Received July 23, 2020, accepted August 8, 2020, date of publication August 17, 2020, date of current version August 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3017014

Evolution-Based Real-Time Job Scheduling for Co-Optimizing Processor and Memory Power Savings

HYOKYUNG BAHN, (Member, IEEE), AND KYUNGWOON CHO
Department of Computer Engineering, Ewha University, Seoul 120-750, South Korea

Corresponding author: Hyokyung Bahn (bahn@ewha.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIP) under Grant 2019R1A2C1009275, and in part by the ICT Research and Development Program of MSIP/IITP (developing system software technologies for emerging new memory that adaptively learn workload characteristics) under Grant 2019-0-00074.

ABSTRACT With the recent advances in battery-based mobile computing technologies, power-saving techniques in real-time embedded devices are becoming increasingly important. This paper presents a novel job scheduling policy for real-time systems, which aims at minimizing the power consumption of processor and memory without missing the deadline constraints of real-time jobs. To do so, we formulate the power saving techniques of processor voltage/frequency scaling and memory job placement as a unified measure, and show that it is a complex search problem that has the exponential time complexity. Thus, an efficient heuristic based on evolutionary computation is performed to cut down the huge searching space and find a reasonable schedule within the feasible time budget. To evaluate the proposed scheduling policy, we conduct experiments under various workload conditions. Our experimental results show that the proposed policy significantly reduces the energy consumption of real-time systems. Specifically, the average reduction in the energy consumption is 41.7% without deadline misses.

INDEX TERMS Real-time job scheduling, evolutionary computation, power saving, genetic algorithm, dynamic voltage/frequency scaling, deadline.

I. INTRODUCTION

Due to the recent advances in IoT (Internet of Things) and mobile computing technologies, reducing the power consumptions in battery-based real-time systems is becoming increasingly important. In this paper, we propose a novel real-time job scheduling policy that aims at minimizing the power consumption in processor and memory subsystems. Specifically, we jointly optimize the computational speed of a processor and job placement in low-power memory. To do so, we formulate the processor's dynamic voltage/frequency scaling problem and the memory job placement problem as a unified measure in order to co-optimize the power saving techniques in processor and memory.

As part of processor power-saving techniques, dynamic voltage/frequency scaling (DVFS) has been widely studied. DVFS varies the supply voltage and clock frequency based on the computation load of jobs [1]. By using DVFS, a real-time system could adjust the processor's computational speed to reduce power consumption when the load of jobs becomes less than the processor's capacity. Thus, the incorporation of DVFS into real-time job scheduling offers more flexibilities for energy-saving purposes.

Memory is another important source of power consumption in mobile embedded systems [2]. In particular, power consumption in memory increases rapidly in modern smart devices due to the ever growing size of DRAM to accommodate more applications [3]. Due to its volatile characteristics, DRAM needs constant recharge of power to maintain its data even though no read/write operation is being performed. This recharge of power, which is called the refresh operation, accounts for a significant portion of memory power consumption as the size of DRAM increases [2], [4].

Non-volatile low-power memory (LPM) technologies have caught interest as an attempt to reduce the power consumption of DRAM. LPM such as PC-RAM (phase change random access memory) and STT-MRAM (spin-transfer torque magnetic random access memory) is a byte-addressable

The associate editor coordinating the review of this manuscript and approving it for publication was Her-Terng Yau.

memory medium like DRAM but it spends less power as it is non-volatile and thus does not need refresh operations [4]–[6]. Although LPM can reduce power consumption, it cannot replace DRAM in its entirety as its access time is slower than that of DRAM [6]–[8]. Instead of total substitution, thus, we use LPM and DRAM together in order to make a tradeoff between the power consumption and the access latency of DRAM and LPM [6], [7]. Our idea is that LPM is slow but placing jobs on LPM is possible if it does not influence the scheduling possibility of the given job set. Thus, we load jobs on LPM instead of DRAM unless it incurs deadline misses, thereby saving the power consumption further. To this end, we formulate the low power techniques for processor and memory as a unified measure, and co-optimize the processor voltage/frequency scaling and memory placement of jobs (i.e., DRAM or LPM) with respect to the power-saving. By so doing, we show that power consumption in real-time systems can be further reduced without deadline misses.

Real-time job scheduling is one of the representative optimization problems. In a real-time job scheduling problem, time-critical tasks should be serviced within certain pre-assigned deadlines dictated by the physical environment. As shown in Figure 1, each job has its own release time, deadline, and the worst-case execution time. A job can start its execution after its release time, and should be serviced for the worst case execution time before its deadline. In other words, the worst case execution time of a job should be located between the release time and the deadline. Then, the scheduling problem becomes a placement problem of each job in its time slot.

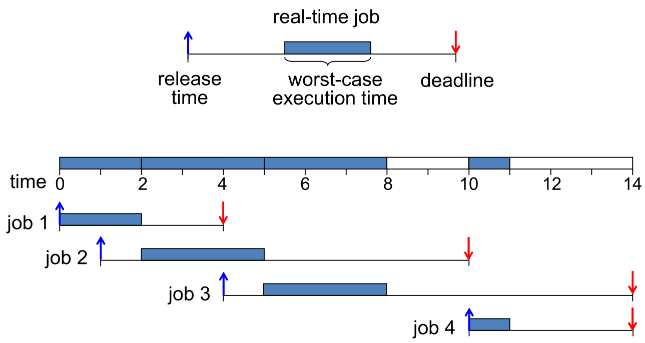


FIGURE 1. Examples of real-time jobs.

In Figure 1, there are four jobs that have their own release time and deadline, and a valid schedule that places each job in an appropriate time slot is exemplified. The problem in this paper is based on this real-time job scheduling problem, but our problem is even more complicated as the worst case execution time can be varied by changing the processor's computing speed and the memory location of jobs. In particular, the worst case execution time of each job will be increased as the processor is in the low voltage/frequency mode and/or the job is located at the slow LPM. Then, the problem becomes the optimization problem of determining the processor's voltage/frequency mode and the memory location of each job for minimizing the power consumption without missing deadlines. Note that this is a typical combinatorial optimization problem without a known method to be solved in polynomial time. That is, finding an optimal solution in an efficient way is not known, and it can only be found by enumerating all possible combinations and then evaluating them. The complexity of the problem search domain is $O(M^n V^n)$, where n is the number of jobs, M is the number of memory types, and V is the number of processor's voltage/frequency modes. That is, for each job n, there are V possible voltage/frequency modes and M possible memory locations, which should be determined, and as the configurations of the processor and the memory are independent, the number of possible combinations is $M^n V^n$. If one could afford to evaluate the cost (i.e., power consumptions of processor and memory) of each solution, then one would do well to select the solution of the lowest cost without deadline misses. For example, there are 20 jobs to schedule, and if the number of voltage/frequency modes is 4 and the number of memory types is 2, all possible schedules reaches 8^{20}, which is more than 10^{18}. Even with high-end server systems, it is not feasible to scan all these cases to find one with the minimum cost within a given time budget. Thus, we need an algorithm that can find a good, but possibly not optimal, solution in an efficient way. To cope with this situation, we use an evolutionary computation method based on genetic algorithms. To evaluate the proposed scheduling policy, we conduct experiments under various workload conditions. Our experimental results show that the proposed policy significantly reduces the power consumption of real-time systems. Specifically, the average reduction in the energy consumption is 41.7% without deadline misses. The contributions made in this paper can be summarized as follows.

- First, unlike traditional real-time job models that only consider the execution in the processor, we define an extended job model to consider the memory configuration of jobs as well as the processor side. Specifically, in our job model, the definition of a job includes the size of the memory footprint and the number of memory read/write operations on that job. Based on this, the worst-case execution time of a job is re-evaluated by reflecting the read/write characteristics of the memory medium the job resides.
- Second, we propose a hybrid memory architecture consisting of DRAM and LPM for real-time systems. Unlike composing DRAM and LPM hierarchically, we present both DRAM and LPM at the same main memory level, managing them under a single address space. Note that general-purpose systems usually compose DRAM and LPM as a hierarchical architecture to improve the virtual memory system performances, but this is difficult in real-time systems as page faults cannot be predicted beforehand, making the deadline guaranteed service difficult. Thus, in our architecture, we set the memory size large enough not to incur unexpected page

faults, but focusing on the reduction of DRAM's power consumptions.

- Third, our model tightly evaluates the scaled worst-case execution time of a job, considering the overlapped latency between processor and memory. That is, the scaled worst case execution time of a job is determined by the slower time component of executing instructions in processor and accessing memory. Thus, we can reduce the power consumption of real-time systems further by adopting LPM without influencing scheduling possibilities.
- Fourth, we design a genetic algorithm that aims at minimizing the power consumption in processor and memory with a constraint that the deadlines of all jobs are satisfied. To co-optimize the power consumption in processor and memory without deadline misses, we define our cost function as the total energy consumption that would be yielded by the processor and memory states the solution represents; and we add some penalty cost to a solution if the solution does not meet the deadline of jobs. This avoids too much discrimination and searches the wide area of problem space, not leading to premature convergence to a local optimum.

