a result, the computation complexity of this algorithm is given by

$$O(S \cdot M \cdot u_{max}/\Delta t).$$

This strategy successfully avoids the exponential computation complexity related to u_{max}. Note that S still has an exponential complexity on s_{max}.

Table 2 Parameters of processor cores.

Parameter	RX63N [11]	RX210 [12]	RL78 [13]
ID (p)	1	2	3
P_{pStby}	1.86×10^{-5} W	1.35×10^{-6} W	6.90×10^{-7} W
P_{pStat}	7.50×10^{-3} W	5.25×10^{-2} W	1.62×10^{-3} W
E_{pDpm}	1.067×10^{-4} J	4.158×10^{-4} J	2.239×10^{-7} J

Table 3 Task related parameters of DTMF.

s_n	set of subtask ID	{1, 2, 3, 4}
T_{gIn}	Input interval of task	12.5 μs
T_{gDl}	Total deadline of task	12.0 ms

Table 4 Task and core related parameters of DTMF.

		s = 1	s = 2	s = 3	s = 4
$E_{pDyn}[p][s]$	p = 1	1.56e-7	7.8e-7	3.047e-7	4.875e-9
	p = 2	1.999e-7	9.995e-7	3.905e-7	6.247e-8
	p = 3	5.693e-8	2.847e-7	1.112e-7	1.779e-9
$T_{proc}[p][s]$	p = 1	1.00e-6	5.00e-6	1.953e-6	3.125e-8
	p = 2	2.115e-6	1.058e-5	4.131e-6	6.611e-7
	p = 3	4.125e-6	2.063e-5	8.057e-6	1.289e-7

Table 5 Task and memory related parameters of DTMF.

		s = 1	s = 2	s = 3	s = 4
$E_{read}[r][s]$	r = 1	4.62e-9	9.24e-9	9.24e-9	7.22e-11
	r = 2	2.64e-9	5.28e-9	5.28e-9	4.13e-11
$T_{read}[r][s]$	r = 1	1.0e-8	2.0e-8	2.0e-8	1.56e-10
	r = 2	2.0e-8	4.0e-8	4.0e-8	3.13e-10
$E_{wrt}[r][s]$	r = 1	9.24e-9	9.24e-9	7.22e-11	3.61e-11
	r = 2	5.28e-9	5.28e-9	4.13e-11	2.06e-11
$T_{wrt}[r][s]$	r = 1	2.0e-8	2.0e-8	1.56e-10	7.81e-11
	r = 2	4.0e-8	4.0e-8	3.13e-10	1.56e-10
P_{rStat}	r = 1	6.29e-8·n	6.29e-8·n	6.29e-8·n	3.15e-8·n
$[r][s][n]$	r = 2	1.26e-9·n	1.26e-9·n	1.26e-9·n	6.29e-10·n

* r = 1: SRAM, r = 2: STT-RAM

5. Evaluation

5.1 Evaluation Setup

In our evaluation, we made both brute force and dynamic programming based search programs. These programs are parallelized, written by Haskell and compiled by "The Glorious Glasgow Haskell Compilation System, version 7.4.2." These programs are run on dual Intel(R) Xeon(R) CPU E5-2680 with 192 GB RAM.

We measured and modeled three processor cores and two kinds of memories. Major collected parameters are shown in **Table 2**. As an example of real applications, we modeled Dual-Tone Multi-Frequency (DTMF) [14]. Major parameters are shown in **Table 3**, **Table 4** and **Table 5**. Both $N_{sIO}[s]$ and $N_{sDep}[s]$ of subtasks 1, 2, 4 are 1 and 128 for subtask 3. We also use some synthetic tasks for evaluation.

5.2 Exploration Speed and Accuracy

As we mentioned, a tradeoff between the computation complexity and accuracy depends on Δt. To find the best Δt, we evaluate the optimal configuration for several Δt with a DTMF task as shown in **Table 6**.

From this result, when Δt becomes smaller, errors of energy and latency also become smaller. When Δt is 0.0003, both errors

© 2014 Information Processing Society of Japan

Table 6 Tradeoff between computation time and accuracy (u_{max} = 2).

