Microprocessors and Microsystems 66 (2019) 19–30

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Adaptive genetic algorithm for energy-efficient task scheduling on asymmetric multiprocessor system-on-chip

Yonghee Yun[a], Eun Ju Hwang[b], Young Hwan Kim[a,*]

[a] Department of Electrical Engineering, Pohang University of Science and Technology (POSTECH), Pohang 790-784, Republic of Korea
[b] Foundry Business, Samsung Electronics Company Ltd., Hwaseong 18448, Republic of Korea

ARTICLE INFO

Article history:
Received 26 September 2017
Revised 2 September 2018
Accepted 28 January 2019
Available online 30 January 2019

Keywords:
Design space exploration
Static task scheduling
Energy optimization
Design-time task scheduling
Asymmetric MPSoC
Genetic algorithm

ABSTRACT

This paper proposes a genetic algorithm (GA) based energy-efficient design-time task scheduling algorithm, AGATS, for an asymmetric multiprocessor system-on-chip. Unlike existing GA-based task scheduling algorithms, AGATS adaptively applies different generation strategies to solution candidates based on their completion time and energy consumption. For solution candidates to evolve intelligently, instead of using conventional genetic operators, AGATS uses three generation strategies: elitism, mutation of elites (MOE), and adaptive generation (AG). The first copies a small portion of elite solution candidates into the next generation to guarantee that solution quality does not decrease from the current to the next generation. The second mutates randomly selected elite solution candidates to maintain both the diversity of candidates and solution quality. Finally, the third adaptively evolves solution candidates toward better candidates based on their completion time and energy consumption. In experiments, AGATS reduced energy consumption by up to 29.3% compared to existing methods and outperformed them in most cases. Furthermore, it identified feasible solutions effectively, which was not the case with the existing methods under tight timing constraints.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

Traditional embedded systems employ a simple structure to perform specific operations. However, in recent years, embedded systems have become increasingly versatile. As such, the structure of these embedded systems has become increasingly complicated [1]. In addition, system requirements for high performance and low power consumption have become more important as the use of mobile embedded systems has become widespread in daily life. Thus, multiprocessor system-on-chip (MPSoC) [2], which contains multiple cores in a single die has been widely used in modern embedded systems to satisfy both performance and power requirements.

MPSoC platforms can be categorized into heterogeneous and homogeneous MPSoC. In a heterogeneous MPSoC platform, processing elements have a different instruction set architecture (ISA). Thus, this platform generally contains CPUs, digital signal processors (DSPs), and GPUs in the same platform. By contrast, a homogeneous MPSoC platform has processing elements that have the same ISA. If the processing elements in the homogeneous platform are completely identical in terms of both performance and power, this platform is referred to as a symmetric MPSoC platform. Otherwise (that is, if the processing elements have different performance and power characteristics but the same ISA), this platform is referred to as an asymmetric MPSoC platform. In recent years, asymmetric MPSoC platforms such as the open multimedia applications platform (OMAP) application processor [3] from Texas Instruments (TI) and the big.LITTLE processor from ARM [4] are widely used in modern embedded systems to reduce energy consumption while satisfying timing requirements.

In MPSoC systems, performance and energy consumption largely depend on the utilization of resources. Therefore, task scheduling which assigns application tasks to a given platform, has become a major design step in modern embedded systems. The task scheduling problem, which is known as NP-complete [5], cannot practically be solved by exhaustive search approaches because such approaches require a computational complexity considered too high for use in the design process. Therefore, most task scheduling algorithms have focused on finding sub-optimal solutions based on heuristic approaches such as simulated annealing (SA), ant colony optimization (ACO), genetic algorithm (GA), and particle swarm optimization (PSO). Among heuristic algorithms, GA has been most widely used to solve task scheduling problems and is known to identify better scheduling solutions than do other heuristic algorithms [6]. However, most existing GA-based

* Corresponding author.
E-mail address: youngk@postech.ac.kr (Y.H. Kim).

https://doi.org/10.1016/j.micpro.2019.01.011
0141-9331/© 2019 Elsevier B.V. All rights reserved.

task scheduling algorithms apply the same crossover and mutation operators to solution candidates in order to explore design space [6–8]. This implies that existing algorithms cannot consider timing and energy consumption information of solution candidates during the generation process. Thus, existing algorithms cannot effectively evolve solution candidates. This results in less optimized scheduling results and no feasible solutions under tight timing constraints.

The remainder of this paper is organized as follows. Section 2 briefly reviews related work and the concept of GA. Section 3 describes the system model, including both the application and multiprocessor models, and defines the MPSoC task scheduling problem. Section 4 describes the proposed task scheduling algorithm. Section 5 presents experimental results. Section 6 concludes the paper.

2. Related work

Task scheduling can be classified into run-time and design-time scheduling. Run-time scheduling, also known as dynamic or online scheduling, allocates tasks during run-time. This scheduling strategy can be used when an application dynamically changes during run-time [11–13]. By contrast, design-time scheduling, also known as static or offline scheduling, allocates tasks at design-time. This scheduling strategy can be used when application tasks are completely determined before the program executes [7–10, 14–25]. When applications are completely predictable before execution, the design-time task scheduling algorithms generally identify solutions that are closer to optimal than those of run-time scheduling algorithms. This is because design-time scheduling algorithms can perform computationally intensive searches to find solutions without being constrained by run-time overhead. Therefore, extensive research on design-time task scheduling algorithms has been conducted.

Early research on design-time task scheduling algorithms has primarily focused on minimizing the make-span which is the completion time of the application. For example, task scheduling algorithms based on SA [14], tabu search [15], and PSO [16] focus on finding scheduling solutions that are optimized only for system performance. In addition, some reliability-aware task scheduling approaches were proposed in [17–18]. The study in [17] explored concurrent replication with canceling to reduce the response times of parallel jobs subject to failure and improve the reliability. In [18], an SA-based reliability-aware task scheduling method for lifetime extensions of platform-based MPSoC designs under performance constraints was proposed. However, as battery-powered mobile devices are widely used in daily life, embedded systems should consider not only operating speed but also the energy consumption of systems. Accordingly, task scheduling algorithms that aim to minimize energy consumption of a symmetric MPSoC while also satisfying performance requirements have influenced modern embedded system design. In [10], an ACO-based task scheduling algorithm was proposed to reduce the energy consumption of the homogeneous symmetric MPSoC systems. This approach considers the time slack of each solution candidate to reduce the processor communication volume and to determine a dynamic voltage and frequency scaling (DVFS) level of each task. The study in [19] proposed an SA-based task scheduling algorithm for reducing the energy of homogeneous symmetric MPSoC systems while also satisfying timing constraints. In [20], a network contention-aware energy management scheme for mapping tasks to NoC- and Voltage-Frequency Island (VFI)-based multicore systems with DVFS capability was proposed. In [7] a GA-based task scheduling algorithm, called GeneS, was proposed to minimize the energy consumption of homogeneous symmetric MPSoCs that support per-core DVFS while also satisfying timing constraints. Thus, GeneS provides not only the task-to-core scheduling but also the DVFS level of each task.

As previously stated, extensive research has been conducted on design-time task scheduling algorithms that target a homogeneous symmetric MPSoC in order to minimize energy consumption while satisfying timing constraints. However, in modern embedded system design, asymmetric or heterogeneous MPSoC platforms are widely used for performance and energy consumption optimization. Therefore, a design-time task scheduling algorithm that targets an asymmetric or heterogeneous MPSoC is essential for practical energy-efficient embedded system design. The scheduling algorithms for asymmetric and heterogeneous MPSoCs can be used interchangeably. In [21], a stochastic dynamic level scheduling algorithm which optimizes the performance of a heterogeneous system was proposed. The study in [22] proposed a hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing platforms. In [23], a scheduling algorithm based on the concept of constrained critical paths was proposed to provide better schedules for heterogeneous systems. A list scheduling algorithm which is based on an optimistic cost table was proposed to optimize the make-span for heterogeneous systems [24]. However, these studies focus on the performance of the system without consideration on the energy consumption. In [25], heuristic scheduling algorithms were proposed to minimize the total cost of heterogeneous multicore embedded systems, under limited time and resource constraints. However, this study cannot consider systems that support DVFS. In [9], a GA-based task scheduling algorithm that targets an asymmetric MPSoC was proposed to reduce energy consumption while satisfying timing constraints. This algorithm classifies solution candidates based on the timing and energy information of the solution candidates, and then applies effective operations to them for evolvement. By means of this process, the algorithm effectively identifies energy-optimized solutions than other existing algorithms. However, this algorithm still applies the same evolution operations to those solution candidates that were classified in the same group and thus cannot consider the differences between those solution candidates. Therefore, these solution candidates may not effectively evolve toward better solutions. In addition, this algorithm also cannot consider systems that support DVFS.