The remainder of this paper is organized as follows. Section II describes the problem model of real-time job scheduling with respect to the processor and memory power-saving. In Section III, the optimization technique based on genetic algorithms is presented. Section IV presents the performance evaluation results to assess the effectiveness of the proposed policy. In Section V, we briefly summarize some related studies of this paper. Finally, Section VI concludes this paper.

II. PROBLEM MODEL

A. JOB MODEL

Suppose that $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is the set of real-time jobs, and the target system has a processor whose voltage/frequency level can be adjusted dynamically, and main memory consists of DRAM and LPM as shown in Figure 2. A job τ_i is represented by $< t_i, p_i, m_i >$, where t_i is the worst case execution time of τ_i with the default voltage/frequency mode of a processor and DRAM memory placement, p_i is the period of τ_i, and m_i is the memory configuration of τ_i, which is defined as $< s_i, r_i, w_i >$, where s_i is the size of τ_i's memory footprint, and r_i and w_i are the number of memory read and write operations, respectively, during the execution of τ_i. As real-time jobs can usually be modeled by periodic jobs, we only consider a periodic job, of which the period implicitly determines the deadline of a job.

By following the common assumptions in previous work [1], we make the six assumptions in our real-time system and job model.

A1. We consider the independent job model, in which a job does not affect other jobs.

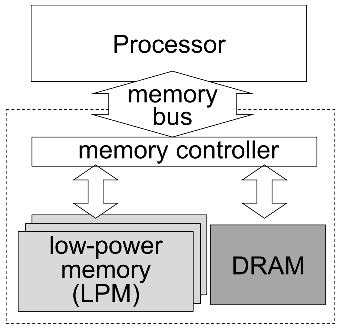


FIGURE 2. The target architecture of the proposed policy.

A2. Although the size of DRAM is set to the entire footprint of all jobs, we partially hibernate DRAM to save the memory power consumption if jobs in DRAM move to LPM.

A3. The overhead of processor's context switching from one job to another is negligible.

A4. The overhead of changing the processor's voltage/frequency mode from one to another is negligible.

A5. A job's relative deadline is equal to its period.

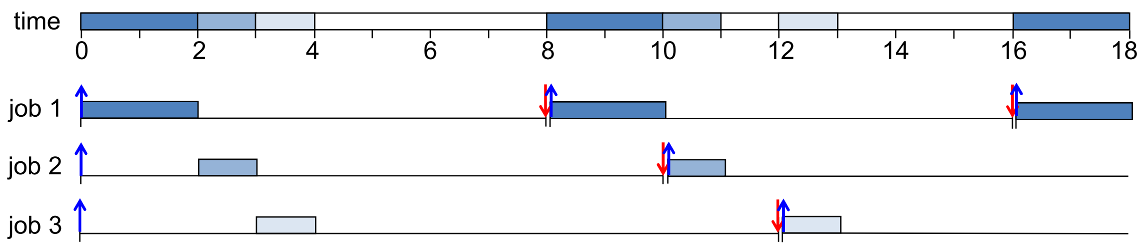
A6. A job can be preempted during its execution.

In our job model, the worst case execution time t_i of a job is determined by the slower component of processor and memory with the given voltage/frequency mode and the memory medium. In particular, as t_i should consider the longest time path between memory and processor, our model applies a function f that scales t_i for considering DVFS and LPM. During this process, we tightly estimate the latency that may overlap between processor and memory, leading to minimized power consumption. The schedulability of real-time job set Γ in our model is tested by the utilization U of a processor as follows.

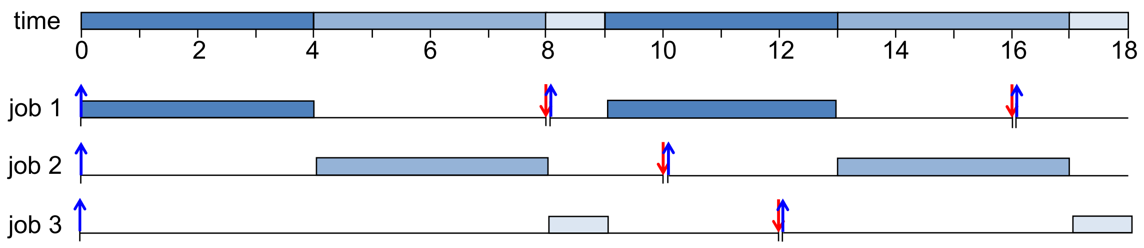
$$U = \sum_{i=1}^{n} \frac{f(t_i)}{p_i} \leq 1 \qquad (1)$$

Once a real-time job set passes the schedulability test, we can determine the execution order of the jobs based on the earliest deadline first (EDF) algorithm, which is known to schedule real-time jobs without missing their deadlines if any feasible schedule exists [1]. Note that EDF schedules the job with the nearest deadline first.

Let us now see an example situation consisting of three jobs τ_1, τ_2, and τ_3, whose worst case execution times t_1, t_2, and t_3 are 2, 1, and 1, respectively, and their periods are 8, 10, and 12, respectively. The schedulability of the jobs can be tested by calculating the utilization of the three jobs, i.e., $U = 2/8 + 1/10 + 1/12 = 0.433$. Since the utilization $U < 1$ is satisfied, the jobs are schedulable. Figure 3(a) depicts the scheduling result for this example. Although the jobs are schedulable, the scheduling result incurs a large proportion of idle intervals. This inefficiency can be relieved by adjusting the processor's voltage/frequency for some idle intervals. For example, if two low frequency levels of 0.5 and



(a) Scheduling result without the proposed policy.



(b) Scheduling result by the proposed policy.

FIGURE 3. **Comparison of the scheduled results.**

0.25 are applied for jobs τ_1 and τ_2, respectively, the scaled worst case execution time $f(t_1)$ and $f(t_2)$ will be 4 and 4, respectively. As a result, the utilization of the processor increases to $U = 4/8 + 4/10 + 1/12 = 0.983$, in which $U < 1$ is still satisfied and is, thus, schedulable. Moreover, if we locate τ_2 in LPM whose access latency is twice that of DRAM, one may think that the scaled worst case execution time $f(t_2)$ will be 8, and thus it is not schedulable as $U = 4/8 + 8/10 + 1/12 = 1.383 > 1$. However, we tightly evaluate the scaled worst case execution time considering the overlapped latency between processor and memory. That is, the scaled worst case execution time $f(t_i)$ of a job τ_i is determined by the slower time component of executing instructions in processor and accessing memory as follows.

$$f(t_i) = max\{f_{DVFS}(t_i), f_{LPM}(t_i)\} + \mu \tag{2}$$

where μ is a stall factor that limits the overlapped execution between processor and memory in the given architecture, and $f_{DVFS}(t_i)$ and $f_{LPM}(t_i)$ are the scaled worst case execution time of τ_i by applying DVFS and LPM, respectively, which can be subsequently defined as follows.

$$f_{DVFS}(t_i) = t_i/\varepsilon \tag{3}$$

$$f_{LPM}(t_i) = t_i^*(\alpha^* r_i + \beta^* w_i) \tag{4}$$

where ε is the voltage/frequency mode parameter for the given processor state, and α and β are the read and write latency parameters for the given memory type.

Thus, in the aforementioned example, the worst case execution time $f(t_2)$ will be still 4 and the utilization of the processor by adopting both DVFS and LPM becomes less than 1, still being schedulable. Figure 3(b) depicts the scheduling result with our model when the aforementioned voltage/frequency scaling and memory placement are used. As can be seen, idle intervals almost disappear in comparison with the result given in Figure 3(a), which will lead to a significant reduction in power consumptions.

B. ENERGY POWER MODEL

In this paper, we analyze the energy consumptions of processor and memory separately and then accumulate them. The energy consumption E is evaluated as

$$E = E_{\text{CPU}} + E_{\text{MEM}} \tag{5}$$

where E_{CPU} is the processor energy consumption andE_{MEM} is the memory energy consumption. Note that we do not consider other resources such as cache memory. The processor energy consumption E_{CPU} can be modeled as the sum of active energy consumption $E_{\text{CPU\_act}}$ and static energy consumption $E_{\text{CPU\_stat}}$ [35], [36], that is

$$E_{\text{CPU}} = E_{\text{CPU\_act}} + E_{\text{CPU\_stat}} \tag{6}$$

The energy consumption of CMOS processors is dominated by active energy, which mainly results from charging and discharging of gates in the circuits, and can be formulated as a function of supply voltage and operating frequency [35], [36], that is

$$E_{\text{CPU\_act}} = \sum_{\tau_i} c\ V_i^2\ f_i\ t_i \tag{7}$$

where c is the effective capacitance, V_i is the supply voltage for executing job τ_i, f_i is the operating frequency for executing job τ_i, and t_i is the actual time to execute job τ_i.

