IPSJ Transactions on System LSI Design Methodology Vol.5 133–142 (Aug. 2012)

[DOI: 10.2197/ipsjtsldm.5.133]

Regular Paper

Efficient Algorithms for Extracting Pareto-optimal Hardware Configurations in DEPS Framework

Hirotaka Kawashima[1,a)] Gang Zeng[2] Hideki Takase[1,†1]
Masato Edahiro[1] Hiroaki Takada[1]

Received: November 28, 2011, Revised: March 2, 2012,
Accepted: April 19, 2012, Released: August 6, 2012

Abstract: A dynamic energy performance scaling (DEPS) framework has been proposed as a generalization of dynamic voltage frequency scaling (DVFS). The DEPS framework selects an energy-optimal hardware configuration at runtime. To reduce runtime overhead, Pareto-optimal combinations of hardware configurations should be provided via DEPS profiling during the design phase. The challenge of DEPS profiling lies in extracting the Pareto-optimal combinations efficiently from the exponential search space. We propose two exact algorithms to reduce the number of calculations in DEPS profiling. These algorithms can be used with common search algorithms. We also propose a heuristic algorithm for searching Pareto-optimal configurations efficiently. Extensive experiments are performed, and they demonstrate that the proposed algorithms can complete DEPS profiling within a reasonable amount of time and generate optimal DEPS profiles. It is believed that the proposed algorithms will enable easy application of the DEPS framework in practice.

Keywords: energy optimization, embedded real-time system, dynamic energy performance scaling

1. Introduction

Power and energy consumption has become one of the major concerns in embedded system design. Reducing them can extend the battery lifetime of portable systems, decrease chip cooling costs, and increase system reliability. Various techniques have been proposed to optimize the energy consumption of embedded systems. Dynamic voltage and frequency scaling (DVFS) is one of the most effective techniques for energy optimization, and it has been applied to many commercial processors. However, as the feature size of very-large-scale integration (VLSI) circuits consistently shrinks, the processor power supply voltage has become so low that room for DVFS has been limited. To overcome this limitation, processors with more configuration parameters than DVFS have been proposed recently, for example, a processor whose instruction cache, as well as voltage and frequency, is dynamically reconfigurable [1].

To make best use of such processors, we have proposed dynamic energy performance scaling (DEPS) [2], a generalization of DVFS, and integrated it into a framework for energy optimization of embedded real-time applications [3]. DVFS controls only supply voltage and operating frequency for saving energy, whereas DEPS aims to control more configuration parameters to further reduce energy consumption. For dealing with various configuration parameters, DEPS generalizes a set of configuration parameters as a *configuration*. For example, for a processor with DVFS and resizable cache, a configuration consists of a voltage, a frequency, and a cache size.

However, this abstraction poses DEPS a challenge, which DEPS cannot employ existing DVFS methods. In most DVFS algorithms it is assumed that the energy is proportional to the square of the supply voltage and the execution time is inversely proportional to the clock frequency. Unfortunately, DEPS cannot calculate execution time and energy consumption via configuration (i.e., set of configuration parameters) by easy formulas. For example, neither energy nor execution time can be calculated by using the cache size for a processor with resizable cache. As a result, DEPS cannot employ most existing DVFS algorithms [2].

To further reduce energy consumption and limit runtime overhead, a DEPS framework performs both design time and runtime optimization. During design time, only the Pareto-optimal configuration sets are extracted and reserved as candidates for on-line use. The Pareto-optimal configuration sets are those configuration sets that are not dominated by any other configuration set in terms of energy consumption and performance. During runtime, the DEPS dynamically selects the optimal configuration sets from these candidates using runtime information. Specifically, the configuration is switched in an application program interface (API) function called *checkpoint*, which has been inserted into the program in advance. Although this two-stage optimization is effective, finding the Pareto-optimal configuration sets efficiently in a reasonable amount of time is an issue because the total number of configuration sets exponentially increases with the number of checkpoints.

[1] Graduate School of Information Science, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] Graduate School of Engineering, Nagoya University, Nagoya, Aichi 464–8603, Japan
[†1] Presently with Graduate School of Informatics, Kyoto University
[a)] hkawashi@ertl.jp

Calculating the optimal voltage and frequency for meeting deadlines and minimizing energy is a common problem in DVFS systems, and various methods have been proposed so far [4]. Azevedo et al. first proposed a method using a checkpoint function for voltage scheduling [5]. Xu et al. provided a unified practical approach for obtaining optimal stochastic inter-task, intra-task, and hybrid DVFS schemes [6]. Bambha et al. proposed a hybrid global/local configuration search for DVFS in embedded multiprocessor systems [7]. Xian et al. proposed voltage scheduling using statistical information of task execution time [8]. All these DVFS methods formulate the execution time and energy consumption using voltage and frequency parameters, and, based on these formulas, the optimal voltage and frequency are determined. However, these methods are not applicable to DEPS since this formulation is not realistic for DEPS.

Recently, Wang et al. conducted similar research, wherein they employed a dynamic-programming-based algorithm to search the optimal configurations for a system with DVFS and resizable cache capability [9]. Although the aim is to find the most energy efficient configuration set, our goal is to extract a Pareto-optimal configuration set. In addition, Wang et al.'s algorithm assumes a fixed execution path and makes reconfiguration decisions on the granularity of the program execution block when tasks are switched. Their method, however, lacks flexibility and cannot deal with input-data-dependent execution. Therefore, it also cannot be adopted in the DEPS framework.

As for Pareto-optimal exploration algorithms, there is some existing research, especially in the design-space exploration field. In general, the objective is to determine a Pareto-optimal design with respect to circuit area and delay. For example, a multiobjective evolutionary algorithm [10] based on the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [11] and a multiobjective simulated annealing algorithm [12] have been proposed for this purpose. Recently, Ando et al. proposed an approximate algorithm for the same problem [13]. It starts searching from two optimal points (i.e., area optimal and delay optimal), and repeats extracting Pareto-optimal points around each starting point. Basically, these Pareto-optimal search algorithms can be applied to the DEPS framework after some modifications. However, a key problem is designing a dedicated algorithm to achieve high efficiency and satisfactory quality simultaneously by taking the features of DEPS into account.

In this paper, we propose several algorithms for efficiently extracting the Pareto-optimal configuration sets as follows:

(1) *Checkpoint-interval Average Energy Consumption* (*CAEC*)
CAEC enables the possibility of calculating the average energy consumption for each checkpoint interval before the Pareto-optimal search iteration, thus reducing the number of calculations for each search iteration. CAEC can be combined with common search algorithms, and the optimality of the results is not affected after adopting CAEC.

(2) *Configuration Pruning Based on CAEC* (*CPBC*)
CPBC identifies energy-inefficient configurations based on CAEC and removes them from the search space, thus resulting in improved efficiency. CPBC can also be combined with common search algorithms without loss of optimality.