This paper proposes a task scheduling algorithm, AGATS (Adaptive Genetic Algorithm for Task Scheduling) that targets an asymmetric MPSoC to determine the static schedule and DVFS levels for each task in order to minimize the energy consumption of a system while satisfying timing constraints. AGATS is based on a GA and adapts the elitism strategy which is a well-known strategy to improve the solution quality of the GA. In addition, AGATS uses two new generation strategies, mutation of elites (MOE) and adaptive generation (AG), for effective evolvement of solution candidates. Typical GA-based approaches use crossover and mutation to take advantages of exploration and exploit, when generating the populations. In AGATS, we use AG and MOE for the same purposes, which performed better in our experiments. During evolvement, AGATS retains elite solution candidates using the elitism strategy, and then mutates elite solution candidates using the MOE strategy in order to increase solution quality. Finally, AGATS uses the AG strategy to generate the next solution candidates adaptively. Therefore, AGATS finds higher quality solutions than existing algorithms. In addition, under tight timing constraints, AGATS finds feasible solutions more effectively than do the benchmark algorithms. Furthermore, AGATS can be used practically for embedded system design based on asymmetric MPSoCs that leverage the per-core DVFS.

3. System models and problem statement

This section describes the application and target platform models, and defines the energy-efficient task scheduling problem for an asymmetric MPSoC.

3.1. Application model

A parallel application can be represented as a task graph that contains information about tasks. As shown in Fig. 1, the task graph is modeled as a directed acyclic graph (DAG). DAG $G(V,E)$ is a node- and edge-weighted graph, where $V = \{v_1, v_2, \ldots, v_M\}$ is a set of M nodes, and $E=\{e_1, e_2, \ldots, e_m\}$ is a set of m edges. Each node represents a task. The direction of each edge denotes the dependency between a pair of tasks. For example, in Fig. 1, v_4 cannot be executed before v_1 is completed and all data transfers between the tasks are finished. Each task has different execution time and energy consumption when it runs on different cores or different DVFS levels. Therefore, the execution time, $T_i(j,k)$, and the energy consumption, $E_i(j,k)$ of the ith task running on the jth core at the kth DVFS level are provided as a library at design-time. In addition, the communication data volume between the two dependent tasks v_a and v_i $comm(v_a,v_i)$ is also provided. Fig. 2(a) provides an example of the library that has the execution time and energy consumption of each task running on different types of cores operating at different DVFS levels. There are two types of cores: high performance cores (HPcores) and low power cores (LPcores), operating at three DVFS levels. As shown in Fig. 2(a), the HPcores are faster but consume more energy than the LPcores because the HPcores require higher voltage to operate than do the LPcores. Fig. 2(b) shows the communication data volume between the tasks that have a precedence relation.

3.2. Target platform model

The target platform is an asymmetric MPSoC that consists of N processing cores $C=\{c_1, c_2, \ldots, c_N\}$ and supports per-core DVFS. The cores are classified into several types of cores having different operating speed and power consumption characteristics but have the same ISA. Each processing core supports dl discrete DVFS levels $F=\{f_1, f_2, \ldots f_{dl}\}$. The cores are connected using 2D mesh-type network-on-chip (NoC) in order to communicate with one another. According to [26], the contention overhead within the NoC is negligible, especially if the system contains 12 or a fewer number of cores. In addition, [27] points out that the buffering energy consumption which is a parameter tightly coupled with the network contention is negligible. Thus, in this work, we assume that the contention overhead within the NoC is negligible.

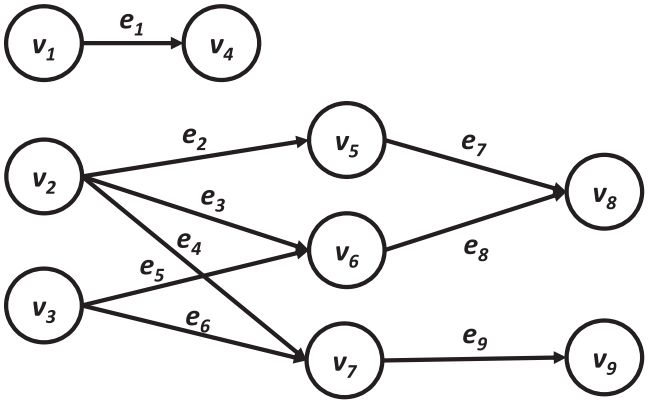


Fig. 1. Example of a task graph represented by a directed acyclic graph.

DVFS level: 1	HPcore		LPcore	
DVFS level : 2	HPcore		LPcore	
DVFS level: 3	HPcore		LPcore	
Task	**Execution time (ns)**	**Energy consumption (μJ)**	**Execution time (ns)**	**Energy consumption (μJ)**
1	78.4	5.3	203.9	1.8
2	142.5	9.3	399.0	3.6
…	…	…	…	…
M	8497.2	699.2	20393.3	184.0

(a)

	Data volume (bit)			
Task	**1**	**2**	**…**	***M***
1	0	2e6	…	3e3
2	2e6	0	…	4e3
…	…	…	…	…
M	3e3	4e3	…	0

(b)

Fig. 2. Example of task information: (a) execution time and energy consumption of each task at three DVFS levels, and (b) data volume between the tasks.

3.3. Energy model

The total energy consumption of the asymmetric MPSoC, E_{total}, is calculated as follows:

$$E_{total} = E_{dynamic} + E_{static} + E_{comm}, \tag{1}$$

where $E_{dynamic}$ is the dynamic energy consumption of the cores, E_{static} is the static energy consumption of the cores when they are in the idle state, and E_{comm} is the communication energy consumption of NoC between the dependent tasks. $E_{dynamic}$ can be calculated as follows:

$$E_{dynamic} = \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{dl} E_i(j,k) \times x_i(j,k), \tag{2}$$

where $x_i(j,k)=1$, when the ith task is mapped onto the jth core operating at the kth DVFS level. E_{static} can be obtained as follows:

$$E_{static} = \sum_{j=1}^{N} P_{static,j} \times T_{idle,j}, \tag{3}$$

where $P_{static,j}$ is the static power of the jth core and $T_{idle,j}$ is the idle time of the jth core. E_{comm} can be calculated using the NoC energy model proposed in [27] as follows:

$$E_{comm} = \sum_{a=1}^{M}\sum_{i=1}^{M} comm(v_a, v_i) \times diff(j,b) \times (MD(j,b)+1) \times E_{Router} + MD(j,b) \times E_{Link}), \tag{4}$$

where $MD(j,b)$ is the Manhattan distance between the jth core to which the ith task is mapped and the bth core to which the preceding task of the ith task is mapped. E_{Router} and E_{Link} are the router energy and link energy, respectively, when one bit of data is transferred. Note that $diff(j,b)=1$ if the ath task and ith task are assigned to the different cores ($j \neq b$). Thus, if the dependent tasks are assigned to the same core, the communication energy consumption is ignored.

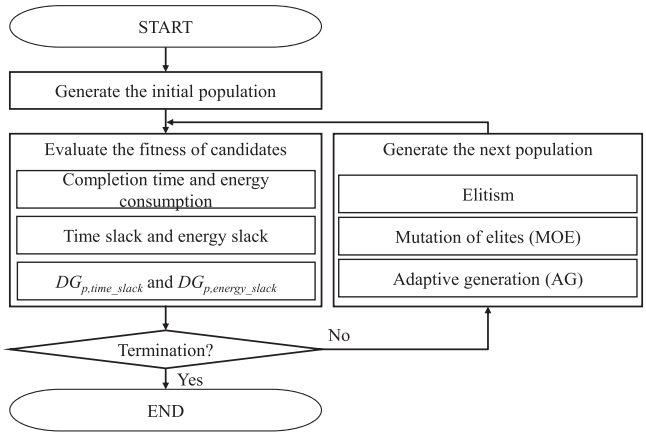


Fig. 3. Flow of AGATS.

Task	1	2	3	4	5	...	*M*-4	*M*-3	*M*-2	*M*-1	*M*
Core	2	1	3	1	2	...	1	2	2	1	3
DVFS	1	2	3	1	2	...	1	2	3	2	3

Fig. 4. Example of a chromosome representation for AGATS.

3.4. Problem statement

The task scheduling and voltage selection problem to minimize the energy consumption of the asymmetric MPSoC that supports per-core DVFS is defined as follows:

Given a DAG $G(V,E)$, an asymmetric MPSoC platform, which supports per-core DVFS, and a timing constraint TC, the goal of task scheduling problem is to identify the best static schedule and the best DVFS level for each task to minimize the total energy consumption while satisfying a timing constraint TC.

4. Proposed task scheduling algorithm

This section presents the proposed task scheduling algorithm, AGATS, which minimizes the energy consumption of the asymmetric MPSoC while satisfying the timing constraint. Fig. 3 presents the flow of AGATS. In contrast to the conventional task scheduling algorithms based on the GA, AGATS uses MOE and AG strategies to generate the next solution candidates effectively.

4.1. Chromosome representation

Fig. 4 presents a chromosome representation used in AGATS. Each chromosome represents a scheduling solution candidate that contains M genes, where M is the number of tasks. Each gene contains information about the task. The first string of genes represents the core numbers to which tasks are assigned and the second string of genes indicates the operating DVFS levels. When creating a schedule from a chromosome, the tasks assigned to the same core are sorted based on their precedence relations. Among tasks without having precedence relations, those having longer execution time take a higher priority for execution than do those having shorter execution time.

4.2. Generation of initial solution candidates

AGATS first constructs several initial solution candidates. The number of solution candidates in each generation, which is called population, is denoted as P. AGATS allocates tasks to the cores randomly for P solution candidates during the initial generation. This random allocation guarantees the diversity of solution candidates, resulting in broad design space exploration. Then, the DVFS level of each task is set to the highest value in order to determine feasible solutions effectively under a tight timing constraint.