In our model, the voltage V_i is adjusted according as the clock frequency f_i is varied. It is known that the clock frequency and the supply voltage have almost linear relations, but for real processors, the supply voltage may not decrease linearly with lowered clock frequency [35]. Thus, we use a real processor model, Transmeta model [35], [37] as listed in Table 1. The static energy consumption of a processor $E_{\text{CPU\_stat}}$ mainly results from the leakage current and is expressed as

$$E_{\text{CPU\_stat}} = N_{\text{gate}} * V_i * I_{\text{leak}} * T \tag{8}$$

where N_{gate} is the number of gates, V_i is the supply voltage, I_{leak} is the leakage current, and T is the total execution time.

TABLE 1. Frequency and supply voltage of Transmeta Crusoe.

f_i (MHz)	200	233	300	400	500
V_i (V)	1.10	1.15	1.20	1.23	1.35
f_i (MHz)	600	700	800	900	1000
V_i (V)	1.53	1.75	2.00	2.35	2.80

The memory energy consumption E_{MEM} can be modeled as the sum of active energy consumption $E_{\text{MEM\_act}}$ and static energy consumption $E_{\text{MEM\_stat}}$ [38], [39], that is

$$E_{\text{MEM}} = E_{\text{MEM\_act}} + E_{\text{MEM\_stat}} \tag{9}$$

The active energy consumption $E_{\text{MEM\_act}}$ refers to the energy dissipated while a read or a write operation is performed [40], [41], which can be calculated as

$$\begin{aligned} E_{\text{MEM\_act}} = &\sum_{\tau_i \in \text{DRAM}} \{r_i * E\_read_{\text{DRAM}} + w_i * E\_write_{\text{DRAM}}\} \\ &+ \sum_{\tau_i \in \text{LPM}} \{r_i * E\_read_{\text{LPM}} + w_i * E\_write_{\text{LPM}}\} \end{aligned} \tag{10}$$

where r_i and w_i are the number of read and write operations on the job τ_i, respectively, E_read_{DRAM} and E_write_{DRAM} refer to the read and write energy consumptions for a word size in DRAM, respectively, and E_read_{LPM} and E_write_{LPM} refer to the read and write energy consumptions for a word size in in LPM, respectively.

The static energy consumption $E_{\text{MEM\_stat}}$ refers to the energy consumed consistently regardless of any operations in memory [41], which can be modeled as

$$\begin{aligned} E_{\text{MEM\_stat}} = &\sum_{\tau_i \in \text{DRAM}} \{Unit\_static\_power_{\text{DRAM}} * s_i * T\} \\ &+ \sum_{\tau_i \in \text{LPM}} \{Unit\_static\_power_{\text{LPM}} * s_i * T\} \end{aligned} \tag{11}$$

where $Unit\_static\_power_{\text{DRAM}}$ is the static power of DRAM per capacity including leakage power and refresh power, $Unit\_static\_power_{\text{LPM}}$ is the static power of LPM per capacity including leakage power, s_i is the aligned size of job τ_i on the given memory type, and T is the total execution time. The leakage power refers to the power consumed even when the memory is idle. Note that the leakage power of LPM is very small in comparison with that of DRAM. The refresh power refers to the power consumed to recharge cells to retain the stored data. As DRAM memory cells store data in small capacitors that lose their charge over time, it needs considerable refresh power to sustain refresh cycles irrespective of read and write operations. Note that LPM does not need the refresh power as it is a non-volatile medium.

C. EXTENDING THE BASIC MODEL

In this subsection, we discuss how our basic model can be extended to more general cases. In general, real-time jobs cannot necessarily be modeled by periodic jobs. For example, a sporadic job or an aperiodic job can be included in a real-time job set, where a sporadic job is similar to a periodic job but its period is not regular and an aperiodic job has no deadline. If our job set Γ is extended to include sporadic jobs, the period p_i of a sporadic job can be defined as the minimum inter-arrival time of the job. Then, the same utilization test in Expression (1) can be applied as if it is a periodic job. An aperiodic job can also be modeled like a periodic job as we can create some periodic server to handle the execution of aperiodic jobs, which is periodically activated and executes aperiodic jobs.

In our basic model, we assume that a job's relative deadline is equal to its period. In the case that a job's deadline is not equal to its period, our utilization test in Expression (1) is not a safe schedulability test, and a more complex test is necessary. Specifically, the demand bound function should be considered for the schedulability test of a real-time job set instead of the utilization function [31], [32]. Then, the schedulability test of Expression (1) should be replaced by the following demand bound function test.

$$\forall t (0 \le t \le p), \sum_{\tau_i \in \Gamma} dbf\ (\tau_i, t) \le t \tag{12}$$

where p is the lowest common multiple of all jobs' periods and $dbf(\tau_i, t)$ denotes the maximum amount of time executed by job τ_i before time t. More details of the demand bound function test can be found in the study of Baruah *et al.* [31].

III. OPTIMIZATIONS WITH GENETIC ALGORITHMS

In this section, we describe the details of our genetic algorithms to optimize the dynamic voltage/frequency scaling of a processor and job placement between DRAM and low-power memory with respect to the minimization of power consumption. As this is a typical combinatorial optimization problem without a known method to be solved in polynomial time, we use an optimization technique based on genetic algorithms. Genetic algorithm is a stochastic algorithm, which mimics the natural evolution of population genetics in problem solving or simulation [9]–[15].

In this paper, the problem is determining the processor's voltage/frequency mode and the memory location of each job for minimizing the energy consumption with the constraints of each job's deadline. To this end, our genetic algorithm maintains a certain number of candidate solutions, and evolves the solution set to obtain the best solution that decides the processor and memory configurations. Specifically, our genetic algorithm first generates an initial solution set; then it selects two solutions in the set and merges as one solution by the crossover and mutation operations; then the solution set is evolved by substituting a solution in the old set with the newly generated solution. Such processes continue until the solution set converges. Finally, the best solution in the converged set is chosen to determine the voltage/frequency mode and the memory placement of each job.

A. ENCODING

In a genetic algorithm, a solution is typically encoded by a linear string. The choice of this encoding is a major feature of a genetic algorithm. In our problem, a solution should determine the memory allocation and processor voltage/frequency states of all real-time jobs in the system. Thus, our encoding represents a solution by two strings as shown in Figure 4. The first string represents the processor state and the second string represents the memory allocation information to service the jobs. The length of each string is set to the number of jobs in the system and each position within a string represents the processor and memory states of a job. That is, the value of the entry in the first and the second string refers to the state of the processor voltage/frequency level and the memory allocation for the corresponding job, respectively.

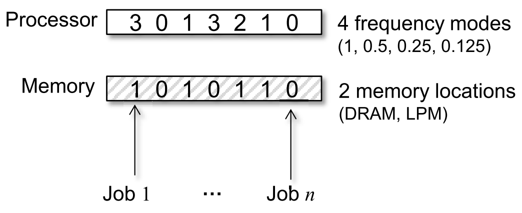


FIGURE 4. **Encoding of the problem.**

For example, if there are four frequency modes and two memory types, each entry in the processor string can have the value of 0, 1, 2, or 3, and each entry in the memory string can have the value of 0 or 1. As shown in Figure 4, the value of 0, 1, 2, and 3 in the processor string represents the frequency mode of 1, 0.5, 0.25, and 0.125, respectively, and the value of 0 and 1 in the memory string indicates that the location of the job is DRAM and LPM, respectively. Note that Job 1 is placed on LPM and it is executed under the processor's frequency mode of 0.125. Based on this encoding method, we generate 100 random solutions as the initial solution set.

In a genetic algorithm, we need a cost function for measuring the quality of each solution. As our goal is minimizing the energy consumption without missing the deadlines of the real-time jobs, we define our cost function as the total energy consumption that would be yielded by the processor and memory states the solution represents; and we add some penalty cost to a solution if the solution does not meet the deadline constraint of the jobs, which is checked through the utilization test explained in Expression (1).

B. SELECTION OPERATIONS

A selection operation selects two solutions in the current set that will be the parents of the next generation. It is based on a probabilistic rule that favors better solutions based on their fitness measure. In our problem, the fitness measure refers to the energy consumption of the solution. However, this operation should avoid too much discrimination. That is, if there exist one or two extremely superior solutions in terms of the energy consumption, the selection pressure may focus narrowly on these extreme ones, and the characteristics from these solutions may dominate the solution space rapidly, potentially causing premature convergence to a local optimum. To resolve this issue, we normalize the fitness measure of the solutions so that the best solution in the set is 4 times more probable to be chosen than the worst one. This is a standard normalization method frequently used in genetic algorithms [16]. The normalized fitness measure of a solution is defined as

$$NV_i = (R_w - R_i) + (R_w - R_b)/3 \tag{13}$$

where R_w, R_b, and R_i are the rankings of the worst solution, the best solution, and solution i, respectively, in the current set. Based on the normalized fitness measure, a roulette wheel selection is used as a selection operator, which assigns each solution i a slot with the size equal to its normalized fitness measure NV_i. The pointer of the wheel is a random real number ranging from 0 to $\sum NV_i$. A solution whose slot spans the pointer is chosen and becomes a parent.