(3) *Pareto-optimal Hardware Configuration Search* (*PHCS*)
PHCS is a heuristic algorithm dedicated for the DEPS framework for efficiently searching Pareto-optimal configuration sets. Our experiments demonstrate that PHCS can achieve significantly better efficiency and quality than a dedicated genetic algorithm.

The main contribution of this paper is that the proposed algorithms and combination of algorithms can improve the search efficiency by 99.89% on average while simultaneously achieving satisfactory quality, which will enable easy application of the DEPS framework in practice.

This paper is organized as follows: An overview of the DEPS framework and the problem setup are given in Section 2. In Section 3, three algorithms for extracting the Pareto-optimal hardware configurations are presented. In Section 4, an evaluation of the proposed algorithms is given. Finally, Section 5 summarizes this paper.

2. DEPS Profile

2.1 Overview of DEPS Framework

In the DEPS framework, we assume the following conditions: A reconfigurable processor that can dynamically change its hardware configuration to provide different levels of performance and energy consumption is available. Static priority-based preemptive scheduling is employed for independent periodic task set scheduling. Hard real-time embedded application is assumed. The source code of the application programs and their typical test data set, including occurrence rate, are given in advance. In other words, the framework assumes that the application is known. The framework uses the energy and performance characteristics of the application for optimizing the energy consumption of the entire system. **Figure 1** depicts the workflow of the integrated optimiza-

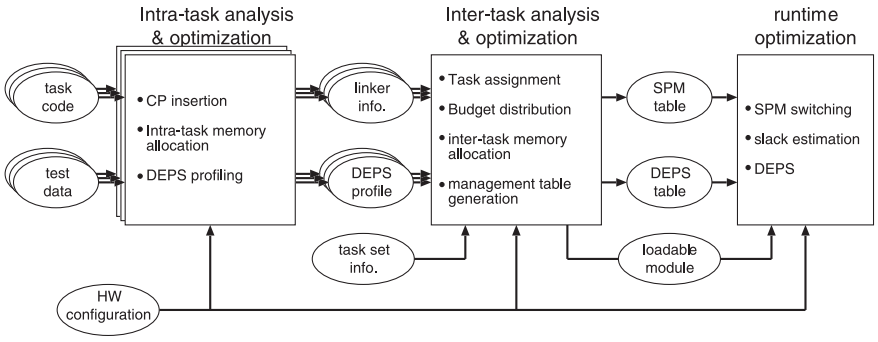

Fig. 1 Overview of our integrated energy optimization framework [3].

tion framework [3] that has been implemented with a complete toolchain and real-time operationg system (RTOS) for automatic energy optimization. The framework consists of three analysis and optimization phases. The first is an intra-task optimization phase for each task. In this phase, checkpoints are inserted into the source code first. Then, the energy-execution time information of a task under different configuration sets is extracted into a table called the *DEPS profile*. The second is an inter-task optimization phase for all tasks on one system. This phase performs system-level energy optimization by considering all task-related information such as period, deadline, etc. Specifically, execution time budgets are distributed to each task in a manner that the total system energy is minimized and all deadlines are met. The optimal configuration candidates and worst-case execution time are summarized into a DEPS management table. The third is a runtime optimization phase. We developed a DEPS-enabled RTOS as an extended version of the TOPPERS/ASP kernel [14]. The checkpoint API of the RTOS switches the configuration according to the DEPS management table and slack information detected at runtime.

2.2 Requirements for the DEPS Profile

The DEPS profile is one of the results of the intra-task analysis and optimization phase. Each task (program) will generate one DEPS profile after this process. DEPS profiling is conducted using typical test data to guarantee the generality of the DEPS profile. The DEPS profile includes the following items:

- **configuration set**: a combination of configurations at each checkpoint.
- **WCET**: the Worst-Case Execution Time of a task under a specified configuration set.
- **AEC**: the Average Energy Consumption of a task under a specified configuration set.

In this paper, a set of these items is called a *profile element*.

The WCET is necessary to guarantee the deadline of each task, and the AEC is required to minimize the average energy for all test data. Note that only a Pareto-optimal configuration set should be reserved in the DEPS profile.

Table 1 shows an example of a DEPS profile where four checkpoints have been inserted into a program, and the processor can provide a total of two configurations. All possible configuration sets and Pareto-optimal sets are shown in Table 1 (a) and Table 1 (b), respectively. Note that only Table 1 (b) is the output of DEPS profiling. For example, the configuration sets $(cfg1, cfg1, cfg1, cfg1)$, $(cfg1, cfg1, cfg1, cfg2)$, and $(cfg1, cfg1, cfg2, cfg2)$ are not Pareto-optimal because they have the same performance but consume more average energy compared to configuration set $(cfg2, cfg1, cfg2, cfg1)$.

2.3 CP-Interval Data Setup

In the DEPS framework, we estimate energy consumption and execution time by using execution trace logs generated by a cycle-accurate simulator [2], [3]. A naive approach for generating the DEPS profile is to operate this simulation with all test data and all configuration sets. Then, the execution time and the amount of energy consumed are estimated by using generated execution trace logs. Finally, the results are summarized into the DEPS profile. This approach however is not realistic because the cycle-accurate simulation is time consuming and the number of possible configuration sets is too large.

Table 1 An example of the DEPS profile.

(a) All possible profile elements

configuration set				WCET	AEC
CP0	CP1	CP2	CP3		
cfg1	cfg1	cfg1	cfg1	38.000	38.500
cfg1	cfg1	cfg1	cfg2	38.000	38.250
cfg1	cfg1	cfg2	cfg1	37.000	35.125
cfg1	cfg1	cfg2	cfg2	38.000	34.875
cfg1	cfg2	cfg1	cfg1	46.000	36.375
cfg1	cfg2	cfg1	cfg2	48.000	36.125
cfg1	cfg2	cfg2	cfg1	46.000	33.000
cfg1	cfg2	cfg2	cfg2	48.000	32.750
cfg2	cfg1	cfg1	cfg1	39.000	36.875
cfg2	cfg1	cfg1	cfg2	40.000	36.625
cfg2	cfg1	cfg2	cfg1	38.000	33.500
cfg2	cfg1	cfg2	cfg2	40.000	33.250
cfg2	cfg2	cfg1	cfg1	48.000	34.750
cfg2	cfg2	cfg1	cfg2	50.000	34.500
cfg2	cfg2	cfg2	cfg1	48.000	31.375
cfg2	cfg2	cfg2	cfg2	50.000	31.125

(b) Pareto-optimal profile elements

configuration set				WCET	AEC
CP0	CP1	CP2	CP3		
cfg1	cfg1	cfg2	cfg1	37.000	35.125
cfg2	cfg1	cfg2	cfg1	38.000	33.500
cfg2	cfg1	cfg2	cfg2	40.000	33.250
cfg1	cfg2	cfg2	cfg1	46.000	33.000
cfg2	cfg2	cfg2	cfg1	48.000	31.375
cfg2	cfg2	cfg2	cfg2	50.000	31.125

For this reason, we assume an independence relation among checkpoints, which means that the execution time and the energy consumption of a CP-interval are not affected by the configuration at the other checkpoints. The CP-interval is defined as an interval from one checkpoint to the next. We call this assumption *inter-CP independence*. With this assumption, the simulation time can be reduced significantly because we need to estimate the execution time and the energy consumed only k times for a CP-interval, where k is the number of possible configurations for a configurable processor.