4.3. Evaluation

AGATS evaluates each solution candidate in the population based on the completion time, total energy consumption, and the degree of speed-up with respect to the time slack and energy slack.

The completion time of the pth solution candidate, indicating the completion of all tasks in the task graph, is denoted as T_p. Therefore, T_p is obtained by adding the execution time of each task on the critical path in the pth solution candidate.

The total energy consumption E_{total} of the pth solution candidate is denoted as E_p. E_p is calculated using Eqs. (1)–(4).

Based on the evaluated completion time and energy consumption of each solution candidate, AGATS ranks the solution candidates. The fitness value of the pth solution candidate, which determines the ranking of the solution candidates is defined as follows:

$$fitness_p = \begin{cases} 1/E_p, & T_p \le TC \\ TC - T_p, & \text{otherwise} \end{cases}, \quad (5)$$

where, $fitness_p$ is the fitness value of the pth solution candidate. If the pth solution satisfies the timing constraint, the fitness value is the inverse of the energy consumption of the pth solution candidate. Thus, in this case, a solution candidate that consumes lower amount of energy takes a higher fitness value than a solution candidate that consumes higher amount of energy. However, if a solution candidate does not satisfy the timing constraint, the fitness value is always negative. This indicates that the fitness value of a solution candidate that does not satisfy the timing constraint is always lower than that of a solution candidate that satisfies the timing constraint. Furthermore, if a solution candidate does not satisfy the timing constraint, the candidate with a shorter completion time takes a higher fitness value than the candidate with a longer completion time. AGATS ranks the solution candidates in descending order of fitness values.

The number of speed-up tasks necessary to increase the speed of execution in the pth solution candidate is denoted as $NT_{p,speedup}$. $NT_{p,speedup}$ is calculated based on the degree of speed-up with respect to the time slack and energy slack of the pth solution candidate. The degree of speed-up with respect to the time slack and energy slack of the pth solution candidate is denoted as $DG_{p,time\_slack}$ and $DG_{p,energy\_slack}$, respectively. The time slack of each

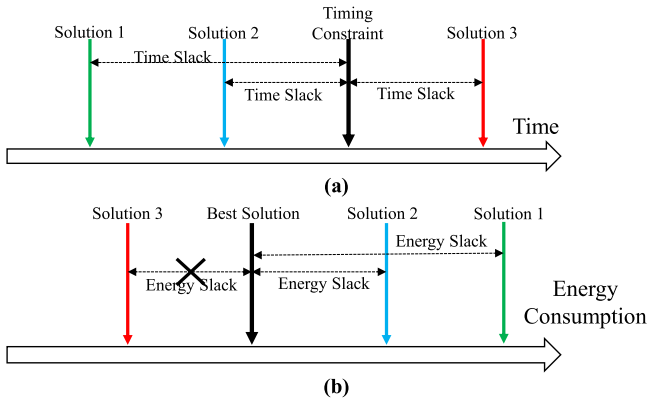


Fig. 5. (a) Time slack, and (b) energy slack of solution candidates.

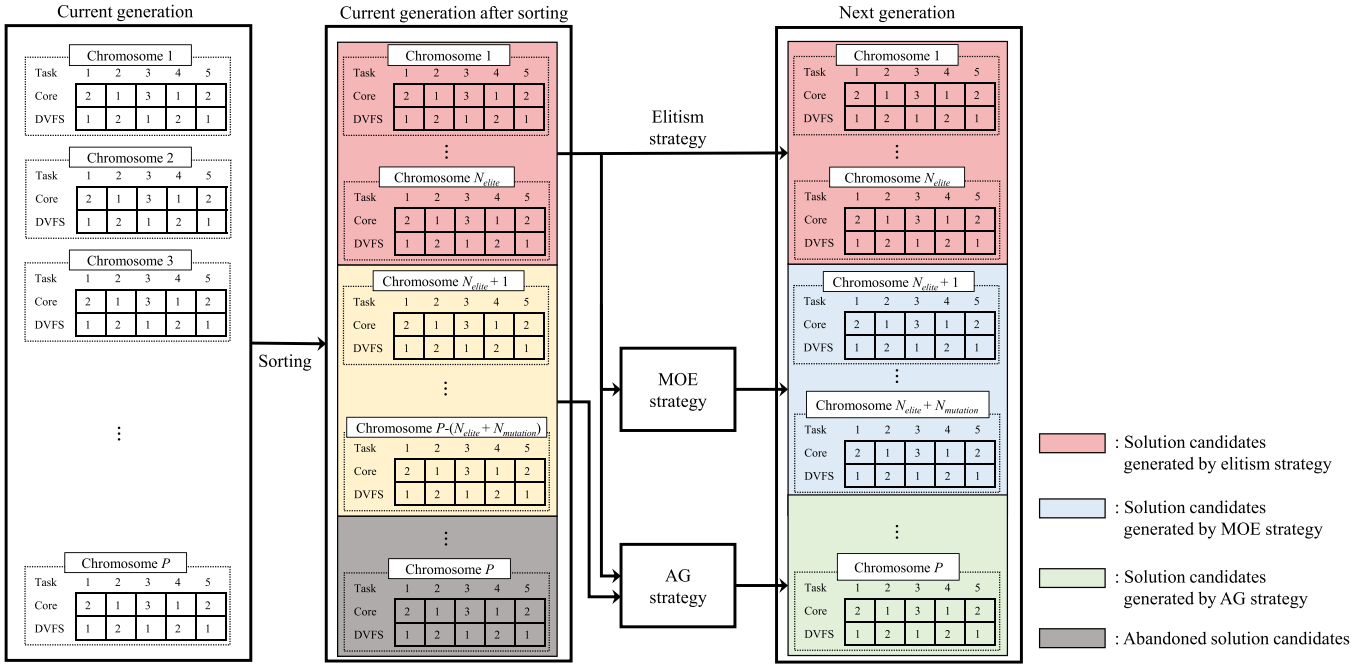


Fig. 6. Overall generation process.

solution candidate indicates the difference between the completion time of the solution and the timing constraint. Fig. 5(a) shows the concept of the time slack. In Fig. 5(a), the completion time of *Solution 3* is longer than the timing constraint. Thus, the execution speed of some tasks in *Solution 3* should be increased to satisfy the timing constraint. By contrast, *Solution 1* and *Solution 2* satisfy the timing constraint. Thus, the execution speed of some tasks in both *Solution 1* and *Solution 2* should be decreased to reduce the energy consumption. To consider these points, AGATS calculates $DG_{p,time\_slack}$ as follows:

$$DG_{p,time\_slack} = \begin{cases} \left\lceil M \times \frac{T_p - TC}{TC} \right\rceil, & \text{if } T_p > TC \\ \left\lfloor M \times \frac{T_p - TC}{TC} \right\rfloor, & \text{otherwise} \end{cases}. \quad (6)$$

In Eq. (6), if T_p is less than TC, $DG_{p,time\_slack}$ is negative. The negative $DG_{p,time\_slack}$ value indicates the degree of speed-down for the pth solution candidate. Thus, *Solution 1* and *Solution 2* in Fig. 5(a) have negative values of $DG_{p,time\_slack}$. As *Solution 1* has a greater time slack than does *Solution 2, Solution 1* has a higher degree of speed-down than *Solution 2*. In order to obtain an integer value of $DG_{p,time\_slack}$ and to change at least one task of the solution candidate, AGATS applies ceiling and floor operators to the cases in which T_p is greater than TC and T_p is smaller than TC, respectively. AGATS calculates $DG_{p,energy\_slack}$ in a similar manner. The energy slack of each solution candidate is the difference between the energy consumption of the best solution and the current solution candidate. In Fig. 5(b), *Solution 1* and *Solution 2* consume more energy than does the best solution. Thus, *Solution 1* and *Solution 2* should reduce their energy consumption by decreasing the speed of execution. To consider this point, AGATS calculates $DG_{p,energy\_slack}$ as follows:

$$DG_{p,energy\_slack} = \begin{cases} \left\lfloor M \times \frac{E_{best} - E_p}{E_{best}} \right\rfloor, & \text{if } E_p > E_{best} \\ 0, & \text{otherwise} \end{cases}, \quad (7)$$

where E_{best} is the energy consumption of the best solution in the current generation. In Eq. (7), $DG_{p,energy\_slack}$ is proportional to the energy slack of the pth solution candidate. In Fig. 5(b), $DG_{p,energy\_slack}$ does not consider *Solution 3* that the energy consumption of the pth solution candidate is less than or equal to that of the best solution in the current generation. This is because the solution candidate that consumes a lower amount of energy than the current best solution does not have to modify the execution speed of tasks while considering energy characteristics. In this case, AGATS sets $DG_{p,energy\_slack}$ to zero, and thus, $DG_{p,energy\_slack}$ can only use either negative values or zero. The negative value of $DG_{p,energy\_slack}$ refers to the degree of speed-down for the pth solution candidate. Therefore, in Fig. 5(b), *Solution 1* has a higher degree of speed-down compared to *Solution 2*. AGATS also uses the floor operator to guarantee that at least one task in the solution candidate is subjected to the evolvement. $DG_{p,time\_slack}$ and $DG_{p,energy\_slack}$ are used in the generation process to determine $NT_{p,speedup}$ by considering both time slack and energy slack of each solution candidate.