C. CROSSOVER AND MUTATION OPERATIONS

After selecting two parent solutions, a crossover operation is performed. The idea of the crossover operation, which is regarded as the most important search operator in genetic algorithms, is that useful segments of the selected parents should be combined to yield a child that will lead to better solutions as time progresses.

A classical 1-point crossover operation works by randomly generating a crossover point and then a child is generated by copying the left part of a parent and the right part of the other parent. In our problem, solutions are encoded as two strings. Thus, we generate crossover points for the two strings independently to perform the crossover operation. Figure 5 depicts an example of the proposed crossover operation. A random number between 1 and n–1, where n is the number of jobs, is generated and a new solution is formed by inheriting the left and the right segments of parents 1 and 2, respectively.

Mutation perturbs some parts of the child solution generated by crossover. The mission of mutation is to search wide area of the problem space and avoid premature convergence. We use a classical mutation operator, which selects some random part of a solution and changes it to a random value [17].

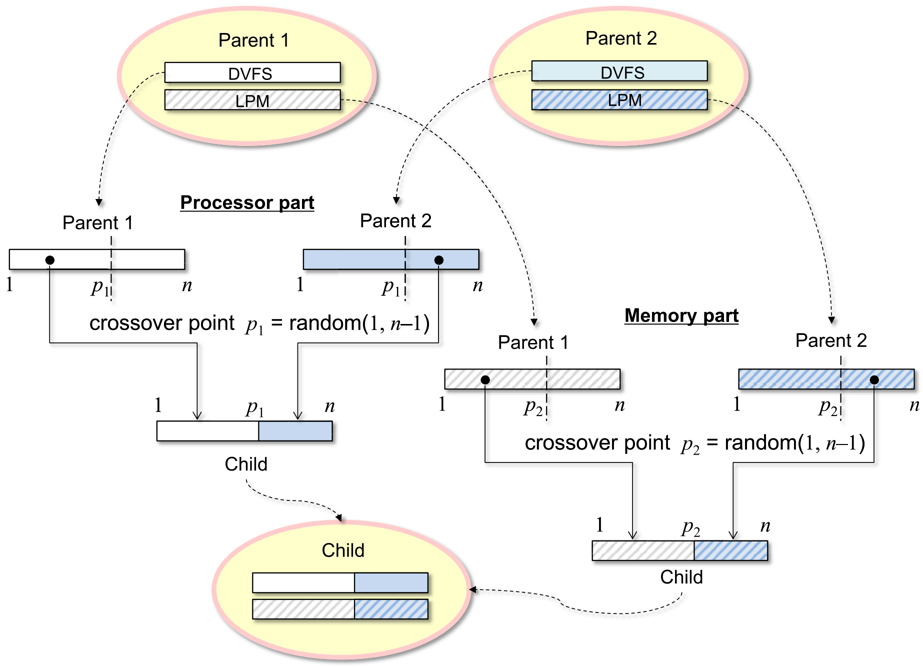


FIGURE 5. Crossover operations in the proposed scheduling algorithm.

D. REPLACEMENT OPERATIONS

When a child solution is generated, a new solution set is produced by replacing a solution in the current generation by the child solution. In this paper, we substitute the solution that incurs the highest cost in the current set with the newly generated child, which is the most commonly used replacement operation in genetic algorithms.

E. STOP CONDITIONS

There are several ways to set the stop condition of genetic algorithms. The two most common methods are (1) to set the maximum number of iterations and (2) to stop the loop when the diversity of solutions in the set falls below a certain level [16], [17]. To analyze the diversity, it is common to compare the fitness values of all solutions in the current set, and check whether they are homogeneous. We use the second method to ensure the convergence of our genetic algorithms.

It is not an easy matter to prove the complexity of the genetic algorithm theoretically as it is sensitive to the experimental parameters of the genetic algorithm. However, in empirical aspects, we found that our genetic algorithm safely converges with a constant number of generations regardless of the number of jobs (up to 1,000 jobs we experimented), and thus the complexity of our genetic algorithm can be considered as $O(1)$.

IV. PERFORMANCE EVALUATIONS

We compare our policy called COOP-GA (Co-optimization with genetic algorithms) with ORIGINAL, DVFS, LPM, and DVFS-LPM, where ORIGINAL is a baseline condition that adopts neither dynamic voltage/frequency scaling nor low-power memory technologies, DVFS uses dynamic voltage/frequency scaling but does not use low-power memory, LPM uses low-power memory but does not use dynamic voltage/frequency scaling, and DVFS-LPM uses both dynamic voltage/frequency scaling and low-power memory. Similar to COOP-GA, DVFS-LPM uses both dynamic voltage/frequency scaling and low-power memory technologies, but its optimization is performed simply by the hill climbing method instead of genetic algorithms.

Before showing the comparison results, Figure 6 plots the evolution of COOP-GA as time progresses. That is, the fitness measure of the best and the worst solutions among 100 solutions in the current searching space and their average are

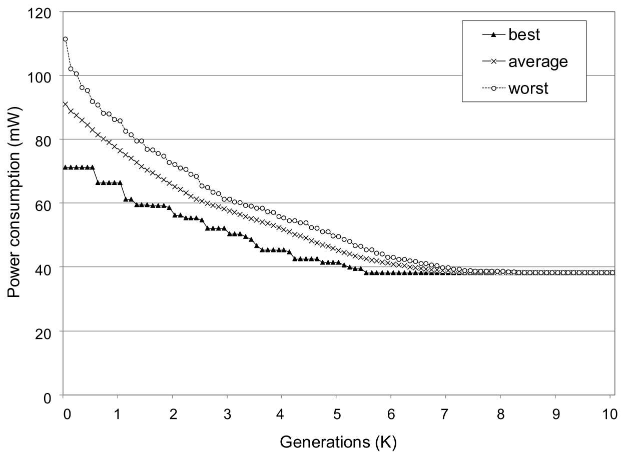


FIGURE 6. Convergence of the solutions as time progresses.

plotted as the generation evolves. As we see in the figure, the quality of solutions improves significantly according to the evolution of the generation, and finally they converge.

In our experiments, the sizes of DRAM and LPM are equally set to accommodate the full job set. Table 2 shows the power consumption and the read/write latency of DRAM and PC-RAM, which is a type of LPM we experimented. Theoretically, we can set the unlimited number of different voltage/frequency levels of a processor. However, commercial processors usually allow a certain limited number of voltage/frequency modes for practical reasons. By considering this, we set 4 frequency levels of 1, 0.5, 0.25, and 0.125.

TABLE 2. Characteristics of LPM and DRAM.

	LPM	DRAM
Read latency	100 (ns)	50 (ns)
Write latency	350 (ns)	50 (ns)
Read energy	0.2 (nJ/bit)	0.1 (nJ/bit)
Write energy	1.0 (nJ/bit)	0.1 (nJ/bit)
Static power	0.1 (W/GB)	1 (W/GB)

We perform experiments under both synthetic and realistic workload conditions. In the synthetic workload, the experiments were performed with four different workload conditions, where the utilizations of the workloads are 0.25, 0.5, 0.75, and 0.95, assuming the default system configuration (i.e., the utilization value under the full voltage/frequency mode of a processor and DRAM only placement). The number of jobs in our synthetic workload is set to 100 and the worst-case execution times of jobs are randomly generated between 1ms and 500ms. The average period of the workload is determined based on the target utilization of the workload (i.e., 0.25, 0.5, 0.75, and 0.95) and the period of each job is randomly generated between 0.5 to 1.5 times of the average period. For realistic workloads, we use the Robotic Highway Safety Marker (RSM) workload [33] and the IoT workload [34]. Tables 3 and 4 depict the parameters of the RSM and IoT workloads, respectively [33], [34].

A. SYNTHETIC WORKLOADS

Figure 7 shows the energy consumption of ORIGINAL, DVFS, LPM, DVFS-LPM, and COOP-GA as the utilization of the workload is varied. The numbers on the y-axis mean the energy consumption of the algorithm normalized to ORIGINAL. That is, the energy consumption of the ORIGINAL is set to 1.0 and the relative value scaled to ORIGINAL is plotted. As we see in the figure, COOP-GA performs the best for all cases regardless of the workload conditions. The reduced energy consumption of COOP-GA is 41.7% on average and up to 73.8% in comparison with the ORIGINAL system. Specifically, COOP-GA performs even better when the utilization becomes small. This is because there are more possibilities of resource optimization when the workload is not heavy. As the load of jobs increases and the utilization becomes close to 1, it is more likely that

TABLE 3. Parameters of the RSM workload.

Task	Period	WCET
Serial	7.8125ms	100us
Length	7.8125ms	1ms
Way Point	23.4375ms	2.5ms
Encoder	23.4375ms	350us
PID	23.4375ms	1.06ms
Motor	23.4375ms	250us

TABLE 4. Parameters of the IoT workload.