Δt [s]	Energy [J]	Latency [s]	Exec time [s]
0.01	no configuration found		0.08
0.003	2.153e-4	1.163e-2	2.2
0.001	2.153e-4	1.163e-2	2.5
0.0003	2.146e-4	1.193e-2	4.0
0.0001	2.146e-4	1.193e-2	5.8
0.00003	2.145e-4	1.199e-2	13.2
Brute Force	2.145e-4	1.199e-2	11.3

Table 7 u_{max} vs. execution time.

u_{max}	1	2	3	4
Brute Force	0.05 s	13.1 s	*	*
Dynamic Programming (proposed)	0.04 s	4.0 s	16.0 s	24.9 s

* memory overflow

Table 8 s_{max} vs. execution time.

s_{max}	2	4	6	8	10
u_{max}	1	2	3	4	5
Brute Force	0.03 s	1.7 s	262.7 s	*	*
Dynamic Programming (proposed)	0.08 s	0.66 s	9.0 s	143.2 s	2035 s

* memory overflow

are smaller than 1%. Thus, we conclude 0.0003 (0.3 ms) is a good candidate for this size of applications and use this value in the rest of this paper. This result also shows that if Δt is small enough, the proposed algorithm can reach to the optimal solution.

Next, we evaluate the exploration speed with DTMF when u_{max} is from 1 to 4 as shown in **Table 7**. With a brute force program, we cannot get any result when u_{max} is larger than 2 due to memory overflow. In contrast, our dynamic programming based algorithm can find the optimal configuration within 25 seconds.

We also evaluate the exploration speed with a wide variaty of s_{max} for a synthetic task which consists of several subtasks whose sizes are the same. The number of subtasks (s_{max}) are varied from 2 to 10 for the evaluation. The maximum number of the cores (u_{max}) are set to half of s_{max}. Input interval T_{gIn} and Total deadline T_{gDl} are 10 ms and 100 ms respectively.

The results are shown in **Table 8**. With a brute force program, we cannot get any result when s_{max} is larger than 6 due to memory overflow. In contrast, our algorithm can still find the optimal configuration in a reasonable time. However, these results also show that the execution time inreased exponentialy with s_{max}.

From these results, we conclude that the evaluation speed is fast enough for current emmbedded systems and applications. However, a further speed upgrade is also important for more complicated systems and applications in the future.

5.3 Lumped Execution

We measured the effectiveness of a lumped execution with a simple sampling application on an RX63N (p = 1). This task consists of one subtask, which consumes 12.5 μJ per instance. We evaluate it with a variety of sensing intervals (T_{gIn}). T_{gDl} is set to 10 times of T_{gIn}.

The result is shown in **Fig. 6**. Always On and Always PG represent no lumped execution and always "On" and "PG" in idle period regardless of the length of it respectively. Lumped & PG represents lumped execution with PG. The y-axis represents the energy consumption per sensing.

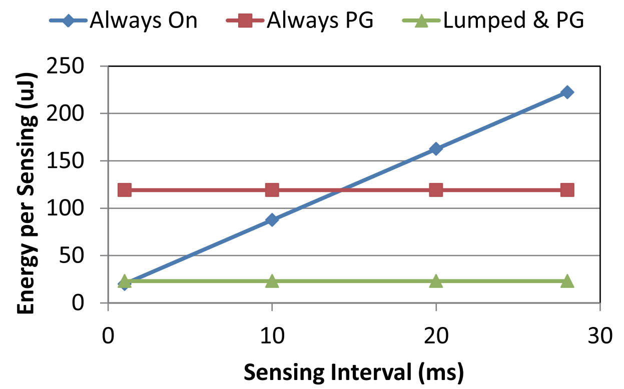


Fig. 6 Input interval vs. efficiency (sampling task).

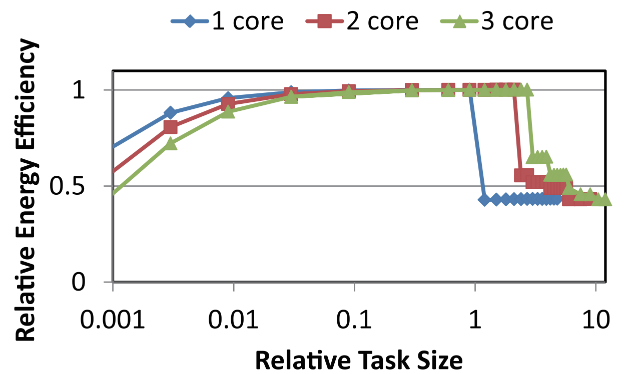


Fig. 7 Task size vs. efficiency (synthetic task).