4.4. Generation

AGATS generates the next solution candidates using three generation strategies: elitism, MOE, and AG. Fig. 6 shows the overall generation process of AGATS. One of the key points of AGATS is to use the three generation strategies in a balanced way by considering the advantages and disadvantages of each strategy.

The first generation method is the elitism strategy, which directly copies a fixed number, N_{elites}, of elite solution candidates into the next generation without modification. These candidates are the top N_{elites} solution candidates determined by their ranks in the current generation. This strategy maintains good solution candidates over generations, and thus, it is known to guarantee that the solution quality does not decrease during the optimization process [28].

The MOE strategy is the second generation method of AGATS. It applies a mutation operator to the randomly selected elite solution candidates to generate a certain number, $N_{mutation}$, of mutation solution candidates. MOE can help to escape from the local optimum by diversifying the population. However, it does not guarantee to move to better solutions. First, AGATS randomly selects a solution candidate that is a type of elite solution candidate. Subsequently, AGATS randomly selects a task of the selected elite solution candidate, and applies the mutation

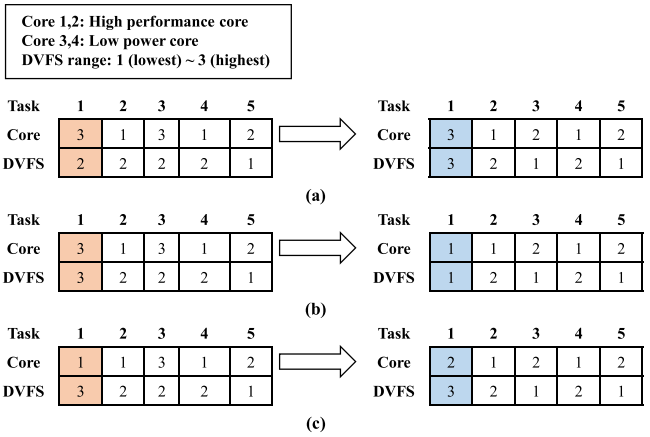


Fig. 7. Example of the MOE strategy.

operation, which randomly increases or decreases the DVFS level of the selected task. If the DVFS level of the selected task is the highest DVFS level when the MOE decides to increase the DVFS level, AGATS migrates the selected task to a randomly selected faster core at the lowest DVFS level. However, if the selected task is assigned to the fastest core in the system at the highest DVFS level, AGATS migrates the selected task to another randomly selected fastest core in the system at the highest DVFS level. Similarly, if the DVFS level of the selected task is the lowest DVFS level when the MOE decides to decrease the DVFS level, AGATS migrates the selected task to a randomly selected slower core at the highest DVFS level. If the selected task is assigned to the slowest core in the system at the lowest DVFS level, AGATS migrates the selected task to another randomly selected slowest core in the system at the lowest DVFS level. For example, as shown in Fig. 7, AGATS randomly selects *Task 1*, and subsequently increases the DVFS level of the selected task. In Fig. 7(a), as the DVFS level of *Task 1* is not the highest, AGATS increases the DVFS level of *Task 1*. Fig. 7(b) shows another case wherein the DVFS level of *Task 1* is the highest level supported by the system. Thus, AGATS migrates *Task 1* to the higher performance core in the system at the lowest DVFS level. Fig. 7(c) shows the case wherein *Task 1* is assigned to the fastest core in the system at the highest DVFS level. Here, AGATS migrates *Task 1* to the other fastest core in the system at the highest DVFS level. In this manner, the MOE strategy slightly alters the timing and energy characteristics of the randomly selected elite solution candidates. This process is iteratively performed until $N_{mutation}$ solution candidates are generated. The MOE strategy is expected to search the design space near the elite solution candidates to determine higher quality solutions.

The final generation method is the AG strategy. This strategy uses the time slack and energy slack of each solution candidate to generate the remaining part, N_{AG} (P - N_{elites} - $N_{mutation}$), of the next solution candidates. This strategy can enable the solution candidates to evolve into better solutions. However, it may result in local optimum solutions. AGATS adaptively applies different operations to the solution candidates based on their $DG_{p,time\_slack}$ and $DG_{p,energy\_slack}$ values.

If $DG_{p,energy\_slack}$ is zero, AGATS generates the next solution candidate considering only the $DG_{p,time\_slack}$ value. In this case, if the $DG_{p,time\_slack}$ value is positive, AGATS sets $NT_{p,speedup}$ to one random value from 1 to $DG_{p,time\_slack}$. Then, AGATS randomly selects $NT_{p,speedup}$ tasks in the pth solution candidate to increase the speed of execution. First, AGATS attempts to increase the DVFS levels of the selected tasks. If the DVFS levels of the selected tasks are already the highest, AGATS migrates the selected tasks to randomly selected faster cores at the lowest DVFS level in order to accelerate the operation. If the selected tasks are already assigned to the fastest core in the system at the highest DVFS level, AGATS migrates the selected tasks to another randomly selected fastest core in the system at the highest DVFS level. This speed-up process can effectively reduce the completion time of the solution candidates that do not satisfy the timing constraint. By contrast, if the $DG_{p,time\_slack}$ value is negative, the operating speed of the pth solution candidate should be decreased. In other words, the energy consumption of the pth solution candidate should be reduced to improve the solution quality. Thus, in this case, AGATS sets $NT_{p,speedup}$ to one random value from -1 to $DG_{p,time\_slack}$. Considering the meaning of $NT_{p,speedup}$, the negative value of $NT_{p,speedup}$ indicates the number of speed-down tasks necessary to decrease the speed of execution in the pth solution candidate. Thus, AGATS selects $|NT_{p,speedup}|$ tasks to reduce the operating speed. First, AGATS attempts to reduce the DVFS levels of the selected tasks. If the DVFS levels of the selected tasks are already the lowest, AGATS migrates the selected tasks to randomly selected slower cores in order to reduce the energy consumption. If the selected tasks are assigned to the slowest core in the system at the lowest DVFS level, AGATS migrates the tasks to another randomly selected slowest core in the system at the lowest DVFS level. Thus, AGATS can effectively reduce the energy consumption of the solution candidates that have a timing margin.

However, if $DG_{p,energy\_slack}$ is not zero but negative, AGATS generates the next solution candidates considering both $DG_{p,time\_slack}$ and $DG_{p,energy\_slack}$. In this case, if $DG_{p,time\_slack}$ is positive, the pth solution candidate does not satisfy the timing constraint and consumes more energy than the current best solution. This indicates that the pth solution candidate has little room for improvement in terms of timing and energy. Thus, AGATS replaces the pth solution candidate with a randomly regenerated solution candidate. This enables a wider design space exploration by replacing solutions that have little room for improvement with newly generated solutions. By contrast, if $DG_{p,time\_slack}$ is negative, the timing margin can be used to increase the energy efficiency by reducing the speed of operation. AGATS first calculates $DG_{p,avg}$. $DG_{p,avg}$ is calculated by averaging $DG_{p,time\_slack}$ and $DG_{p,energy\_slack}$ as follows:

$$DG_{p,avg} = \left(DG_{p,time\_slack} + DG_{p,energy\_slack}\right)/2 \tag{8}$$

As $DG_{p,energy\_slack}$ and $DG_{p,time\_slack}$ are both negative, $DG_{p,avg}$ is also negative. Thus, in this case, AGATS sets $NT_{p,speedup}$ to one random value from -1 to $DG_{p,avg}$, and then selects $|NT_{p,speedup}|$ tasks to reduce the operating speed. The speed reduction process is the same as that using only $DG_{p,time\_slack}$. Fig. 8 shows the examples of the AG strategy. In Fig. 8, the first solution candidate has a positive $DG_{1,time\_slack}$ value of 3 and a zero $DG_{1,energy\_slack}$ value. Thus, AGATS sets $NT_{1,speedup}$ to one random value from 1 to 3. In this example, the value of $NT_{1,speedup}$ is set to 3, and thus, AGATS randomly selects three tasks (*Task 1, Task 3*, and *Task 4*) to increase their DVFS levels or to change their core types to faster cores. *Task 1* originally ran on a faster core operating at the 1st DVFS level. Thus, the AG strategy increases the DVFS level of *Task 1* to the 2nd DVFS level. *Task 3* originally ran on a slower core operating at the 3rd DVFS level, which is the maximum DVFS level in this example. Thus, the AG strategy changes the core allocation of *Task 3* to a faster core with the lowest DVFS level. *Task 4* originally ran on the fastest core in the system, operating at the 2nd DVFS level. Thus, the AG strategy increases the DVFS level of *Task 4* to the maximum DVFS level. By contrast, the second solution candidate in Fig. 8 has a negative $DG_{2,time\_slack}$ value of -2 and zero $DG_{2,energy\_slack}$ value. Thus, AGATS sets $NT_{2,speedup}$ to one random value from -1 to -2. In this example, the value of $NT_{2,speedup}$ is set to -2, and thus AGATS randomly selects two tasks (*Task 2*