Task	Period	WCET
Sense Temperature	100ms	10us
Send data to server	1min	6ms
Sense Vibration	10ms	600us
Compress and send	1s	7.5ms
Get info. & calc.	10ms	1ms
Control machine	10ms	1ms
Update GUI	1s	20ms

low-power techniques incur deadline misses of jobs, difficult to find better solutions. When we compare the five schemes, COOP-GA reduces the energy consumption of ORIGINAL, DVFS, LPM, and DVFS-LPM by 41.7%, 29.7%, 37.9%, and 24.0%, on average, respectively. DVFS performs better than LPM, which implies that reducing the energy consumption in a processor is more effective than that in memory. However, DVFS-LPM performs better than DVFS and LPM, implying that adopting the two low-power techniques together makes even better results.

Figures 8 and 9 separately depict the energy consumption in processor and memory, respectively, for the five policies as the utilization is varied. As can be seen in Figure 8, DVFS, DVFS-LPM, and COOP-GA, which adopt dynamic voltage/frequency scaling, reduce a substantial amount of processor's energy consumption. However, COOP-GA performs even better than the others by optimization with the genetic algorithm. The energy consumption of LPM and ORIGINAL is high as they do not use voltage/frequency scaling, although the gap is small as the utilization becomes large. This is because voltage/frequency scaling is less effective as the load of jobs approaches the full capacity of a processor. That is, the chance of utilizing idle periods of a processor by lowering the supply voltage/frequency is difficult in these cases. When comparing DVFS and DVFS-LPM, DVFS performs better than DVFS-LPM with respect to the processor's energy consumption. This is because low-power memory is slow, and thus the execution time of the processor is also increased, which incurs additional energy consumption in the processor.

When we compare the energy consumption in memory, policies that use low-power memory significantly reduces the energy consumption. As shown in Figure 9, DVFS-LPM, LPM, and COOP-GA consume less energy than DVFS and ORIGINAL, which adopt only DRAM memory.

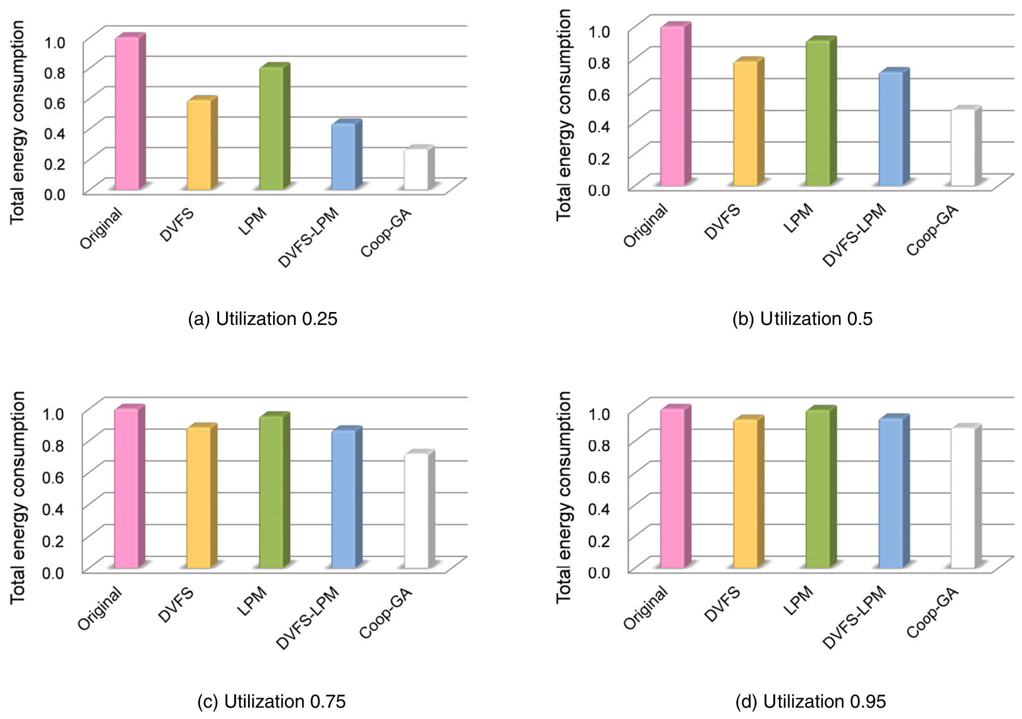


FIGURE 7. Total energy consumptions as the utilization is varied (synthetic workload).

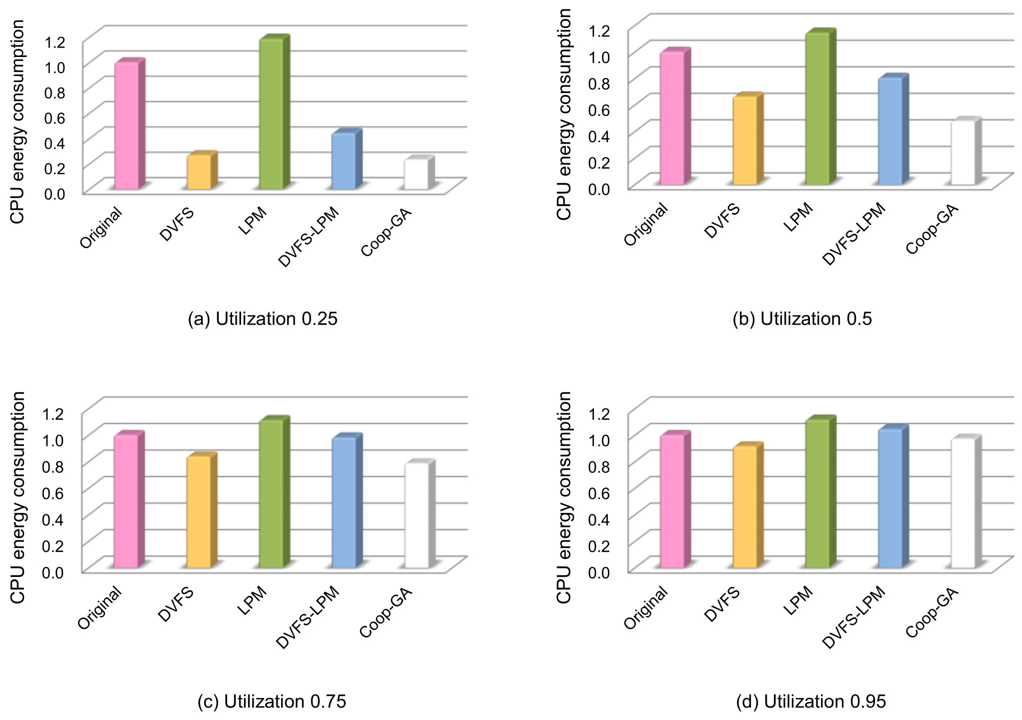


FIGURE 8. Energy consumptions in processor as the utilization is varied (synthetic workload).

The reason is that the static energy consumption of LPM is very small, and thus adopting less DRAM by partially substituting it with low-power memory saves the refresh power of DRAM significantly. Specifically, policies adopting low-power memory, i.e., LPM, DVFS-LPM, and COOP-GA, consume 50-70% less energy than ORIGINAL and DVFS.

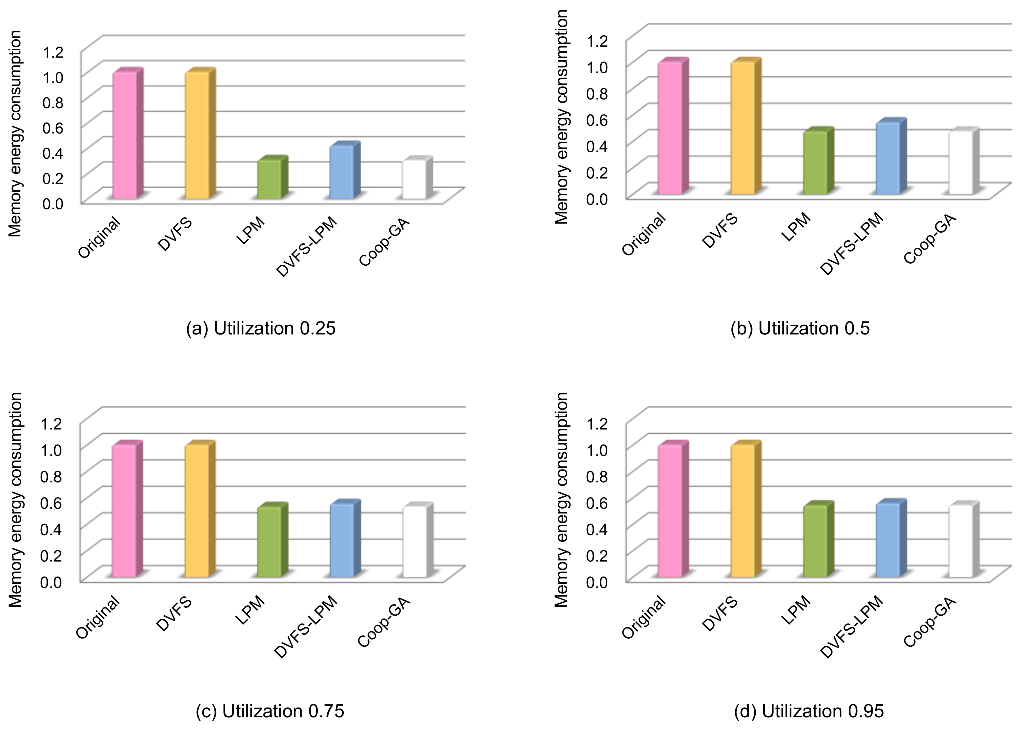


FIGURE 9. **Energy consumptions in memory as the utilization is varied (synthetic workload).**