This result shows that as the sensing interval becomes longer the energy of Always On also increases due to its large static power. On the other hand, the energy of Always PG consumes almost a constant energy by reducing the static power. However, when the sensing interval is short, the energy consumption is larger than that of Always On due to its large PG overhead.

In contrast, Lumped & PG successfully reduce the energy consumption. In this execution, 10 instances are lumped ($N_{Lmp}[u]$ =10) and the PG frequency becomes 1/10. As a result, Lumped & PG achieve 81% of energy reduction compared to Always PG.

5.4 Case Study

5.4.1 Synthetic Task

To observe the general trend, we use a synthetic task which consists of 10 subtasks whose sizes are the same. The sizes of subtasks are varied for the evaluation. And we evaluate for a wide variety of subtask size. The input interval T_{gIn} and the total deadline T_{gDl} are 10 ms and 100 ms respectively.

The result is shown in **Fig. 7**. The lines show the best configuration when u_{max} is set to 1, 2 and 3. The x-axis shows the relative task size normalized to the maximum executable size on a single smallest core (p = 3). The y-axis shows the energy efficiency that is also normalized to the energy efficiency of the smallest processor (p = 3).

This result shows that 1-core configuration is the best when the task size is smaller than 1. When the task size is very small, the static energy is not negligible. Thus, a multi-core configuration is not preferable. a 2-core configuration is the best when the task size is between 1 and 2. When the task size is larger than 1, a

© 2014 Information Processing Society of Japan

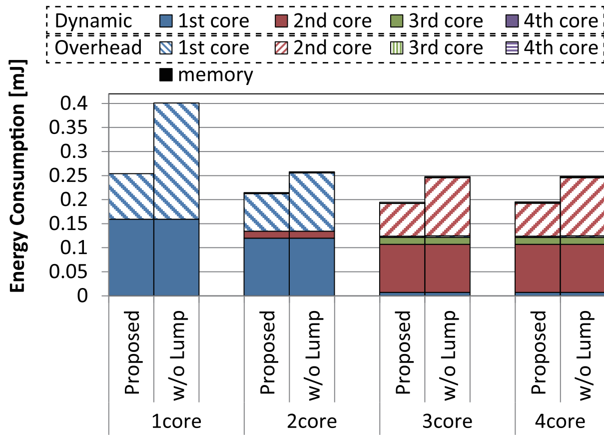


Fig. 8 Energy breakdown of DTMF.

single smallest core ($p = 3$) cannot execute this task. Thus, a bigger core, which is less efficient, should be used. On the other hand a dual smallest core can execute this task and achieve the best efficiency. Similarly, a triple smallest core is the best when the task size is more than 3.

When the task size exceeds the execution ability of the smallest core, a bigger core is chosen. Within a single core, the efficiency immediately drops to 0.43, which is the efficiency of the biggest core. In contrast, with a dual- or triple-core, the efficiency drops gradually. While the efficiency is between 1.0 and 0.43, a heterogeneous multi-core is the best configuration.

5.4.2 Real Application

We evaluate with DTMF as an example of real applications and the result is shown in **Fig. 8**. The y-axis shows the total energy consumption per output. The left bar of each pair is the result with a lumped execution and the right bar is that of without any lumped execution. In each bar, the solid parts (Dynamic) represent the dynamic energy of each core, hatched parts (Overhead) represent the static energy and power gating overhead of each core and the black part (memory) represents all memory related energy. These results show that the lumped execution successfully achieve overhead energy reduction and the total energy consumption is reduced from 17% (2core) to 37% (1core) while the dynamic energy does not change. The memory related energy is less than 1% of the total energy consumption and negligible. Meanwhile, the leftmost bar corresponds to the best configuration with a single core. We assume this single core configuration is a limit of manually optimization. Compared with this result, a 3-core configuration reduced 32% of energy consumption.

With a 4-core configuration, the energy consumption is slightly bigger than that of a 3-core configuration. This is because of a static energy overhead.

6. Related Work

The wide variety of scheduling techniques is proposed to make full usage of various low power technologies. In this paper, we focus on deadlines that are longer than input intervals and observe a lumped execution is important.

However, most previous work assumes that deadlines of tasks are equal to their input intervals. Therefore, a lumped execution is not applicable [15], [16]. One exception is Ref. [8], they lump two instances by scheduling one instance at the end of a time period and the next instance at the beginning of the next time period. However, this approach cannot lump more than two instances. Another similar approach is found in Ref. [17]. They focus on several tasks that execute on the same processor and lump multiple instances of the different tasks. Since they also assume deadlines are the same as input intervals, multiple instances of the same task cannot be lumped.