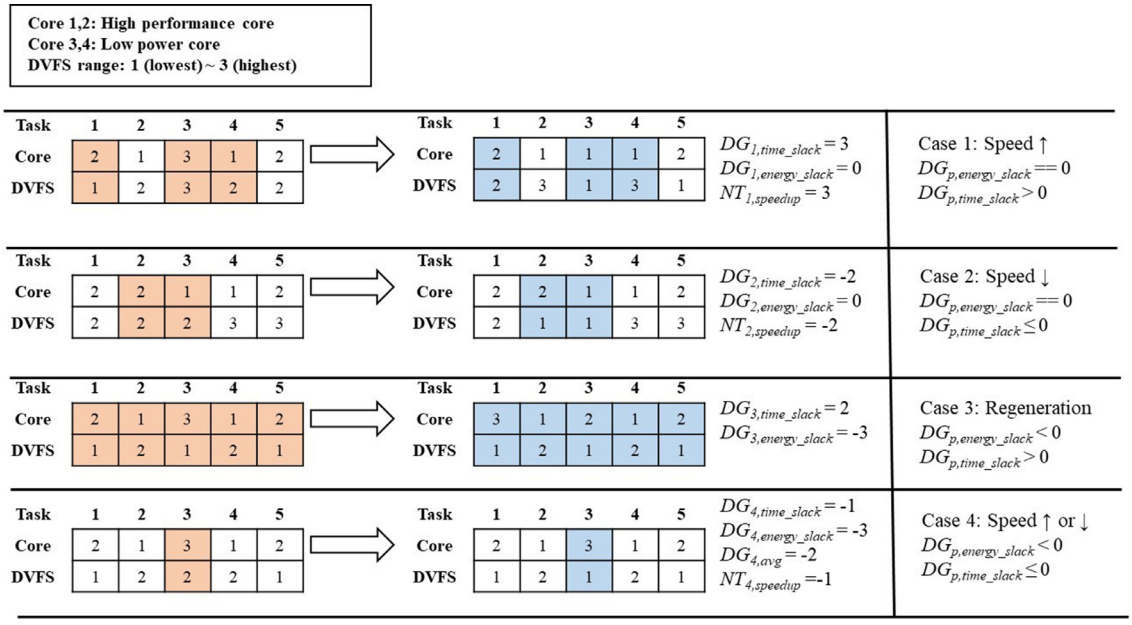


Fig. 8. Examples of the AG strategy.

and *Task 3*) to decrease their DVFS levels or to change their core types to the slower cores. *Task 2* originally ran on a faster core operating at the 2nd DVFS level. Thus, the AG strategy simply decreases the DVFS level of the task. Similarly, *Task 3* originally ran on a faster core operating at the 2nd DVFS level. Thus, the AG strategy decreases the DVFS level of *Task 3* to the 1st DVFS level. The third solution candidate in Fig. 8 has a positive $DG_{3,time\_slack}$ value of 2 and a negative $DG_{3,energy\_slack}$ value of −3. Thus, the AG strategy replaces the third solution candidate with a newly generated solution candidate. Finally, the fourth solution candidate in Fig. 8 has negative $DG_{4,time\_slack}$ and $DG_{4,energy\_slack}$ values. Thus, the AG strategy first calculates $DG_{4,avg}$. As the value of $DG_{4,avg}$ is −2, the AG strategy sets $NT_{4,speedup}$ to one random value from −1 to −2. In this example, the AG strategy sets $NT_{4,speedup}$ to −1 and thus, AGATS randomly selects one task (*Task 1*) to decrease the speed of execution. AGATS applies the AG strategy to all the remaining solution candidates in order to generate the next set of solution candidates.

4.5. Termination

The iteration process is terminated if the number of generations reaches *MaxGeneration*, or the best solution is not updated for 50% of *MaxGeneration* times consecutively.

4.6. Computational complexity

The computational complexity of AGATS is determined by two computationally dominant parts. The first part is to sort the tasks assigned to the cores. In the worst case, if all the tasks are assigned to one core, the computational complexity for sorting the tasks is $O(M\ log\ M)$. Since this sorting is performed for all solution candidates in every generation, the computational complexity for the first part is $O(G{\cdot}P{\cdot}M\ log\ M)$, where G is the number of generations. The second part is to sort the solution candidates based on their fitness value. The computational complexity for sorting the solution candidates is $O(P{\cdot}log\ P)$. Since this sorting is performed in every generation, the computational complexity of the second part is $O(G{\cdot}P{\cdot}log\ P)$. Therefore, the computational complexity of AGATS is $O(G{\cdot}P{\cdot}(M{\cdot}log\ M{+}log\ P))$.

5. Experimental results

5.1. Experimental environments

To evaluate the effectiveness of AGATS, we used two types of application sets: (1) eight application sets from Embedded Systems Synthesis Benchmarks (E3S) [29] and (2) five application sets from Task Graphs For Free (TGFF) [30]. Table 1 provides information on the application sets generated from E3S. The first five application sets in Table 1 were built using task graphs from E3S (automotive/industrial, networking, telecommunications, consumer, and office automation). The other three application sets were created by combining the task graphs of the first five application sets. As the number of tasks in the task graphs from E3S was too small to demonstrate the effectiveness of AGATS, we created application sets by combining the corresponding task graphs. For

Table 1
Eight application sets composed of task graphs from the E3S benchmark suite.

Application set	Task graph	# of task graphs	Total # of tasks	Total # of edges
1	auto-indust0-3	4	96	84
2	consumer0-1	2	24	24
3	networking0-3	2	26	18
4	office-automation0	4	20	20
5	telecom0-8	2	60	48
6	consumer0-1, networking0-3	1	25	21
7	auto-indust0-3, office-automation0, telecom0-8	2	118	100
8	auto-indust0-3, consumer0-1, networking0-3, office-automation0, telecom0-8	1	84	71

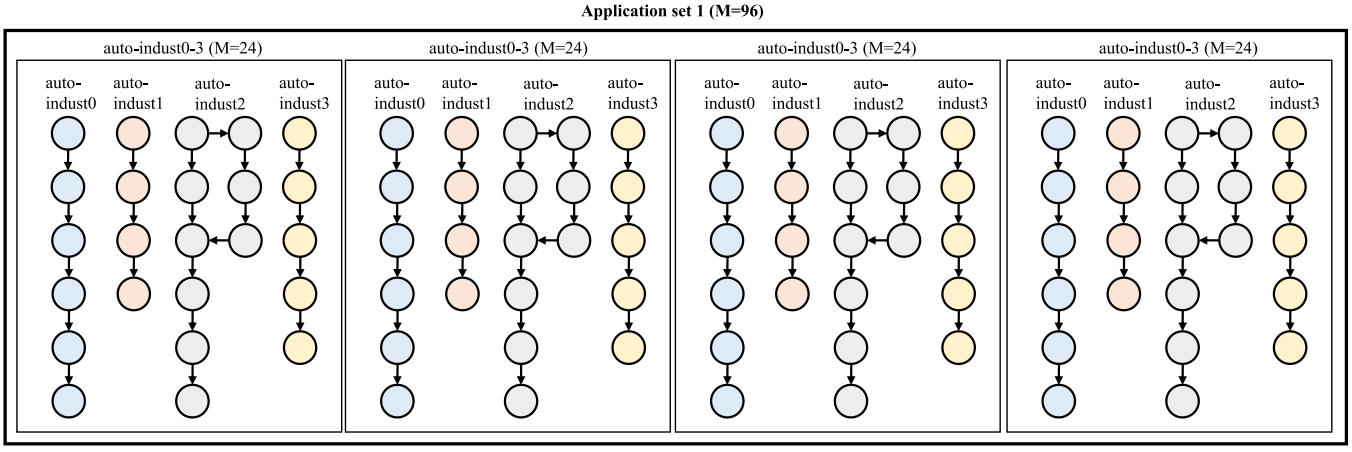


Fig. 9. Composition of application set 1.

Table 2
Five application sets composed of task graphs from TGFF.

Application set	Task graph	Total # of tasks	Total # of edges
1	Random task graph	161	212
2	Random task graph	217	283
3	Random task graph	269	321
4	Series-parallel task graph	153	182
5	Series-parallel task graph	230	264

example, application set 1 consisted of four types of task graphs from E3S (auto-indust0 to auto-indust3) and had four task graphs that were completely independent from each other, as shown in Fig. 9. Therefore, each task graph created by auto-indust0-3 had 24 tasks, indicating that application set 1 had 96 tasks in total. The other application sets from E3S were created similarly. Table 2 provides information on the application sets generated by TGFF. TGFF supports two types of task graphs: random task graph and series-parallel task graph. Using TGFF, we generated three random task graphs (application set 1–3) and two series-parallel task graphs (application set 4–5) for the experiments.

We obtained the real data of static power, execution time of each task and energy consumption of each task for PowerPC 750CX, provided by the E3S benchmark suite [29]. The target asymmetric MPSoC used in the experiments contained two types of cores: LPcores and HPcores. In the experiments, we used ARM Cortex-A53 as an LPcore and ARM Cortex-A57 as an HPcore. We set three DVFS levels for each type of core in their available DVFS levels: LPcores (450, 575, and 700 MHz) and HPcores (800, 950, and 1100 MHz). In order to consider the core differences between PowerPC 750CX [29] and the cores in the target asymmetric MPSoCs (ARM Cortex-A53 and ARM Cortex-A57), we adjusted the required execution time and energy consumption of each task from the PowerPC 750CX to the target cores by scaling them according to their voltage, frequency, and performance/energy efficiency between the HPcore and LPcore. E_{Router} and E_{Link} were set to 0.284 and 0.449 nJ/bit, respectively. P was set to 1000 and *MaxGeneration* was set to 500.