Obviously, this assumption is reasonable for the DVFS function at each checkpoint. However, its effect on other configurable hardware such as the configurable cache is unclear. For this reason, we investigated the inter-CP independence on a prototype processor with configurable instruction cache [1], which is also used in our experiments. We guarantee the independence between checkpoints by adding a cache refresh operation at the beginning of each checkpoint. Our results show that the maximum overheads of the WCET and the AEC caused by this refresh are 0.06% and 0.08%, respectively. Because this overhead is negligible in most cases, in this paper, we assume inter-CP independence.

Based on this discussion, the CP-interval data can be created as follows: First, simulations are performed using a fixed hardware configuration for all checkpoints. Then, execution trace logs are split at each checkpoint, and the execution time and energy consumption of each CP-interval are estimated. Finally, the DEPS profile is generated by combining the above CP-interval data according to the given configuration sets.

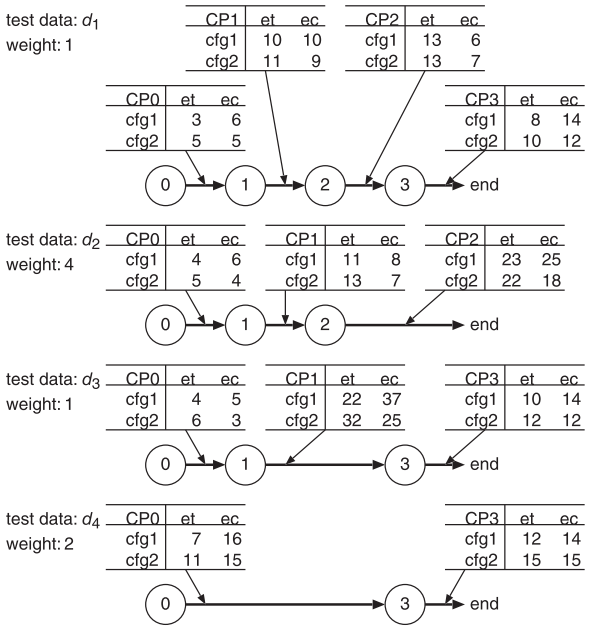

Fig. 2 Example of the CP-interval data for DEPS profiling.

We show an example of the obtained CP-interval data in **Fig. 2**. It is supposed that the processor has two hardware configurations, $cfg1$ and $cfg2$, and four checkpoints have been inserted into a program in advance. The checkpoint CP0 is always placed at the beginning of a program. The other checkpoints are inserted according to the result of the CP insertion [3]. As can be seen, the same program executes different paths and checkpoints that are dependent on given test data. Therefore, even for the same checkpoint, the CP-interval data could vary depending on the test data. For checkpoints that are not executed for a given test datum, its CP-interval datum is treated as 0. To obtain all CP-interval data in this example, only two simulations should be performed with configuration sets $(1, 1, 1, 1)$ and $(2, 2, 2, 2)$. Eventually, the DEPS profile shown in Table 1 (b) can be obtained from this CP-interval data by DEPS profiling.

We summarize all the input and output parameters for DEPS profiling as follows:

input

- $\boldsymbol{H} = \{cfg1, cfg2, \cdots, cfgk\}$: possible hardware configurations. k indicates the number of configurations.
- $\boldsymbol{D} = \{d_1, d_2, \cdots, d_m\}$: typical test data set. m indicates the number of test data.
- $weight(d)$: occurrence rate for test data d.
- $et(d, i, c)$: execution time for test data $\forall d \in \boldsymbol{D}$, the CP-interval that starts from checkpoint i and configuration $\forall c \in \boldsymbol{H}$.
- $ec(d, i, c)$: energy consumption for test data $\forall d \in \boldsymbol{D}$, the CP-interval that starts from checkpoint i and configuration $\forall c \in \boldsymbol{H}$.

output

The DEPS profile consists of the Pareto-optimal sets of the following parameters:

- $\boldsymbol{C} = (c_0, c_1, \cdots, c_{n-1})$: Pareto-optimal configuration set. c_i indicates a configuration selected at checkpoint i, and n indicates the number of checkpoints.

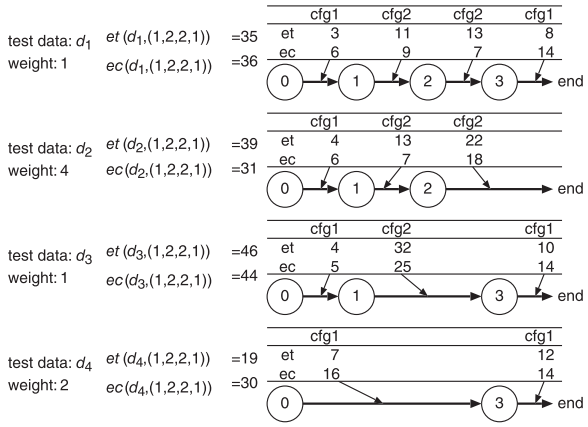

Fig. 3 Energy consumption and execution time of test data under the configuration set $\boldsymbol{C} = (1, 2, 2, 1)$.

- $wcet(\boldsymbol{D}, \boldsymbol{C})$: worst-case execution time for a test data set $\boldsymbol{D}$ and a configuration set $\boldsymbol{C}$.
- $aec(\boldsymbol{D}, \boldsymbol{C})$: average energy consumption for a test data set $\boldsymbol{D}$ and a configuration set $\boldsymbol{C}$.

2.4 Calculating WCET and AEC

Using the CP-interval data, the $wcet(\boldsymbol{D}, \boldsymbol{C})$ and the $aec(\boldsymbol{D}, \boldsymbol{C})$ for a configuration set $\boldsymbol{C}$ and a test data set $\boldsymbol{D}$ can be calculated as follows.