Their assumption is helpful to limit a search space and/or establish scheduling algorithms. However, from the limited search space, a limited optimal scheduling can be found.

We have solved this computation complexity problem by modeling and dynamic programming.

7. Conclusion

For developing multi-core embedded systems, it is challenging to find an optimal design from their extensive design space.

In this paper, we have proposed a design method for embedded systems to minimize its energy consumption under performance constraints. We firstly established energy models for low power techniques and our target systems. We then explored for the best configuration by constructing an optimization problem. Finally, we proposed an approximate solution using dynamic programming with a lower computation complexity and compared it to a brute force explicit solution. We confirmed with our evaluations that the proposed method successfully found a near-optimal configuration with a reasonable execution time. We also evaluated with a real application and found a better configuration which reduced the total energy consumption by 32% if compared to the manually optimized single core configuration.

We have successfully removed an exponential complexity on the number of cores, however, an exponential complexity on the number of subtasks still remains. To cope with this problem is an important future work. Additionally, to validate our model with real hardware is also an important future work.

Acknowledgments This work is supported by Normally-Off Computing Project of NEDO in Japan.

References

[1] Puri, R., Stok, L. and Bhattacharya, S.: Keeping Hot Chips Cool, *Proc. 42nd Annual Design Automation Conference* (DAC '05), pp.285–288, ACM (2005).

[2] Zhao, L., Ikebuchi, D., Saito, Y., Kamata, M., Seki, N., Kojima, Y., Amano, H., Koyama, S., Hashida, T., Umahashi, Y., Masuda, D., Usami, K., Kimura, K., Namiki, M., Takeda, S., Nakamura, H. and Kondo, M.: Geyser-2: The Second Prototype CPU with Fine-grained Run-time Power Gating, *16th Asia and South Pacific Design Automation Conference* (*ASP-DAC*), pp.87–88 (2011).

[3] Usami, K., Kudo, M., Matsunaga, K., Kosaka, T., Tsurui, Y., Wang, W., Amano, H., Kobayashi, H., Sakamoto, R., Namiki, M., Kondo, M. and Nakamura, H.: Design and Control Methodology for Fine Grain Power Gating based on Energy Characterization and Code Profiling of Microprocessors, *19th Asia and South Pacific Design Automation Conference* (*ASP-DAC*) pp.843–848 (2014).

[4] Benini, L., Bogliolo, A. and De Micheli, G.: A survey of design techniques for system-level dynamic power management, *IEEE Trans. Very Large Scale Integration* (*VLSI*) *Systems*, Vol.8, No.3, pp.299–316 (2000).

[5] Weiser, M., Welch, B., Demers, A. and Shenker, S.: Scheduling for Reduced CPU Energy, *Proc. 1st USENIX Conference on Operating Systems Design and Implementation* (OSDI '94), USENIX Association (1994).

[6] Yao, F., Demers, A. and Shenker, S.: A scheduling model for reduced

© 2014 Information Processing Society of Japan

CPU energy, *Proc. 36th Annual Symposium on Foundations of Computer Science, 1995*, pp.374–382 (1995).
[7] Huang, W. and Wang, Y.: An optimal speed control scheme supported by media servers for low-power multimedia applications, *Multimedia Systems*, Vol.15, No.2, pp.113–124 (2009).
[8] Gerards, M.E.T. and Kuper, J.: Optimal DPM and DVFS for Frame-based Real-time Systems, *ACM Trans. Archit. Code Optim.*, Vol.9, No.4, pp.41:1–41:23 (2013).
[9] Augustine, J., Irani, S. and Swamy, C.: Optimal Power-Down Strategies, *SIAM J. Comput.*, Vol.37, No.5, pp.1499–1516 (2008).
[10] Pollack, F.: Micro-32 Keynote.
[11] Renesas Electronics Corporation: *RX63N*, available from ⟨http://japan.renesas.com/products/mpumcu/rx/rx600/rx63n_631/index.jsp⟩.
[12] Renesas Electronics Corporation: *RX210*, available from ⟨http://japan.renesas.com/products/mpumcu/rx/rx200/rx210/index.jsp⟩.
[13] Renesas Electronics Corporation: *RL78*, available from ⟨http://japan.renesas.com/products/mpumcu/rl78/index.jsp⟩.
[14] Renesas Electronics Corporation: *M3S-DTMF-Tiny*, available from ⟨http://japan.renesas.com/products/tools/middleware/tiny_soft/dtmf/m3s_dtmf_tiny/index.jsp⟩.
[15] Aydin, H., Melhem, R., Mosse, D. and Mejia-Alvarez, P.: Power-aware scheduling for periodic real-time tasks, *IEEE Trans. Comput.*, Vol.53, No.5, pp.584–600 (2004).
[16] Seo, E., Jeong, J., Park, S. and Lee, J.: Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors, *IEEE Trans. Parallel and Distributed Systems*, Vol.19, No.11, pp.1540–1552 (2008).
[17] Niu, L. and Quan, G.: Peripheral-Conscious Scheduling on Energy Minimization for Weakly Hard Real-time Systems, *Design, Automation Test in Europe Conference Exhibition, 2007* (*DATE '07*), pp.1–6 (2007).