In order to demonstrate the effectiveness of AGATS, the following two GA-based benchmark algorithms were used: recent GA-based task scheduling algorithm (RGA) [9], and a second GA-based task scheduling algorithm (GeneS) [7] which considers the MPSoCs that support per-core DVFS. As RGA cannot be used for DVFS-enabled systems, we compared AGATS with RGA for the systems that do not support the DVFS. In other words, the systems support only one DVFS level. As GeneS can be used for DVFS-enabled systems, we compared AGATS with GeneS for those systems that support three DVFS levels. We established three timing constraints (tight: 1.5 times the critical path, normal: 2.0 times the critical path, and loose: 2.5 times the critical path) for the experiments [10]. As both AGATS and benchmark algorithms use randomness during the optimization process, the experiments were conducted 100 times repeatedly. We then took the average of the results.

5.2. Parameter sensitivity

One of the key points of AGATS is to use the three strategies, including elitism, MOE, and AG, in a balanced way. In order to determine the degree of each strategy, we performed experiments by varying N_{elites} and $N_{mutation}$ values. In the experiments, we analyzed the effects of N_{elites} and $N_{mutation}$ values on the optimality when mapping the E3S application sets with tight timing constraints to a 10-core asymmetric MPSoC system. The 10-core asymmetric MPSoC consists of five HPcores and five LPcores. Fig. 10(a) shows the energy consumption comparison for various N_{elites} values when the $N_{mutation}$ value was fixed to 100. Fig. 10(b) shows the energy consumption comparison for various $N_{mutation}$ values when the N_{elite} value was fixed to 50. As shown in Fig. 10(a) and (b), local optimums were produced when the solution candidates were evolved only by AG (N_{AG}=1000). In addition, when the solution candidates were evolved without AG (N_{AG}=0), the solutions were less optimized. However, when using the three strategies together, AGATS escaped from the local optimums and identified high quality solutions. From the experiments, we found that N_{elites} produced the best results in the range of 10–100. We also found that N_{mutation} produced the best results in the range of 10–300. Therefore, We set the N_{elites} and $N_{mutation}$ values to 50 and 100, respectively.

5.3. Energy consumption evaluation

To show the effectiveness of AGATS, we compared AGATS with the benchmark algorithms. Table 3 shows the energy consumption comparison between AGATS and the two benchmark algorithms for a 6-core asymmetric MPSoC system containing three HPcores and three LPcores. The results show that AGATS outperformed the benchmark algorithms in most cases. Especially, AGATS reduced the energy consumption by up to 13.8% and 14.9% compared to RGA and GeneS, respectively, for a system that does not support per-core DVFS. When the target system supports three-level per-core DVFS, AGATS reduced the energy consumption by up to 17.0% compared to GeneS. Furthermore, as the AG strategy effectively

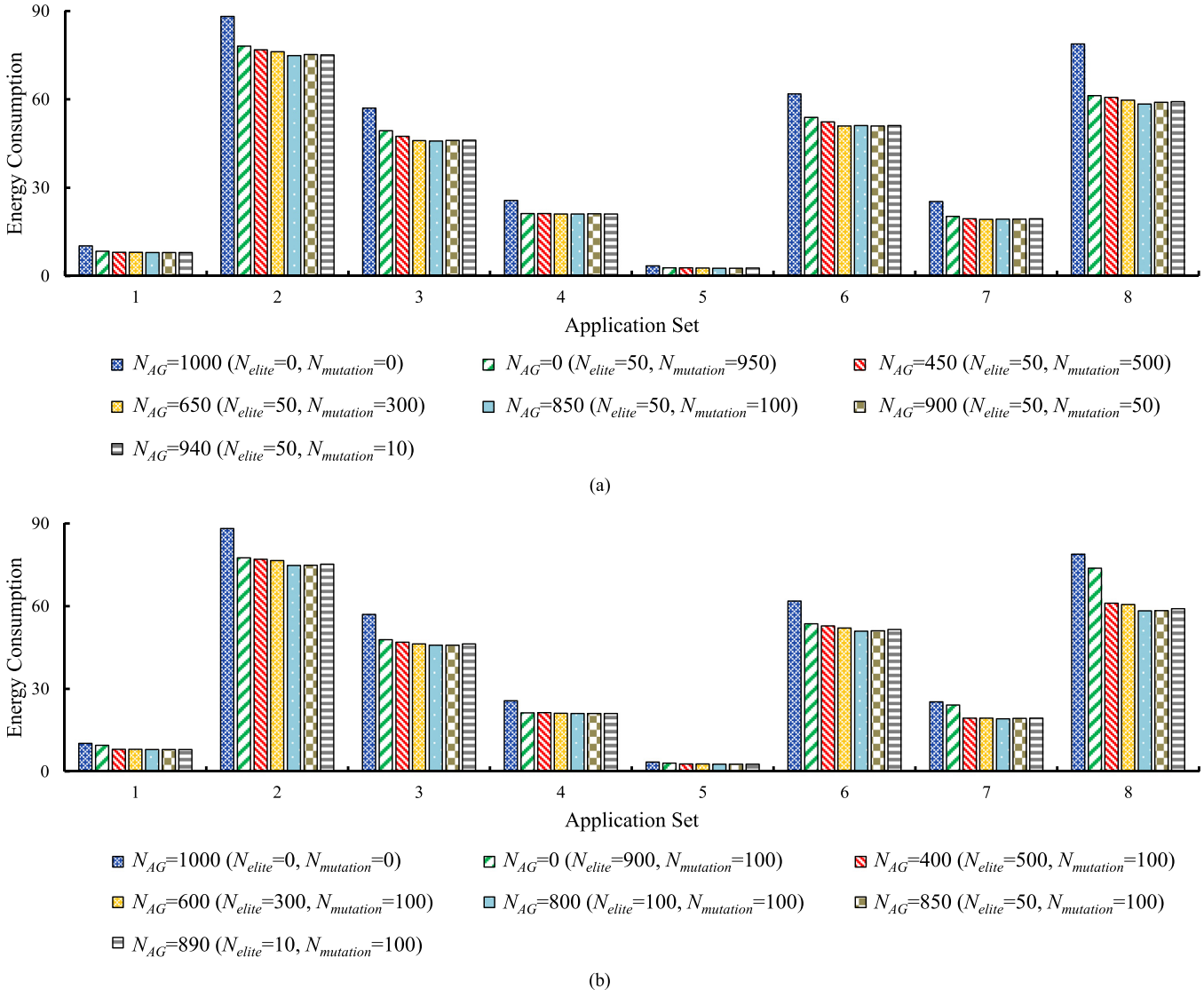


Fig. 10. (a) Energy consumption comparison for various Nelite values when the Nmutation value was fixed to 100. (b) Energy consumption comparison for various Nmutation values when the Nelite value was fixed to 50.

reduces the completion time of solution candidates that do not satisfy the timing constraint, AGATS found feasible solutions more effectively than did GeneS and RGA.

Tables 4 and 5 present the energy consumption comparison between AGATS and the benchmark algorithms for an 8-core (4 HPcores + 4 LPcores) and 10-core (5 HPcores + 5 LPcores) asymmetric MPSoC system, respectively. Although its underlying hardware architecture was changed, AGATS found higher-quality solutions than did the benchmark methods. As shown in Table 4, AGATS reduced the energy consumption by up to 14.5% and 21.2% compared to RGA and GeneS, respectively, for an 8-core system that does not support per-core DVFS. For the 8-core system that supports three-level per-core DVFS, AGATS reduced the energy consumption by up to 20.4% compared to GeneS. For the 10-core system that does not support per-core DVFS, AGATS reduced the energy consumption by up to 17.8% and 23.2% compared to RGA and GeneS, respectively. For the 10-core system that supports three-level per-core DVFS, AGATS reduced the energy consumption by up to 20.3% compared to GeneS. AGATS achieved even greater energy reduction when the target system contained more processing cores. This indicates that AGATS can be more effective than the benchmark algorithms for complex systems. Moreover, AGATS still identified feasible solutions effectively compared to the benchmark approaches, when the target system contained more processing cores.

Table 6 shows the energy consumption comparison between AGATS and GeneS for applications generated by TGFF. Since the size of the task graphs from TGFF is large, we used 10-core (5 HPcores + 5 LPcores), 16-core (8 HPcores + 8 LPcores), and 20-core (10 HPcores and 10 LPcores) asymmetric MPSoCs for the experiments. As shown in Table 6, for all cases, AGATS consumed less energy and identified feasible solutions effectively compared to GeneS. AGATS reduced energy consumption by up to 18.2%, 26.2%, and 29.3% compared to GeneS for 10-core, 16-core and 20-core asymmetric MPSoC, respectively. From the experiments, we found that AGATS maintained the effectiveness of finding high quality solutions for various task graphs and architectures.