First, the total execution time and energy consumption of a program are calculated. For a given configuration set $\boldsymbol{C}$ and test data d, these can be calculated as the sum of the execution times and the energy consumptions of the CP-intervals as follows:

$$et(d, \boldsymbol{C}) = \sum_i et(d, i, c_i), \tag{1}$$

$$ec(d, \boldsymbol{C}) = \sum_i ec(d, i, c_i). \tag{2}$$

As an example, the execution time and the energy consumption of Fig. 2 under the configuration set $(1, 2, 2, 1)$ are shown in **Fig. 3**. For example, the execution time and the energy consumption of d_2 can be calculated as follows:

$$et(d_2, (1, 2, 2, 1)) = 4 + 13 + 22 + 0 = 39,$$
$$ec(d_2, (1, 2, 2, 1)) = 6 + 7 + 18 + 0 = 31.$$

Next, the worst-case execution time and the average energy consumption of a program are calculated. The WCET is the largest execution time of the program for a set of test data and a specific configuration set. The AEC is the average energy consumption of the program determined by considering the occurrence rate of all test data. We can calculate the WCET and the AEC using $et(d, i, c)$ and $ec(d, i, c)$ as follows:

$$wcet(\boldsymbol{D}, \boldsymbol{C}) = \mathrm{MAX}_{d \in \mathbf{D}} \sum_i et(d, i, c_i), \tag{3}$$

$$aec(\boldsymbol{D}, \boldsymbol{C}) = \frac{\sum_{d \in \mathbf{D}} \sum_i ec(d, i, c_i) weight(d)}{\sum_{d \in \mathbf{D}} weight(d)}. \tag{4}$$

As an example, we show the calculation of the WCET and the AEC for $\boldsymbol{C} = (1, 2, 2, 1)$ by using the results of Fig. 3 as follows:

$$wcet(\boldsymbol{D}, (1, 2, 2, 1)) = \max(35, 39, 46, 19) = 46,$$
$$aec(\boldsymbol{D}, (1, 2, 2, 1)) = \frac{36 \times 1 + 31 \times 4 + 44 \times 1 + 30 \times 2}{1 + 4 + 1 + 2} = 33. \tag{5}$$

As can be seen, the WCET is calculated as the largest execution time of four test data, and the AEC is calculated as a weighted average of the energy consumption of four test data.

A naive approach for extracting Pareto-optimal configuration sets is the exhaustive search. This method first calculates the WCET and the AEC for all possible configuration sets. Then, it excludes the configuration sets that are not Pareto-optimal. Finally, the remaining sets will be the DEPS profile. A problem with this approach is that the possible configuration sets could be k^n, which is exponential with the number of checkpoints. To resolve this problem, we propose three algorithms to extract Pareto-optimal configuration sets efficiently in the next section.

3. DEPS Profiling Acceleration Algorithms

In this section, three algorithms are presented for accelerating DEPS profiling. The first two methods are exact algorithms. One is used for reducing the number of calculations for the AEC, and the other is aimed at pruning the inefficient configurations. The third method is a heuristic search algorithm for extracting Pareto-optimal configuration sets. The two exact algorithms are fundamental methods, and they can be combined with the third algorithm or other search algorithms to further improve the search efficiency.

3.1 CAEC for Reducing the Number of Calculations for AEC

According to the inter-CP independence, energy consumption of a CP-interval does not depend on the configurations selected at other checkpoints. Therefore, the average energy consumption of each CP-interval can be calculated independently. We define the CAEC that starts from checkpoint i with configuration $c \in \boldsymbol{H}$ as

$$caec(\boldsymbol{D}, i, c) = \frac{\sum_{d \in \mathbf{D}} ec(d, i, c) weight(d)}{\sum_{d \in \mathbf{D}} weight(d)}. \quad (6)$$

The *caec* is calculated as the weighted average of the energy consumptions of the CP-intervals that start from the same checkpoint. Then, the AEC can be calculated efficiently using the *caec* as follows:

$$\begin{aligned} aec(\boldsymbol{D}, \boldsymbol{C}) &= \frac{\sum_{d \in \mathbf{D}} \sum_i ec(d, i, c_i) weight(d)}{\sum_{d \in \mathbf{D}} weight(d)} \\ &= \sum_i \frac{\sum_{d \in \mathbf{D}} ec(d, i, c_i) weight(d)}{\sum_{d \in \mathbf{D}} weight(d)} \\ &= \sum_i caec(\boldsymbol{D}, i, c_i). \end{aligned} \quad (7)$$

To find the Pareto-optimal configuration sets from s configuration set candidates, $s \times m \times n$ product sums are required according to Eq. (4), where m and n indicate the number of test data and the number of checkpoints, respectively. On the other hand, the CAEC method only needs $s \times n$ summations of *caec*s (Eq. (7)), and the *caec*s are precalculated with $k \times m \times n$ product sums, where k is the number of possible configurations (Eq. (6)). Therefore, a drastic reduction in processing time is possible, especially for large s such as in the exhaustive search space k^n. We use the example in Fig. 2 and **Fig. 4** to illustrate this reduction. Let $m = 4$, $n = 4$, and $k = 2$. For the exhaustive search, 16 configuration sets should be investigated, then $s = 16$. Without CAEC,

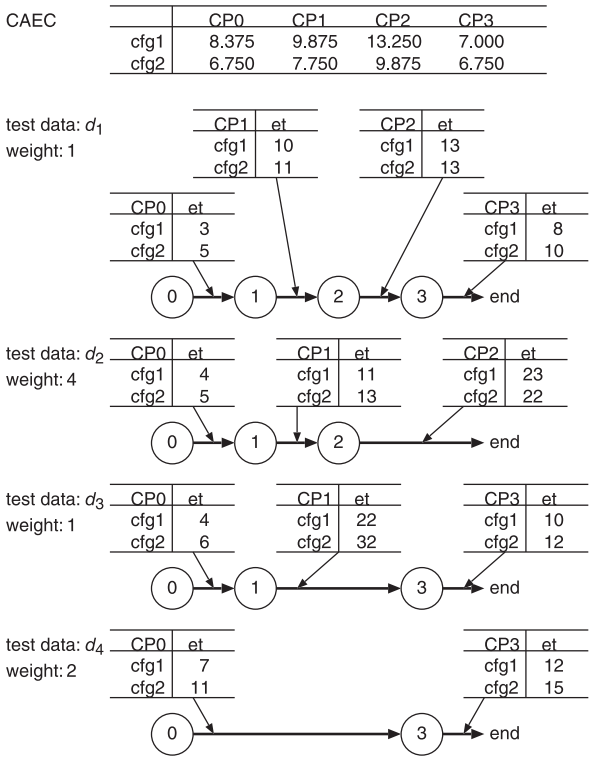

Fig. 4 CAEC and CP-interval data.

$16 \times 4 \times 4 = 256$ product sums are needed. However, with CAEC, only $2 \times 4 \times 4 = 32$ product sums and $16 \times 4 = 64$ summations are needed for calculating *caec*s and investigating configuration sets, respectively.

The *caec*s calculated for each CP-interval are shown in Fig. 4. For example, the AEC of the configuration set $(1, 2, 2, 1)$ is calculated using *caec*s as follows:

$$aec(\boldsymbol{D}, (1, 2, 2, 1)) = 8.375 + 7.750 + 9.875 + 7.000 = 33.$$

Note that this result is the same as Eq. (5). That is, the AEC result calculated using *caec* is the same as that calculated using Eq. (4). Although, CAEC is effective for AEC calculation, a similar method cannot be applied to the calculation of the WCET, since the MAX operation is required.