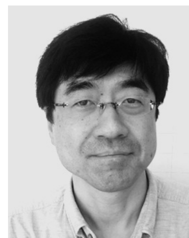
Takashi Nakada received his M.E. and Ph.D. degrees from Toyohashi University of Technology in 2004 and 2007 respectively. He has been a Project Assistant Professor at the University of Tokyo since 2012. His research interests includes Normally-Off Computing, processor architecture and related simulation technologies. He is a member of IEEE, ACM and IEICE.

Kazuya Okamoto received his M.E. degree from The University of Tokyo in 2013. He is now the President at the No-Consulting, and an adviser at the Genestream, Inc. His research interests are processor architecture and compiler.

Toshiya Komoda received his M.E. and Ph.D. degrees from The University of Tokyo in 2010 and 2014 respectively. He has been engaged in the DeNA Co., Ltd. since 2014. His research interests are processor architecture and compiler.

Shinobu Miwa received his Doctor of Informatics degree from Kyoto University in 2007. He is now an Assistant Professor at the University of Tokyo. His research interests are computer architecture, high performance computing and embedded systems. He received the Best Paper Award in 2010 Embedded System Symposium. He is a member of IEEE and IEICE.

Yohei Sato received his B.S. and M.S. degrees from The University of Aizu in 1999 and 2001 respectively. Since 2001, he has been involved in CMOS analog development department, in Hitach, Ltd. and Renesas Electronics Corporation. He developed high speed flash A/D converters and power management IPs. His current research interest is Normally-Off computing architecture.



Hiroshi Ueki received his B.S. and M.S. degrees in physics and nuclear technology from Kyoto University. Since 1991, he has been involved in microcontroller and SoC design, in Mitsubishi Electric Corporation and Renesas Electronics Corporation. He developed CPU and peripheral circuits for HDD controller, flash-memory control module for microcontroller and power management module for SD-card controller. He is now a Section Manager of power module design of System Integration Business Division in Renesas Electronics Corporation.

Masanori Hayashikoshi received his B.S. and M.S. degrees in electronic engineering from Kobe University in 1984 and 1986 respectively. In 1986, he joined the LSI Research and Development Laboratory, Mitsubishi Electric Corporation. He is currently a Chief Professional of Core Technology Business Division in Renesas Electronics Corporation. Since 1986, he has been engaged in the research and development of EEPROM's, high density DRAM's, Low power SDRAM's, embedded MRAM's for MCUs, and Normally-Off computing architecture as the challenge for further low-power solution with NVRAM.

© 2014 Information Processing Society of Japan

Toru Shimizu received his Ph.D. degree of information science from The University of Tokyo. Since 1986, he has been involved in microprocessor, micro-controller and SoC design R&D, in Mitsubishi Electric and Renesas Electronics Corporation. He developed leading-edge RISC microprocessors with embedded DRAM, micro-controllers with embedded flash memory and multi-core microprocessor. His R&D activities cover not only LSI architecture and LSI design, but embedded software and application system technologies. He is an IEEE Fellow and a senior member of IEICE.

Hiroshi Nakamura received his Ph.D. degree from The University of Tokyo in 1990. He is a Professor in the Graduate School of Information Science and Technology and a Director of Information Technology Center at The University of Tokyo. His research interests include power-efficient computer architecture and VLSI design for high-performance and embedded systems. He is now leading the "Normally-Off Computing Project" supported by NEDO/METI. He is a senior member of IEEE and ACM.

© 2014 Information Processing Society of Japan