However, as the access latency of low-power memory is slow, placing a job on low-power memory may degrade the execution time in the processor, possibly increasing processor's energy consumption. This can be seen in Figure 8 that LPM spends more energy than ORIGNAL, and DVFS-LPM also spends more energy than DVFS. However, the gap is not wide and can be compensated by the energy-saving effect in memory. Also, COOP-GA does not degrade the energy-savings in processor although it adopts low-power memory. This is because energy-savings can be maximized by co-optimizing the processor's low voltage/frequency mode and placing jobs on low-power memory. Another interesting result is that the effectiveness of the low-power memory technique is less influenced by the load of workloads. That is, low-power memory techniques are still effective even when the utilization is close to 1 as shown in Figure 9(d), which is different from the processor voltage/frequency scaling cases.

Figures 10 and 11, respectively, show the active and static energy consumptions. Although COOP-GA performs slightly worse than DVFS in terms of active energy consumption in some cases, it significantly performs better than the other four policies in static energy consumption, leading to the minimized total energy consumption. In active energy consumption, we can observe the significant effect of dynamic voltage/frequency scaling as shown in Figure 10. Lowering the voltage/frequency of a processor is effective in active energy consumption because energy consumption in the CMOS digital circuits is proportional to the square of the supply voltage. Such effects can be seen apparently when the utilization of the workload is low. This is because there are more chances to lower the supply voltage in the idle periods of a processor. Meanwhile, using low-power memory slightly increases the active energy consumption as shown in Figure 10. The reason is that low-power memory simulated in our study needs more energy than DRAM when performing read/write operations as shown in Table 2.

Now, let us see the static energy consumption. As can be seen in Figure 11, the relative effect of dynamic voltage/frequency scaling and low-power memory is similar with respect to static energy consumption, and combining the two techniques obtains even better results. Dynamic voltage/frequency scaling is effective in static energy saving as the idle intervals of a processor can be reduced by lowering the supply voltage/frequency. Also, low-power memory is effective in static energy saving because power consumptions by refresh operations can be reduced by substituting for DRAM.

B. REALISTIC WORKLOADS

To investigate the efficiency of COOP-GA in more realistic conditions, we simulate additional experiments under two realistic workload situations, Robotic Highway Safety Marker (RSM) workload [33] and IoT workload [34].

Figure 12 depicts the total energy consumption when RSM and IoT workloads are used. Similar to synthetic workload cases, COOP-GA reduces the energy consumption of real-time systems significantly. Specifically, the trend of the graphs resembles the synthetic workload case with the utilization of 0.25. COOP-GA reduces the energy consumption

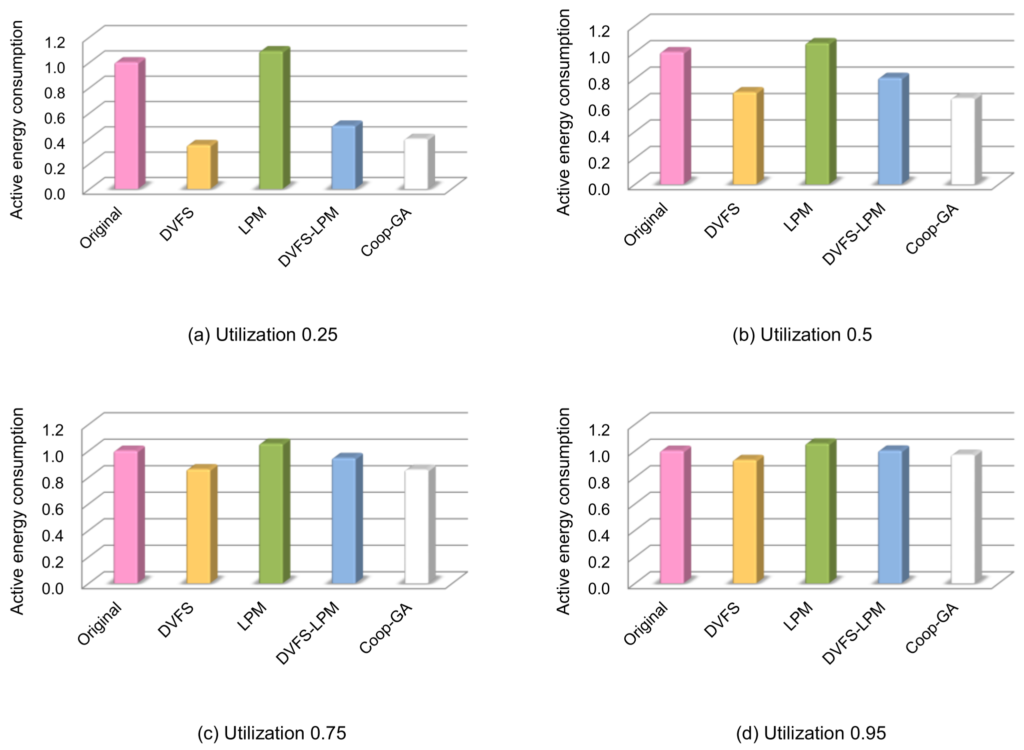


FIGURE 10. Active energy consumptions as the utilization is varied (synthetic workload).

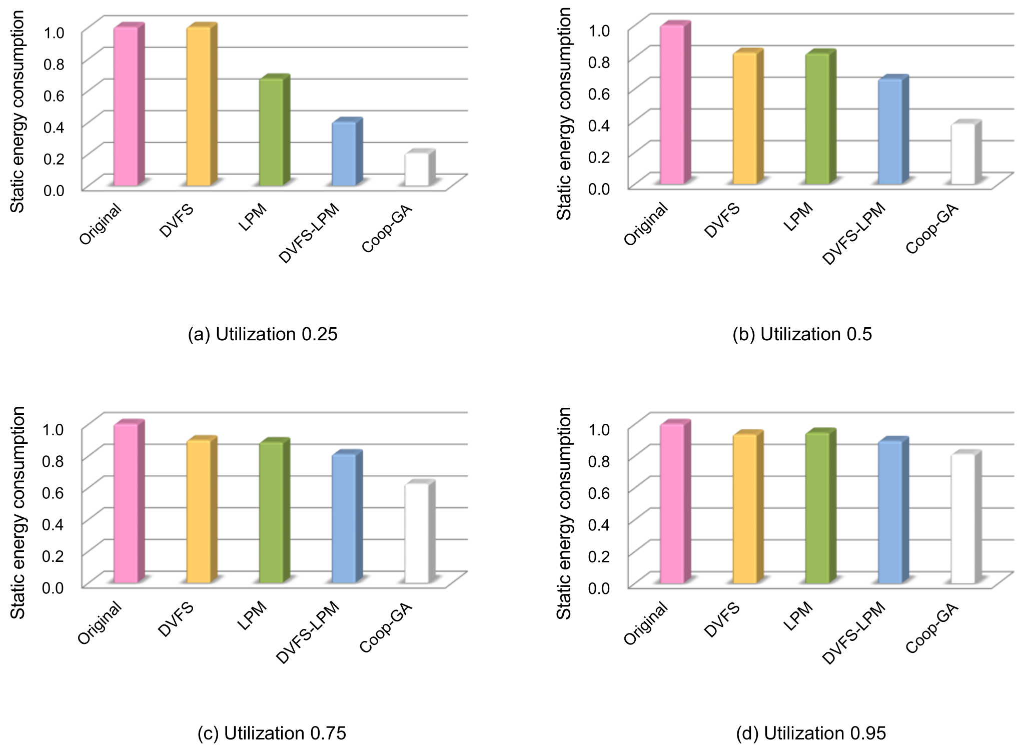


FIGURE 11. Static energy consumptions as the utilization is varied (synthetic workload).

of ORIGINAL, DVFS, LPM, and DVFS-LPM by 85.5%, 71.8%, 79.9%, and 49.1%, respectively, under the RSM workload and 76.3%, 64.7%, 71.7%, and 58.5%, respectively, under the IoT workload.

Figures 13 and 14 separately show the energy consumptions in processor and memory under the RSM and IoT workloads. As we see in Figure 13, algorithms adopting DVFS significantly reduce the processor's energy consumption.