As presented by Tables 3–6, AGATS preserves the elite solutions during the optimization process using an elitism strategy and effectively explores the design space of the system by using the MOE and AG strategies. Using the three generation strategies, AGATS not only identifies higher quality solutions than the benchmark algorithms but also effectively determines feasible solutions under tight timing constraints. Thus, AGATS can

Table 3
Energy consumption for the eight application sets with three timing constraints for the 6-core MPSoC system.

Application set (E3S)	TC	No DVFS (mJ)			3-level DVFS (mJ)	
		RGA	GeneS	AGATS	GeneS	AGATS
1	Tight	Fail	Fail	11.7	Fail	10.4
	Normal	Fail	Fail	10.9	Fail	7.7
	Loose	Fail	11.3	9.6	8.7	7.2
2	Tight	Fail	Fail	Fail	Fail	Fail
	Normal	103.7	104.7	102.4	75.3	72.7
	Loose	100.8	103.9	98.4	67.8	66.7
3	Tight	Fail	Fail	66.4	Fail	Fail
	Normal	Fail	Fail	61.6	Fail	43.7
	Loose	Fail	61.3	57.0	43.3	40.1
4	Tight	Fail	Fail	Fail	Fail	Fail
	Normal	32.4	32.1	32.2	Fail	24.4
	Loose	32.2	30.5	32.2	21.0	21.0
5	Tight	Fail	Fail	Fail	Fail	Fail
	Normal	3.9	Fail	3.6	Fail	2.6
	Loose	3.9	Fail	3.6	Fail	2.6
6	Tight	79.8	82.8	73.8	61.2	55.5
	Normal	73.5	72.5	70.2	50.8	49.1
	Loose	68.9	68.6	66.5	46.2	46.5
7	Tight	Fail	Fail	29.7	Fail	Fail
	Normal	29.3	30.4	27.9	22.7	19.5
	Loose	28.4	29.2	26.2	20.2	18.8
8	Tight	96.5	Fail	89.2	Fail	68.1
	Normal	92.1	93.3	81.5	68.1	59.0
	Loose	87.6	87.1	75.5	61.5	55.8

Fail = at least once, the algorithm failed to find a feasible solution after 100 experiments.

Table 4
Energy consumption for the eight application sets with three timing constraints for the 8-core MPSoC system.

Application set (E3S)	TC	No DVFS (mJ)			3-level DVFS (mJ)	
		RGA	GeneS	AGATS	GeneS	AGATS
1	Tight	Fail	Fail	11.6	Fail	7.8
	Normal	Fail	12.5	10.4	9.1	7.7
	Loose	Fail	10.9	8.6	8.5	6.8
2	Tight	Fail	Fail	106.4	Fail	77.3
	Normal	102.9	104.0	101.9	73.8	71.3
	Loose	101.6	103.8	96.6	67.5	65.9
3	Tight	Fail	Fail	64.2	Fail	50.2
	Normal	Fail	67.2	58.4	47.8	42.0
	Loose	59.6	58.9	51.5	41.2	37.7
4	Tight	30.8	Fail	30.7	Fail	21.2
	Normal	30.7	30.9	30.7	21.4	20.5
	Loose	30.7	29.7	30.7	20.5	19.9
5	Tight	Fail	Fail	4.0	Fail	2.7
	Normal	3.7	3.9	3.3	2.9	2.4
	Loose	3.7	3.9	3.3	2.9	2.4
6	Tight	76.4	80.5	71.9	58.4	51.6
	Normal	70.6	69.6	66.5	49.1	47.5
	Loose	67.0	67.2	64.7	44.7	44.1
7	Tight	Fail	Fail	28.0	Fail	20.3
	Normal	27.9	29.6	25.7	20.7	18.6
	Loose	26.5	28.2	23.5	19.6	17.5
8	Tight	91.5	Fail	82.8	74.9	62.2
	Normal	87.4	90.0	74.7	64.4	55.2
	Loose	81.9	84.4	72.6	58.7	51.6

Fail = at least once, the algorithm failed to find a feasible solution after 100 experiments.

evolve solution candidates toward better solutions, whereas the existing algorithms tend to excessively depend on randomness. Although we assume that the contention overhead within the NoC is negligible, AGATS showed much better mapping performance than the benchmark approaches. Thus, the solution produced by AGATS will likely perform best under the real contention effect.

To show the scalability of AGATS, we conducted experiments for various asymmetric MPSoCs. Fig. 11 shows the energy consumption of the mapping solutions produced by AGATS and GeneS when the 7th and 8th E3S application sets were mapped onto the various asymmetric MPSoCs. AGATS achieved greater energy reduction than did GeneS as the number of cores increased for the different timing constraints as shown in Fig. 11. This indicates that AGATS is more scalable than GeneS. Therefore, AGATS is expected to be significantly more useful than GeneS for a future MPSoC architecture expected to have hundreds or even thousands of cores. Moreover, based on our observation, AGATS showed better performance when the task graph had more parallelism (e.g. application sets 1, 5, 7, and 8). Thus, AGATS can be effectively applied in future programs that have greater parallelism owing to improvements in the compiler or improved parallel programming skills.

Table 5
Energy consumption for the eight application sets with three timing constraints for the 10-core MPSoC system.

Application set (E3S)	TC	No DVFS (mJ)			3-level DVFS (mJ)	
		RGA	GeneS	AGATS	GeneS	AGATS
1	Tight	Fail	Fail	11.6	Fail	7.9
	Normal	Fail	12.6	9.9	9.2	7.7
	Loose	Fail	10.7	8.2	8.4	6.7
2	Tight	Fail	Fail	106.1	Fail	74.9
	Normal	102.8	103.9	101.9	72.8	70.5
	Loose	99.6	103.8	95.1	67.4	65.7
3	Tight	Fail	Fail	64.2	Fail	45.9
	Normal	Fail	66.3	56.9	46.3	41.4
	Loose	57.6	56.9	47.3	40.8	35.9
4	Tight	30.4	Fail	30.4	25.2	21.1
	Normal	30.3	30.4	29.5	21.1	20.3
	Loose	30.3	29.3	29.4	20.1	19.6
5	Tight	Fail	Fail	4.0	Fail	2.6
	Normal	3.4	3.8	3.0	2.7	2.3
	Loose	3.4	3.7	3.0	2.7	2.3
6	Tight	75.5	78.9	70.6	56.7	51.1
	Normal	68.7	68.4	65.6	48.0	46.6
	Loose	65.7	66.9	63.7	44.0	43.0
7	Tight	30.5	Fail	26.4	24.0	19.3
	Normal	27.2	28.9	23.7	20.3	17.6
	Loose	25.7	27.6	21.4	19.3	16.4
8	Tight	92.5	Fail	78.9	73.2	58.4
	Normal	85.3	87.9	72.5	63.1	52.6
	Loose	79.7	83.1	72.3	56.5	48.5

Fail = at least once, the algorithm failed to find a feasible solution after 100 experiments.

Table 6
Energy consumption for the five application sets from TGFF with three timing constraints.

Application set (TGFF)	TC	10-core system energy (mJ)		16-core system energy (mJ)		20-core system energy (mJ)	
		GeneS	AGATS	GeneS	AGATS	GeneS	AGATS
1	Tight	Fail	128.0	Fail	119.2	Fail	111.9
	Normal	Fail	114.1	138.8	102.5	133.1	94.9
	Loose	120.1	102.5	115.8	87.5	115.0	81.3
2	Tight	Fail	219.2	Fail	200.4	Fail	197.6
	Normal	Fail	196.4	Fail	188.4	243.7	183.6
	Loose	232.5	190.3	223.1	172.4	223.4	163.7
3	Tight	Fail	Fail	Fail	439.5	Fail	404.5
	Normal	Fail	Fail	Fail	376.8	Fail	365.9
	Loose	Fail	Fail	Fail	351.3	Fail	330.7
4	Tight	Fail	Fail	Fail	237.9	Fail	234.3
	Normal	Fail	208.3	Fail	200.4	Fail	197.4
	Loose	Fail	191.6	252.9	182.9	253.6	179.5
5	Tight	Fail	Fail	Fail	Fail	Fail	Fail
	Normal	Fail	Fail	Fail	323.2	Fail	322.9
	Loose	Fail	Fail	Fail	278.1	Fail	277.8

Fail = at least once, the algorithm failed to find a feasible solution after 100 experiments.