3.2 CPBC for Pruning the Inefficient Configurations

The idea behind Configuration Pruning Based on CAEC (CPBC) is that if a hardware configuration is identified as inefficient (i.e., it consumes more energy but with the same or worse performance than other configurations), it can be excluded from the search space, resulting in shorter processing time. However, evaluating whether a hardware configuration is efficient poses a problem.

A straightforward approach is that if both execution time and energy consumption of a configuration are larger than those of other configurations for all test data, the configuration can be identified as inefficient, because in this case the evaluated configuration is significantly inferior to others in terms of execution time and energy consumption. The inefficient configurations can be evaluated independently at each checkpoint as the inter-CP independence. Formally, a configuration $c' \in \boldsymbol{H}$ at checkpoint i is inefficient if the following conditions can be satisfied:

$$\forall d \in \boldsymbol{D},\ \exists c^* \in \boldsymbol{H},$$
$$((et(d,i,c') \geq et(d,i,c^*)) \cap (ec(d,i,c') \geq ec(d,i,c^*))). \quad (8)$$

However, these conditions are too strong to be applied in practice efficiently. Given the example in Fig. 2, there are no configurations that can be determined as inefficient according to these conditions.

To relax these conditions, CAEC can be employed. Because our objective is to minimize the average energy consumption for all test data, a configuration with higher average energy consumption and the same or longer execution time can be determined as an inefficient configuration. As a result, more inefficient configurations could be pruned by using the CAEC criterion.

Formally, a configuration $c' \in \boldsymbol{H}$ at checkpoint i is inefficient if

$$\exists c^* \in \boldsymbol{H},\ ((\forall d, et(d,i,c') \geq et(d,i,c^*))$$
$$\cap (caec(\boldsymbol{D},i,c') \geq caec(\boldsymbol{D},i,c^*))). \quad (9)$$

Considering the same example in Fig. 2, we see that the cfg1 at CP2 can be identified as inefficient compared with cfg2, because the following conditions are satisfied using the *caec* results in Fig. 4:

$$et(d_1, CP2, cfg1) \geq et(d_1, CP2, cfg2),$$
$$et(d_2, CP2, cfg1) \geq et(d_2, CP2, cfg2),$$
$$caec(\boldsymbol{D}, CP2, cfg1) \geq caec(\boldsymbol{D}, CP2, cfg2).$$

As a result, the total number of configuration sets for the search can be reduced from 16 to 8 by pruning cfg1 at CP2.

3.3 PHCS Heuristic Search Algorithm

To further improve the search efficiency, a heuristic search algorithm—the Pareto-optimal Hardware Configuration Search (PHCS) algorithm—is proposed. The complete algorithm is listed in **Fig. 5**. Different from the exhaustive search, the PHCS only conducts local search around some seeds that are Pareto-optimal configuration sets during the search process. This local search is repeated from one seed to another until no new seed can be found. The idea is to improve the search efficiency by limiting the search space on the current Pareto-optimal configuration sets. The local search ensures that, for a current Pareto-optimal configuration set, only one configuration is changed and other configurations remain unchanged. In other words, once an effective configuration has been found for a checkpoint, it could be reserved. This method is also consistent with the inter-CP independence principle. The PHCS algorithm mainly consists of two operations: one determines the seed configuration sets and the other searches the Pareto-optimal configuration sets around the seeds.

Input: CP-interval data set *intcp*
Output: DEPS profile *depsprof*
1: *depsprof* = {}
2: *initialseed* = **get_initialseed**(*intcp*)
3: *searched* = {}
4: *not_searched* = {*initialseed*}
5: **while** *not_searched* ! = {} **do**
6: *seed* = **get_nextseed**(*not_searched*)
7: *cfgsetlist* = **generate_cfgsetlist**(*seed*)
8: **for all** *cfgset* ∈ *cfgsetlist* **do**
9: *prof_element* = calculate_wcet_aec(*intcp*, *cfgset*)
10: **if** is_ParetoOptimal(*depsprof*, *prof_element*) **then**
11: append(*depsprof*, *prof_element*)
12: **end if**
13: **end for**
14: remove_nonPareto(*depsprof*)
15: *searched*.add(*seed*)
16: *not_searched* = exclude(*depsprof*, *searched*)
17: **end while**

Fig. 5 The PHCS algorithm.

First, it is necessary to decide the initial seed configuration set *initialseed* (line 2, **get_initialseed**) as the starting point of the search. We select the configuration set with the smallest AEC as the *initialseed*. Note that this *initialseed* is necessarily included in the final DEPS profile; i.e., it must be Pareto-optimal. Thus, the search efficiency is expected to be improved from a known effective configuration set instead of a random one. The *initialseed* can be found easily by selecting the configurations with the smallest *caec* at each checkpoint. For the example in Fig. 4, the *initialseed* is (2, 2, 2, 2).

Next, the algorithm searches for Pareto-optimal configuration sets around the seed. For this purpose, a list of neighbor configuration sets of *seed* is constructed in the **generate_cfgsetlist** (line 7). The list includes the configuration sets that have only one different configuration from the configuration set of the *seed*. The maximum number of generated configuration sets could be $n(k-1)$ for k possible configurations and n checkpoints. Given a seed configuration set (2, 2, 2, 2) for the example in Fig. 4, the generated *cfgsetlist* are (**1**, 2, 2, 2), (2, **1**, 2, 2), (2, 2, **1**, 2), and (2, 2, 2, **1**). If CPBC is adopted, the identified inefficient configurations will not appear in the *cfgsetlist*. For the above example, (2, 2, **1**, 2) will be removed from the list because it is inefficient according to the result of CPBC. For each configuration set in *cfgsetlist*, its corresponding WCET and AEC will be calculated in line 9. If a configuration set is determined as Pareto-optimal at this moment, it will be appended to the DEPS profile temporarily (lines 10 to 12). Some existing configurations could become non-Pareto-optimal after adding a new configuration set to the DEPS profile. For this reason, these configurations should be removed from the DEPS profile after investigating all configuration sets in *cfgsetlist* (line 14).

To avoid a duplicated search, the searched seeds are registered in the *searched* list (line 15). Also, the remaining configuration sets in the DEPS profile after excluding the searched seeds are registered in the *not_searched* list (line 16). After searching around a seed, a new *seed* will be selected from the *not_searched* by the **get_nextseed** in line 6. For the same reason as for the selection of *initialseed*, the selection of the next seed is also important. However, since it is difficult to estimate which seed will be left in the final DEPS profile, we thus consider the following three strategies for selecting the next seed in **get_nextseed**:

- **WAS (WCET Ascending Search)**: A configuration set with the smallest WCET in *not_searched* is selected as the next seed. As described above, **get_initialseed** provides the

configuration set with the smallest AEC (i.e., the largest WCET). Therefore, the relatively most distant configuration set from the initial seed is selected as the next seed.