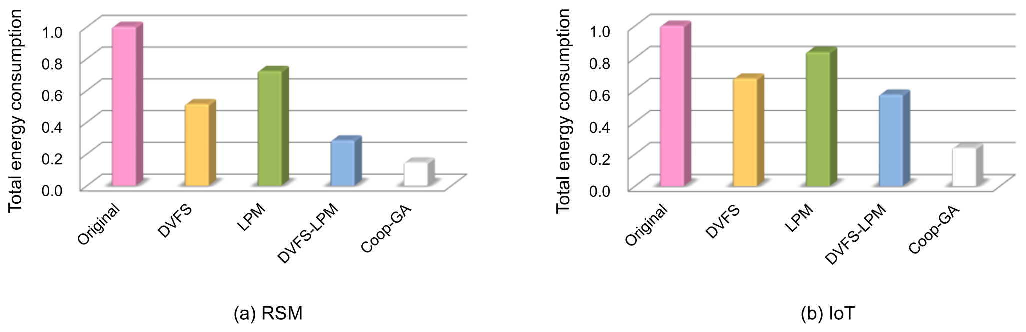


FIGURE 12. Total energy consumptions (realistic workload).

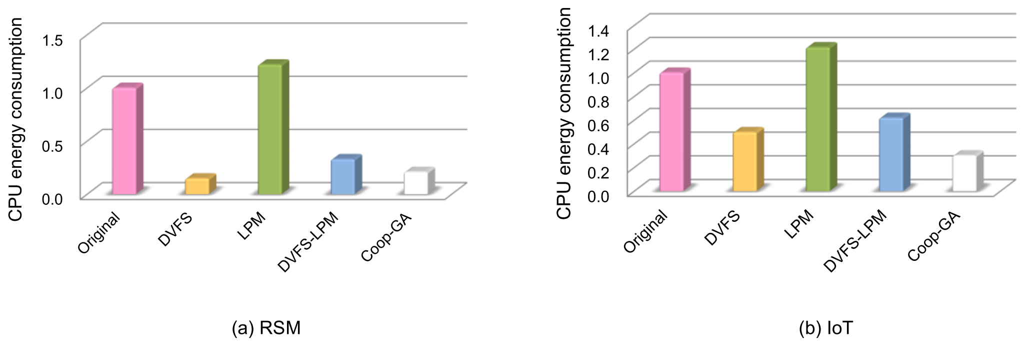


FIGURE 13. Energy consumptions in processor (realistic workload).

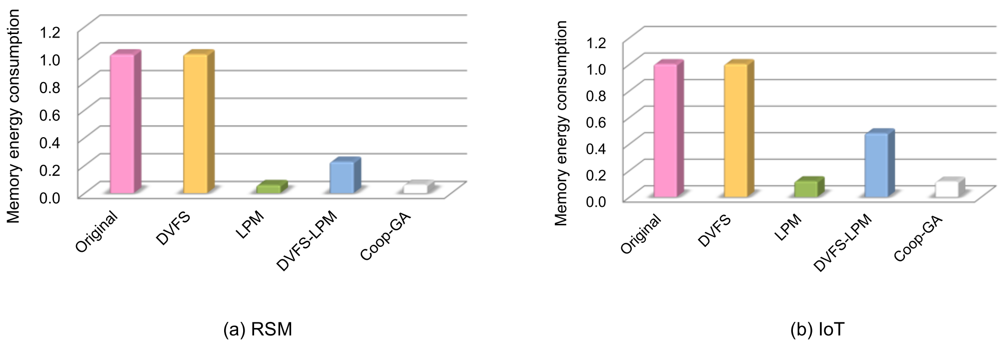


FIGURE 14. Energy consumptions in memory (realistic workload).

In particular, DVFS, DVFS-LPM, and COOP-GA reduce the energy consumption in processor by 85.2%, 67.1%, and 79.1% under RSM, and 50.2%, 38.1%, and 69.9% under IoT, compared to ORIGINAL, respectively, as shown in Figure 13.

In case of the memory energy consumption, algorithms adopting LPM significantly reduce the energy consumption as shown in Figure 14. In particular, LPM, DVFS-LPM, and COOP-GA reduce the memory energy consumption by 94.3%, 77.4%, and 94.0% under RSM, and 88.2%, 52.1%, and 88.5% under IoT, respectively, in comparison with ORIGINAL. This is because LPM's static energy consumption is very small.

V. RELATED WORKS

A. LOW-POWER MEMORY TECHNOLOGIES

Low-power memory (LPM) technologies have recently been considered as an additional memory medium to save the excessive power consumption of DRAM. As LPM is byte-addressable medium like DRAM but its power consumption is significantly less than DRAM due to no refresh operations, it is expected to be used as the main memory medium in emerging computing systems. Mogul *et al.* present a novel scheme that manages DRAM and LPM together as a unified memory system [5]. They aim at allocating read-intensive data to LPM, whereas write-intensive data to DRAM. This is because LPM is slower than DRAM specially for write operations. Dhiman *et al.* also use memory subsystems composed of DRAM and LPM, and allow moving data between DRAM and LPM for balancing the endurance of LPM [6]. Unlike our scheme, the target architectuire of these studies is not real-time systems, but general-purpose time-sharing systems. Thus, they consider the limited endurance of LPM rather than deadline constraints of real-time jobs in locating data on memory.

Qureshi *et al.* present a multi-level memory subsystem composed of LPM and DRAM [7]. In particular, they utilize DRAM as a buffer to LPM writes for extending the lifetime of LPM and alleviate the write latency of LPM. Zhou *et al.* present a multi-level memory subsystem composed of LPM and DRAM [8]. Specifically, they present a novel page eviction scheme that improves the cache miss ratio as well as the number of write operations from DRAM. These studies compose DRAM and LPM as a hierarchical architecture to improve memory system performances, not focusing on deadline guaranteed services in real-time systems.

Lee *et al.* present a new memory management scheme for the memory subsystem composed of LPM and DRAM [18]. Their scheme places read-intensive data on LPM whereas write-intensive data on DRAM through the analysis study of memory reference traces. Narayan *et al.* present a data allocation scheme for memory subsystem composed of DRAM and LPM in terms of the object level [19]. In particular, their scheme tries to reduce power consumptions as well as to improve performances, by allocating memory objects to the best-fit memory module through their characterization studies. Kannan *et al.* present a management policy for memory subsystems in virtualized environments [20]. They present a guest operating system that aims at avoiding migrations by locating data to appropriate memory media. They also propose data migration schemes and memory sharing schemes for virtual machines to improve performances. Lin *et al.* use greedy approximation and dynamic programming methods to solve the memory mapping problem between heterogeneous memory media [21]. The targets of these studies are also general-purpose systems, and thus they focus mainly on the improvement of overall system performances rather than the deadline-guranteed services in deciding memory allocation.

Zhang *et al.* present a job placement policy for heterogeneous memory types for power-saving purposes [22]. In particular, their policy places jobs one by one on LPM and checks the schedulability of the jobs, which is repeated until the placement of all jobs is completed. This study focuses on real-time embedded systems like our approach, but they only consider the memory allocation problem, without considering the dynamic voltage/frequency scaling of processors. Unlike previous studies, our model tightly evaluates the scaled worst-case execution time of a job, considering the overlapped latency between processor and memory, thereby minimizing the power consumption of real-time systems further.

B. DYNAMIC VOLTAGE/FREQNECY SCALING

Dynamic voltage/frequency scaling (DVFS) techniques have been widely studied for power saving in real-time processors [23]–[27]. Pillai and Shin present a technique to determine the lowest voltage/frequency to satisfy the deadline constraints of given jobs [28]. Their technique consists of static DVFS, cycle-conserving DVFS, and look-ahead DVFS. Static DVFS chooses the voltage/frequency of a processor statically, while cycle-conserving DVFS utilizes the reclaimed cycles for decreasing the voltage/frequency if the execution time of a job becomes shorter than the worst case execution time. Look-ahead DVFS further decreases the voltage/frequency value by making use of future computation requirement analysis and delaying the scheduling of the job based on the analysis results.

Lee *et al.* utilize the slack time to decrease the voltage/frequency of a processor [1]. In particular, voltage/frequency values are adjusted when unused clock cycles are reclaimed by finishing a job before its deadline. Ghor and Aggoune determine the least voltage/frequency schedules of real-time jobs by making use of slacks [24]. Specifically, their policy stretches the execution time of jobs based on off-line scheduling and determines the schedule of jobs as late as possible without violating deadline constraints. Nam *et al.* present a tight evaluation of real-time job schedulability considering both dynamic voltage/frequency scaling and hybrid memory allocation [29].

Unlike the aforementtioned studies that separately consider the execution in processor and states in memory, we define an extended job model to consider the memory allocation of jobs as well as the processor configuration. Based on this, the worst-case execution time of a job is re-evaluated by reflecting the read/write characteristics of the memory medium the job resides. Also, our model tightly evaluates the scaled worst-case execution time of a job, considering the overlapped latency between processor and memory, reducing the power consumption of real-time systems further.