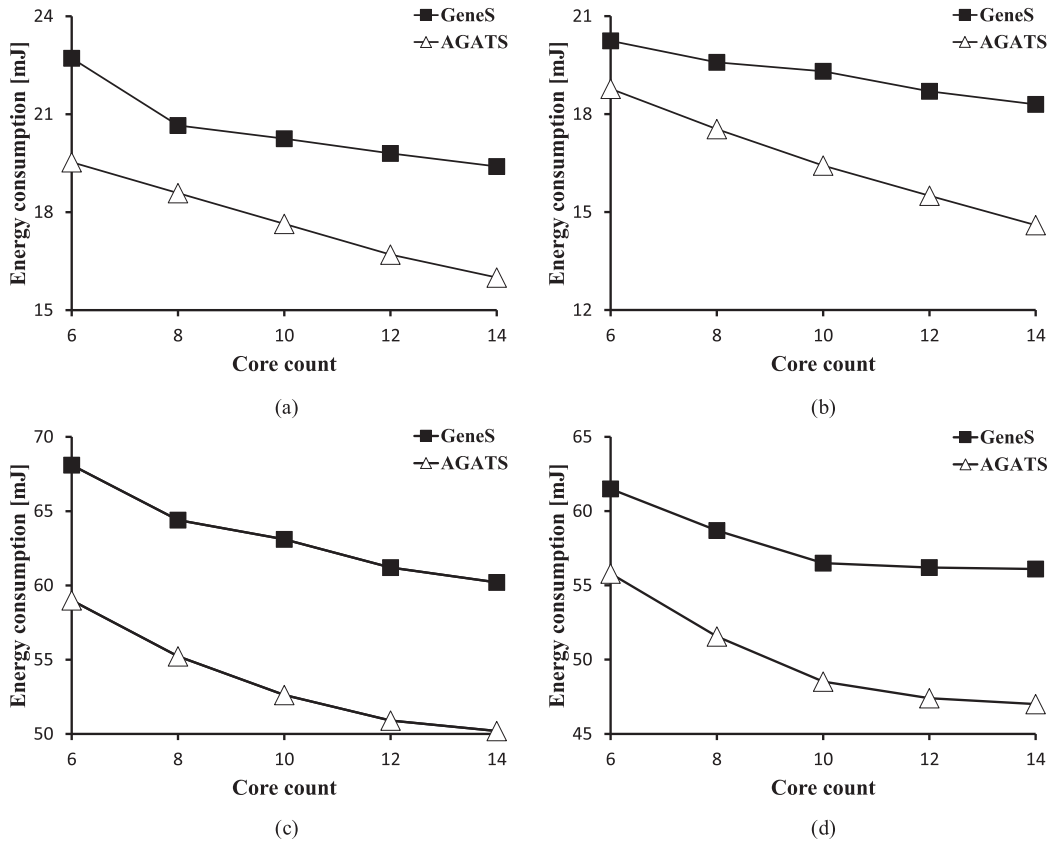


Fig. 11. Energy consumption of AGATS and GeneS for the 7th and 8th application set as the core count is increased with: (a) 7th application set with normal timing constraint, (b) 7th application set with loose timing constraint, (c) 8th application set with normal timing constraint, and (d) 8th application set with loose timing constraint.

6. Conclusion

We developed an energy-efficient design-time task scheduling algorithm, AGATS, for an asymmetric MPSoC. AGATS maintains solution quality by adapting the elitism strategy. In order to maintain the diversity of the elite solution candidates, next solution candidates are then generated using the MOE strategy. Finally, AGATS adaptively generates next solution candidates based on the time slack and energy slack of each solution candidate by using the AG strategy. Experimental results showed that AGATS reduced the energy consumption by up to 29.3% and effectively found feasible solutions under tight timing constraints compared to the benchmark methods. Furthermore, AGATS effectively reduced the energy consumption when the number of cores in the system increased. Therefore, AGATS is expected to be widely used to design energy-efficient embedded systems based on an asymmetric MPSoC.

Acknowledgement

This research was supported by Samsung Electronics Co., Ltd., IDEC, and the MSIT (Ministry of Science and ICT), Korea, under the "ICT Consilience Creative program" (IITP-2018-2011-1-00783) supervised by the IITP (Institute for Information & Communications Technology Promotion).

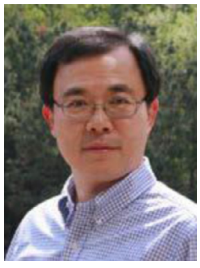
References

[1] W. Quan, Scenario-based run-time adaptive Multi-Processor System-on-Chip, 2015 Ph.D dissertation.
[2] W. Wolf, A. Jerraya, G. Martin, Multiprocessor system on-chip (MPSoC) technology, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 27 (10) (Oct. 2008) 1701–1713.
[3] Texas Instruments, OMAPTM Application Processors, 2013. http://www.ti.com/lsds/ti/omap-applicationsprocessors/features.page.
[4] P. Greenhalgh, Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7: iImproving energy efficiency in high-performance mobile platforms, 2011. http://www.arm.com/files/downloads/bigLITTLEFinalFinal.pdf.
[5] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[6] T. Braun, et al., A Comparison of eleven static heuristics for scheduling a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[7] Y. Wang, et al., Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip, ACM Trans. Des. Autom. Electron. Syst. 16 (2) (2011).
[8] S. Saha, et al., Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems, in: Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, IEEE, 2012.
[9] E. Hwang, Task scheduling for energy efficient MPSoC design, 2013 Ph.D dissertation.
[10] H. Kim, S. Kang, Communication-aware task scheduling and voltage selection for total energy minimization in a multiprocessor system using ant colony optimization, Inf. Sci. vol.181 (18) (2011).
[11] E. Carvalho, N. Calazans, F. Moraes, Dynamic task scheduling for MPSoCs, IEEE Des. Test 27 (5) (2010).
[12] S. Hong, et al., Process variation aware thread scheduling for chip multiprocessors, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2009.
[13] F. Wang, et al., Variation-aware task and communication scheduling for mpsoc architecture, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 30 (2) (2011).
[14] H. Orsila, et al., Automated memory-aware application distribution for multi--processor system-on-chips, J. Syst. Archit. 53 (11) (2007).
[15] S. Manolache, P. Eles, Z. Peng, Task scheduling and priority assignment for soft real-time applications under deadline miss ratio constraints, ACM Trans. Embed. Comput. Syst. 7 (2) (2008).
[16] A. Al Badawi, A. Shatnawi, Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization, Comput. Oper. Res. 40 (10) (2013).
[17] Z. Qiu, J.F. Pérez, Evaluating replication for parallel jobs: an efficient approach, IEEE Trans. Parallel Distrib. Syst. 27 (8) (2016) 2288–2302.

[18] L. Huang, F. Yuan, Q. Xu., On task allocation and scheduling for lifetime extension of platform-based MPSoC designs, IEEE Trans. Parallel Distrib. Syst. 22 (12) (2011) 2088–2099.
[19] C. Radu, L. Vintan, Optimized algorithms for network-on-chip application scheduling, 2011 Ph.D dissertation.
[20] J.J. Han, et al., Contention-aware energy management scheme for NoC-based multicore real-time systems, IEEE Trans. Parallel Distrib. Syst. 26 (3) (2015) 691–701.
[21] K. Li, et al., Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems, IEEE Trans. Comput. 64 (1) (2015) 191–204.
[22] Y. Xu, et al., A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2015) 3208–3222.
[23] M.A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Comput. 38 (4–5) (2012) 175–193.
[24] H. Arabnejad, J. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694.
[25] J. Liu, et al., Minimizing cost of scheduling tasks on heterogeneous multicore embedded systems, ACM Trans. Embed. Comput. Syst. 16 (2) (2017) 36.
[26] B. Nikolić, S.M. Petters, EDF as an arbitration policy for wormhole-switched priority-preemptive NoCs—Myth or fact? in: Proceedings of the International Conference on Embedded Software (EMSOFT), IEEE, 2014.
[27] J. Hu, R. Marculescu, Energy-aware mapping for tile-based NoC architectures under performance constraints, in: Proceedings of the Asia and South Pacific Design Automation Conference, ACM, 2003.
[28] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evol. Comput. 8 (2) (2000) 173–195.
[29] The Embedded System Synthesis Benchmarks Suite (E3S) website. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s.
[30] K. Vallerio, Task Graphs for Free (TGFF v3.0), 2008. [Online]. Available: http://ziyang.eecs.northwestern.edu/dickrp/tgff/manual.pdf.

Yonghee Yun received the B.S. degree in electrical engineering in 2014 from Dankook University, Rep. of Korea, and is currently pursuing a Ph.D. degree at Pohang University of Science and Technology, Rep. of Korea. His current research interests include MPSoC design methodology, and system-level power/thermal analysis and management.

Eun Ju Hwang received the B.E. degree in electrical engineering and computer science in 2008 from Kyungpook National University, Republic of Korea and the Ph.D. degree in electrical engineering in 2014 from Pohang University of Science and Technology, Republic of Korea. Since her graduation, she has worked as a senior researcher at Samsung Electronics, Republic of Korea. Her current research interests include all aspects of low power design, especially near-threshold voltage design methodology and system-level power analysis and management.



Young Hwan Kim (S'86-M'89-SM'14) received the B.E. degree in electronics in 1977 from the Kyungpook National University, Rep. of Korea, and M.S. and Ph.D. degrees in electrical engineering in 1985 and 1988 from the University of California, Berkeley, USA. From 1977 to 1982, he was with the Agency for Defense Development, Republic of Korea, where he was involved in various military research projects, including the development of autopilot guidance and control systems. From 1983 to 1988, he worked as a Post Graduate Researcher, developing VLSI CAD programs at the Electronic Research Laboratory of the University of California, Berkeley, USA. He is currently a Professor in the Division of Electronic and Computer Engineering at Pohang University of Science and Technology, Republic of Korea. His research interests include plasma and liquid crystal display systems, multimedia circuit design, MPSoC and GPGPU system design for display and computer vision applications, statistical analysis and design technology for deep-submicron semiconductor devices, and power noise analysis. He has served as an editor of the Journal of the Institute of Electronics Engineers of Korea, and as General Chair and a committee member of various Korean domestic and international technical conferences, including International SoC Design Conference and IEEE ISCAS 2012.