- **WDS (WCET Descending Search)**: A configuration set with the largest WCET in *not_searched* is selected as the next seed. In contrast to the **WAS**, the relatively nearest configuration set from the initial seed is selected as the next one.
- **FFFS (First Find First Search)**: A configuration set that is found earliest is selected as the next seed. An advantage of this strategy is its simple implementation.

The PHCS algorithm will repeat the above two operations until no new seed can be found in the *not_searched*. This means that all configuration sets in the DEPS profile have been searched and no more Pareto-optimal configuration sets could be found.

4. Experimental Evaluations

4.1 Experimental Setup

The proposed DEPS profiling algorithms and an exhaustive algorithm are evaluated in terms of computation time and solution quality on benchmarks and a real application. Regarding related work, we implement a modified genetic algorithm (GA) based on the NSGA-II [11] for comparison, since a GA is a typical meta-heuristic algorithm and is suitable for exploring the Pareto-optimal configuration set for our problem.

To obtain results from the exhaustive search within a reasonable amount of time, test programs need to have a relatively small number of checkpoints. We select several benchmarks from EEMBC whose numbers of checkpoints inserted by our toolchain are less than 16 [3], [15]. Test data provided by the benchmark suite are used for evaluation, and their distribution of occurrence rate is assumed uniform.

To validate practicality, a real application is also employed for evaluation. The application is a brief version of a video-conference system using the Xvid MPEG-4 video codec and the FFmpeg library. Two tasks from the application (i.e., video encoding and decoding) are used; these are denoted as *venc* and *vdec* in the experiments. Four video data sets with different characteristics and occurrence rate are used as test data, and each data set has ten frames.

A prototype reconfigurable processor [1] with eight possible configurations is employed for the experiment. The processor can allow the frequency and voltage to be set to low or high, and can provide a resizable four-way instruction cache whose size can be set as 2k, 4k, 6k, or 8k byte by changing its associativity. A cycle-accurate simulator for the reconfigurable processor is used for generating the execution traces by running benchmarks and real programs on it. A dedicated energy estimation tool based on the energy characterization framework [16] is used for energy estimation using traces. Its estimation error is 3% on average. Note that the overhead for the configuration switch can be simulated and estimated by the simulator and the energy estimation tool, respectively. The DEPS profiling algorithms are implemented with Perl v5.10.1 and executed on a computer with a Xeon X5680 3.33-GHz processor with 65 GB of memory.

In the experiments, the following three search algorithms are evaluated:

- **EXS**: An exhaustive search algorithm for investigating all possible configuration sets. The DEPS profiles generated by **EXS** are optimal; that is, all real Pareto-optimal configuration sets can be found by this algorithm.
- **GA**: A dedicated genetic algorithm for searching energy-efficient configuration sets. The algorithm uses the same initial seed configuration set as that of the **PHCS**: One-point crossover is employed, and the crossover point is determined randomly for each pair of parents. Mutation occurs at the given rates. Only current Pareto-optimal configuration sets are selected and used as the next parent population. This algorithm does not guarantee the optimality of the obtained DEPS profiles.
- **PHCS**: The proposed heuristic search algorithm. The **PHCS** is implemented with three different strategies—**WAS**, **WDS**, and **FFFS**—for evaluation. The algorithm does not guarantee the optimality of its results owing to its heuristic feature.

Each search algorithm can be accelerated with **CAEC** or **CAEC+CPBC**. Note that **CPBC** only cannot be applied because **CPBC** depends on **CAEC**. To evaluate the effectiveness of each method, experiments are conducted with the following acceleration settings:

- *no acceleration*
- *with CAEC only*
- *with CAEC+CPBC*

Unlike the **EXS** and the **PHCS**, the **GA** requires several parameters for running, and its results are undetermined. For all **GA**-related experiments, we use three parameter sets, as shown in **Table 2**. The generation parameter is employed to control the running time of the **GA**. The mutation rate is selected from 5%, 10%, 20%, and 30%, and only the one with the best results is listed in the table. In addition, we perform the **GA** 100 times in each experiment, and only statistical results are employed for fair comparison.

Table 2 Parameters of GA-based algorithm.

benchmark	parameter set A		parameter set B		parameter set C	
	generations	mut. rate	generations	mut. rate	generations	mut. rate
canrdr01	70	30	70	30	20	10
idctrn01	120	30	120	30	50	20
pntrch01	140	30	140	30	70	5
puwmod01	170	30	170	30	50	30
rspeed01	60	30	60	30	20	30
ttsprk01	220	30	220	30	60	10
venc	300	5	300	30	150	10
vdec	300	30	300	30	100	30

4.2 Results and Discussion

The experimental results for the EEMBC benchmarks and a real application are listed in **Tables 3** and **4**. In these tables, the CPs column shows the number of inserted checkpoints, and the optimality column indicates the optimality of the generated DEPS profiles. We define "a DEPS profile is optimal" if and only if the DEPS profile is the same as that generated by the **EXS**. The "◯" mark indicates that the DEPS profile generated by the **PHCS** is optimal. For the **GA**, the optimality column indicates the probability of obtaining the optimal results. For example, 39% means the **GA** can obtain the optimal DEPS profile 39 times in 100 executions.

In Table 3, we use the parameter sets A, B, and C of Table 2 for the **GA** with *no acceleration*, *with CAEC only*, and *with CAEC+CPBC*, respectively. The generation parameters are selected in such a way that the **GA** will run for a little longer time than the **PHCS**. The purpose is to compare the optimality of the **GA** and the **PHCS** in case that they have similar computation time.

Note that the results of the **GA** in Table 3 are unsuitable for evaluating the effectiveness of **CAEC** and **CPBC** on the **GA**, since different parameter sets are employed. Alternatively, we

Table 3 Experimental results for EEMBC benchmarks and a real application.