VI. CONCLUSION

In this paper, we presented a novel real-time job scheduling approach that aims at minimizing the power consumption of processor and memory, without violating the deadline constraint of real-time jobs. Our policy formulates the power saving techniques of processor voltage/frequency scaling and memory job placement as a unified measure and co-optimizes the processor and memory power consumption. As our problem is a complex search problem, we made use of an efficient heuristic based on genetic algorithms to cut down the huge searching space and find a reasonable solution within a feasible time budget. To evaluate the proposed policy, we conducted experiments under various workload conditions. Our experimental results showed that the proposed policy reduces the energy consumption of real-time systems by 41.7% on average and up to 73.8%, without deadline misses.

As a future work, we consider measurement studies in real systems for validating the effectiveness of the proposed policy. We also plan to extend our policies by considering (m, k)-firm deadlines [30], in which deadline constraints of real-time jobs are relaxed such that a certain ratio of deadline misses can be allowed.

Real-time scheduling assumes the worst case execution time of a job, but the real execution may be finished much earlier than the worst case, leading to the significant waste of given resources. Some reactive strategies have been studied in order to address this issue. Chen *et al.* present an uncertainty-aware scheduling algorithm that determines the baseline schedule of real-time jobs beforehand, but during the execution, the reactive strategy dynamically generates new proactive baseline schedules by considering the completion of jobs or arrival of new jobs, allowing for more efficient resource management [42], [43]. This is effective in cloud environments as resources can be scaled up or down in accordance with the workload situation changes. However, our approach basically assumes the traditional hard real-time scheduling problem, where the real-time jobs are given beforehand and the resources are fixed, and thus, the scheduling is determined offline. However, we can use the basic idea of the reactive approach together with our algorithm as adjusting the voltage/frequency mode of a processor has the similar effect of scaling up/down of cloud resources. For example, during the execution of jobs based on the off-line schedule determined by our GA, if the actual execution time of a job becomes shorter than its worst case execution time, the voltage/frequency mode of a processor can be lowered for the remaining time slot allocated to that job. Through some preliminary experiments, we see that such approaches can reduce the processor's energy consumption by 3-10% depending on the reduction of the job's execution time. Unlike cloud environments, however, as the resources are fixed in our system, real-time jobs should be given beforehand and abruptly arriving jobs are not allowed in order to meet the deadline requirement. Thus, we plan to extend our power-saving technique in processor and memory by making use of the complete idea of the reactive approach in cloud environments.

REFERENCES

[1] Y. Lee, Y. Doh, and C. Krishna, "EDF scheduling using two-mode voltage clock scaling for hard real-time systems," in *Proc. CASES*, 2001, pp. 221–228.

[2] S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. ACM ASPLOS*, 2011, pp. 213–224.

[3] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. USENIX Annu. Tech. Conf.*, 2010, p. 21.

[4] S. Eilert, M. Leinwander, and G. Crisenza, "Phase change memory: A new memory technology to enable new memory usage models," in *Proc. 1st IEEE Int. Memory Workshop (IMW)*, 2009, pp. 1–2.

[5] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Proc. HotOS*, 2009, pp. 4–14.

[6] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. DAC*, 2009, pp. 559–664.

[7] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 24–33.

[8] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 14–23.

[9] H. Adeli and K. Sarma, *Cost Optimization of Structures: Fuzzy Logic, Genetic Algorithms, and Parallel Computing*. West Sussex, U.K.: Wiley, 2006.

[10] M.-X. Jiang, X.-X. Luo, T. Hai, H.-Y. Wang, S. Yang, and A. N. Abdalla, "Visual object tracking in RGB-D data via genetic feature learning," *Complexity*, vol. 2019, pp. 1–8, May 2019.

[11] W. Wang, Y. Dong, S. Zhong, and F. Liu, "Finite-time robust stability of uncertain genetic regulatory networks with time-varying delays and reaction-diffusion terms," *Complexity*, vol. 2019, pp. 1–18, Mar. 2019.

[12] D.-Y. Lin and Y.-H. Ku, "Using genetic algorithms to optimize stopping patterns for passenger rail transportation," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 29, no. 4, pp. 264–278, Apr. 2014.

[13] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.

[14] K. Park, B. K. Oh, H. S. Park, and S. W. Choi, "GA-based multi-objective optimization for retrofit design on a multi-core PC cluster," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 30, no. 12, pp. 965–980, Dec. 2015.

[15] Q. Gu, X. Li, and S. Jiang, "Hybrid genetic grey wolf algorithm for large-scale global optimization," *Complexity*, vol. 2019, Feb. 2019, Art. no. 2653512.

[16] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.

[17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*, 3rd ed. Heidelberg, Germany: Springer-Verlag, 1996.

[18] S. Lee, H. Bahn, and S. H. Noh, "CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2187–2200, Sep. 2014.

[19] A. Narayan, T. Zhang, S. Aga, S. Narayanasamy, and A. Coskun, "MOCA: Memory object classification and allocation in heterogeneous memory systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2018, pp. 326–335.

[20] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "HeteroOS: OS design for heterogeneous memory management in datacenter," in *Proc. ACM/IEEE 44th Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 521–534.

[21] Y. Lin, N. Guan, and Q. Deng, "Allocation and scheduling of real-time tasks with volatile/non-volatile hybrid memory systems," in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp. (NVMSA)*, Aug. 2015, pp. 1–6.

[22] Z. Zhang, P. Liu, L. Ju, and Z. Jia, "Energy efficient real-time task scheduling for embedded systems with hybrid main memory," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2014, pp. 1–10.

[23] K. Choi, W. Lee, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2005, pp. 29–34.

[24] H. E. Ghor and E. M. Aggoune, "Energy saving EDF scheduling for wireless sensors on variable voltage processors," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 2, pp. 158–167, 2014.

[25] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proc. 8th ACM Int. Conf. Autonomic Comput. (ICAC)*, 2011, pp. 31–40.

[26] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Trans. Softw. Eng.*, vol. 15, no. 10, pp. 1261–1269, Oct. 1989.

[27] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[28] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Oper. Syst. Princ. (SOSP)*, 2001, pp. 89–102.

[29] S. Nam, K. Cho, and H. Bahn, "Tight evaluation of real-time task schedulability for processor's DVS and nonvolatile memory allocation," *Micromachines*, vol. 10, no. 6, p. 371, Jun. 2019.

[30] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1443–1451, Dec. 1995.

[31] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Syst.*, vol. 2, no. 4, pp. 301–324, 1990.

[32] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *Proc. 21st Euromicro Conf. Real-Time Syst.*, Jul. 2009, pp. 249–258.

[33] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proc. 24th IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2003, pp. 52–62.

[34] Z. Wang, Y. Liu, Y. Sun, Y. Li, D. Zhang, and H. Yang, "An energy-efficient heterogeneous dual-core processor for Internet of Things," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 2301–2304.

[35] D. Zhu, D. Mosse, and R. Melhem, "Power-aware scheduling for AND/OR graphs in real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 849–864, Sep. 2004.

[36] J. Zhou, T. Wei, M. Chen, J. Yan, X. S. Hu, and Y. Ma, "Thermal-aware task scheduling for energy minimization in heterogeneous real-time MPSoC systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 8, pp. 1269–1282, Aug. 2016.

[37] Transmeta Crusoe. *Operating Modes for New Generation Processors*. Accessed: Jun. 20, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Transmeta_Crusoe

[38] C. Zhang, T. Meng, and G. Sun, "PM3: Power modeling and power management for processing-in-memory," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 558–570.

[39] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid DRAM-NVM memory architecture," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 936–941.

[40] H. A. Khouzani, F. S. Hosseini, and C. Yang, "Segment and conflict aware page allocation and migration in DRAM-PCM hybrid main memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1458–1470, Sep. 2017.

[41] J. Zhan, Y. Zhang, W. Jiang, J. Yang, L. Li, and Y. Li, "Energy-aware page replacement and consistency guarantee for hybrid NVM–DRAM memory systems," *J. Syst. Archit.*, vol. 89, pp. 60–72, Sep. 2018.

[42] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *J. Syst. Softw.*, vol. 99, pp. 20–35, Jan. 2015.

[43] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Trans. Services Comput.*, early access, Aug. 21, 2018, doi: 10.1109/TSC.2018.2866421.

HYOKYUNG BAHN (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, in 1997, 1999, and 2002, respectively.

He is currently a Full Professor of computer science and engineering with Ewha University, Seoul, South Korea. His research interests include operating systems, caching algorithms, storage systems, embedded systems, system optimizations, and real-time systems. He has published more than 70 articles in leading conferences and journals in these fields, including USENIX FAST, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and the *ACM Transactions on Storage*. He also received the Best Paper Awards at the USENIX Conference on File and Storage Technologies, in 2013.

KYUNGWOON CHO received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Seoul National University, in 1995, 1997, and 2012, respectively. He was a Chief Officer at the Clunix Research and Development Center, Seoul, South Korea. He is currently a Senior Researcher with the Embedded Software Research Center, Ewha University, Seoul. His research interests include multimedia systems, cloud computing, real-time systems, embedded systems, and operating systems.