benchmark	CPs	algorithm	no acceleration		with CAEC only		with CAEC+CPBC	
			time (sec)	optimality	time (sec)	optimality	time (sec)	optimality
canrdr01	7	EXS	76,238.365		39,059.617		11.202	
		GA	55.566	39%	29.991	39%	9.919	46%
		PHCS (WAS)	45.827	◯	23.734	◯	**6.892**	◯
		PHCS (WDS)	45.953	◯	23.633	◯	7.065	◯
		PHCS (FFFS)	49.861	◯	26.140	◯	7.632	◯
idctrn01	8	EXS	255,313.542		122,810.840		49.749	
		GA	122.531	22%	63.668	22%	25.047	4%
		PHCS (WAS)	85.315	◯	43.613	◯	16.652	◯
		PHCS (WDS)	79.964	◯	45.284	◯	**15.739**	◯
		PHCS (FFFS)	100.012	◯	55.569	◯	19.402	◯
pntrch01	8	EXS	83,819.994		43,571.593		92.179	
		GA	57.886	7%	30.705	5%	11.940	0%
		PHCS (WAS)	36.083	◯	19.446	◯	9.750	◯
		PHCS (WDS)	33.240	◯	19.076	◯	**8.726**	◯
		PHCS (FFFS)	40.825	◯	22.825	◯	10.651	◯
puwmod01	13	EXS	(could not finish within 4 days)		(could not finish within 4 days)		1,987.296	
		GA	2,079.850	98%	1,032.680	97%	270.064	40%
		PHCS (WAS)	1,657.360	◯	870.025	◯	177.860	◯
		PHCS (WDS)	1,609.047	◯	882.287	◯	**173.219**	◯
		PHCS (FFFS)	1,983.931	◯	1,021.846	◯	212.892	◯
rspeed01	5	EXS	291.915		149.479		0.832	
		GA	14.119	66%	7.544	62%	2.915	53%
		PHCS (WAS)	10.599	◯	5.888	◯	2.098	◯
		PHCS (WDS)	10.545	◯	5.498	◯	1.839	◯
		PHCS (FFFS)	10.570	◯	5.582	◯	**1.830**	◯
ttsprk01	15	EXS	(could not finish within 4 days)		(could not finish within 4 days)		778.982	
		GA	311.691	28%	160.561	30%	33.353	0%
		PHCS (WAS)	255.781	◯	132.016	◯	26.885	◯
		PHCS (WDS)	229.937	◯	119.342	◯	**23.848**	◯
		PHCS (FFFS)	235.234	◯	123.585	◯	24.724	◯
venc	9	EXS	173,103.936		93,332.111		11,728.793	
		GA	25.697	0%	23.619	1%	7.343	0%
		PHCS (WAS)	17.249	◯	8.631	◯	7.843	◯
		PHCS (WDS)	15.096	◯	8.641	◯	**6.711**	◯
		PHCS (FFFS)	18.845	◯	8.613	◯	8.533	◯
vdec	9	EXS	172,467.763		93,189.838		184.202	
		GA	22.410	43%	12.333	51%	3.761	10%
		PHCS (WAS)	7.392	◯	4.657	◯	2.284	◯
		PHCS (WDS)	7.136	◯	4.668	◯	**2.100**	◯
		PHCS (FFFS)	8.633	◯	4.656	◯	2.461	◯

Table 4 Evaluation for CAEC and CPBC on GA using parameter set A.

task	no acceleration		with CAEC only		with CAEC+CPBC	
	time (sec)	optimality	time (sec)	optimality	time (sec)	optimality
canrdr01	55.566	39%	29.991	39%	30.050	96%
idctrn01	122.531	22%	63.668	22%	65.988	43%
pntrch01	57.886	7%	30.705	5%	30.694	28%
puwmod01	2,079.850	98%	1,032.680	97%	1,117.850	100%
rspeed01	14.119	66%	7.544	62%	7.573	92%
ttsprk01	311.691	28%	160.561	30%	186.636	93%
venc	25.697	0%	14.083	0%	18.535	0%
vdec	22.410	43%	12.333	51%	13.452	88%

perform another experiment with the **GA** using the same parameter set A, the results of which are given in Table 4.

4.2.1 Evaluation for PHCS

In this section, we evaluate the **PHCS** by comparing it with the **EXS** and the **GA**. First, we compare the **PHCS** with the **EXS**. As can be seen from Table 3, although the **PHCS** is a heuristic search algorithm, it can obtain the same DEPS profile as that of the **EXS**, i.e., the optimal DEPS profile in all experiments. Except for puwmod01 and ttsprk01, the **PHCS** reduces the processing time over the **EXS** by 99.37%, 99.36%, and 57.73% on average for *no acceleration*, *with CAEC only*, and *with CAEC+CPBC*, respectively. Whereas the **EXS** with *no acceleration* cannot finish DEPS profiling for puwmod01 and ttsprk01 within 4 days, the **PHCS** with *no acceleration* can finish within 27 min and 3.8 min, respectively. It is observed from the results that the **EXS** algorithm is not practical for a program with more than 13 checkpoints. In contrast, the **PHCS** can complete profiling for benchmarks with 15 checkpoints.

Next, we compare the **PHCS** with the **GA**. As can be seen in Table 3, only for puwmod01 *with CAEC+CPBC*, the **GA** shows 98% optimality, which is comparable to the **PHCS**. However, for the other benchmarks, the optimality of the DEPS profiles generated by the **GA** are significantly inferior to that of the **PHCS**. In short, while the **PHCS** obtains the optimal results in all experiments, the **GA** can only achieve 0% to 93% optimality when it consumes similar computation time to the **PHCS**.

For the evaluation of the three strategies for searching the next seed in the **PHCS**, the best results with the shortest time are shown in boldface in the tables. As can be seen, the **WDS** can achieve the best results for most programs with only two exceptions. Therefore, the **WDS** is considered to be the best strategy, and its results are selected to represent the **PHCS** in the analysis in the rest of this paper.

4.2.2 Evaluation for CAEC and CPBC

In this section, we first discuss the effectiveness of CAEC and CPBC on the **EXS** and the **PHCS**, then analyze them on the **GA**.

We evaluate **CAEC** by comparing the results of *no acceleration* and *with CAEC only* in Table 3. The results show that *with CAEC only* can decrease the processing time by 48.2% and 44.1% on average for the **EXS** and **PHCS**, respectively, compared with *no acceleration*. Then, we discuss the contribution of **CPBC** by comparing the results of *with CAEC only* and *with CAEC+CPBC*. From the results in Table 3, it is known that *with CAEC+CPBC* can further reduce the processing time by 97.73% and 61.74% on average for the **EXS** and the **PHCS**, respectively, compared *with CAEC only*. The results also reveal that the effectiveness of the **PHCS** in reducing processing time varies from 22.34% to 80.37%, which is dependent on benchmarks.

Second, we evaluate CAEC and CPBC on the **GA**. Experimental results are given in Table 4. As mentioned earlier, the parameter set A in Table 2 is employed for the **GA** with different acceleration options. The results show that *with CAEC only* can reduce the processing time by 47.07% on average over *no acceleration*, and *CAEC+CPBC* can improve the optimality by 29.2% on average over *with CAEC only*. Meanwhile, it is also observed that *with CAEC only* cannot improve the optimality and *CAEC+CPBC* cannot reduce the processing time.

4.2.3 Summary of Evaluations

In our experiments, the **PHCS** achieves much higher efficiency than the exhaustive search algorithm while maintaining the same optimality. **CAEC+CPBC** can accelerate DEPS profiling by 98.78%, 42.45%, and 78.28% on average for the **EXS**, the **GA**, and **PHCS**, respectively, compared with *no acceleration*. In addition, adoption of *CAEC+CPBC* results in lossless optimality for the **EXS** and **PHCS** and improved optimality for the **GA**. In summary, it is observed through the experiments that the proposed algorithm combination of **PHCS (WDS)** with **CAEC+CPBC** can improve the search efficiency by 99.89% on average over an exhaustive search algorithm. Moreover, it can obtain a DEPS profile with significantly better quality than a dedicated genetic algorithm. Through our experiments, it is validated that the proposed algorithms can complete DEPS profiling in reasonable time and with satisfactory quality.

5. Conclusion

In this paper, we proposed several algorithms for efficient DEPS profiling. First, CAEC and CPBC were proposed for reducing the number of calculations of DEPS profiling. They are general methods that can be used alone or can be combined with common search algorithms without loss of optimality. Experimental results showed that they can reduce processing time by 98.78%, 42.45%, and 78.28% on average for the exhaustive, GA, and our search algorithm, respectively. Second, a heuristic algorithm, PHCS, was proposed to improve the search efficiency. This algorithm conducts a local search by taking advantage of the inter-CP independence, thus resulting in improved efficiency and high quality for DEPS profiling. In our experiments, the PHCS can improve the search efficiency by 99.89% over an exhaustive algorithm and generate optimal DEPS profiles for all benchmarks we tested and a real application. The proposed algorithms enable the DEPS framework to optimize the energy consumption of embedded systems within a reasonable amount of time and enable it to be applicable to practical system designs.

Acknowledgments This work is supported by Core Research for Evolutional Science and Technology (CREST) of the Japan Science and Technology Agency.

References

[1] Ishihara, T.: A Multi-Performance Processor for Reducing the Energy Consumption of Real-Time Embedded Systems, *IEICE Transactions*, Vol.93-A, No.12, pp.2533–2541 (2010).

[2] Zeng, G., Tomiyama, H. and Takada, H.: A Generalized Framework for Energy Savings in Hard Real-Time Embedded Systems, *IPSJ Trans. Systems LSI Design Methodology*, Vol.2, pp.167–179 (2009).

[3] Takase, H., Zeng, G., Gauthier, L., Kawashima, H., Atsumi, N., Tatematsu, T., Kobayashi, Y., Kohara, S., Koshiro, T., Ishihara, T., Tomiyama, H. and Takada, H.: An Integrated Optimization Framework for Reducing the Energy Consumption of Embedded Real-Time Applications, *ISLPED*, pp.271–276 (2011).

[4] Chen, J.-J. and Kuo, C.-F.: Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms, *Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.28–38 (2007).

[5] Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A. and Nicolau, A.: Profile-based dynamic voltage scheduling using program checkpoints, *Proc. Design, Automation and Test in Europe Conference*, pp.168–175 (2002).

[6] Xu, R., Melhem, R. and Mossé, D.: A unified practical approach to stochastic DVS scheduling, *Proc. 7th ACM & IEEE International Conference on Embedded Software*, *EMSOFT '07*, New York, NY, USA, pp.37–46, ACM (2007).
[7] Bambha, N.K., Bhattacharyya, S.S., Teich, J. and Zitzler, E.: Hybrid global/local search strategies for dynamic voltage scaling in embedded multiprocessors., *CODES '01*, pp.243–248 (2001).
[8] Xian, C., Lu, Y.-H. and Li, Z.: Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time, *Proc. 44th Annual Design Automation Conference*, *DAC '07*, New York, NY, USA, pp.664–669, ACM (2007).
[9] Wang, W., Ranka, S. and Mishra, P.: Energy-aware dynamic reconfiguration algorithms for real-time multi-tasking systems, *Elsevier Sustainable Computing: Informatics and Systems*, Vol.1, pp.35–45 (2011).
[10] Sun, F., Ravi, S., Raghunathan, A. and Jha, N.K.: A Synthesis Methodology for Hybrid Custom Instruction and Coprocessor Generation for Extensible Processors, *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Vol.26, No.11, pp.2035–2045 (2007).
[11] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. Evolutionary Computation*, Vol.6, No.2, pp.182–197 (2002).
[12] Schafer, B.C. and Wakabayashi, K.: Design space exploration acceleration through operation clustering, *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, Vol.29, No.1, pp.153–157 (2010).
[13] Ando, Y., Shibata, S., Honda, S., Tomiyama, H. and Takada, H.: Fast Design Space Exploration for Mixed Hardware-Software Embedded Systems, *Proc. Internatioanl SoC Design Conference*, pp.92–95 (2011).
[14] TOPPERS project, available from ⟨http://www.toppers.jp/en/⟩.
[15] Tatematsu, T., Takase, H., Zeng, G., Tomiyama, H. and Takada, H.: Checkpoint Extraction Using Execution Traces for Intra-Task DVFS in Embedded Systems, *The 6th International Symposium on Electronic Design, Test and Application* (*DELTA2011*), Queenstown, New Zealand, pp.19–24 (2011).
[16] Lee, D., Ishihara, T., Muroyama, M., Yasuura, H. and Fallah, F.: An Energy Characterization Framework for Software-Based Embedded Systems, *ESTImedia*, pp.59–64 (2006).

Hirotaka Kawashima received his B.E. and Master's degrees in information science from Nagoya University, Nagoya, Japan, in 2002 and 2007, respectively. He joined the Center for Embedded Computing Systems, Nagoya University, as a researcher in 2010 and is currently working toward his Ph.D. degree. His current interests include low-power system design and computer arithmetic. He is a member of IEICE.

Gang Zeng is a lecturer at the Graduate School of Engineering, Nagoya University. He received his Ph.D. degree in information science from Chiba University in 2006. From 2006 to 2010, he was a Researcher and then Assistant Professor at the Center for Embedded Computing Systems, the Graduate School of Information Science, Nagoya University. His research interests include power-aware computing and real-time embedded system design. He is a member of IEEE.

Hideki Takase received his Ph.D. degree in information science from Nagoya University in 2012. Currently, he is an Assistant Professor at Graduate School of Informatics, Kyoto University. His research interests include compilers, real-time operating systems, low energy design, and VLSI design for embedded systems. He received the Incentive Award from the Computer Science group of IPSJ in 2008.

Masato Edahiro is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He received his Ph.D. degree in computer science from Princeton University in 1999. He joined NEC Corporation in 1985, had worked in its research center for 26 years, and moved to Nagoya University in 2011. His research topics include graph and network algorithms and software for multi- and many-core processors. He is a member of IEEE, IEICE, and ORSJ.

Hiroaki Takada is a Professor at the Department of Information Engineering, the Graduate School of Information Science, Nagoya University. He is also the Executive Director of the Center for Embedded Computing Systems (NCES). He received his Ph.D. degree in information science from the University of Tokyo in 1996. He was a Research Associate at the University of Tokyo from 1989 to 1997 and was a Lecturer and then an Associate Professor at Toyohashi University of Technology from 1997 to 2003. His research interests include real-time operating systems, real-time scheduling theory, and embedded system design. He is a member of ACM, IEEE, IEICE, and JSSST.

(Recommended by Associate Editor: *Shinsuke Kobayashi*)