IPSJ Transactions on System LSI Design Methodology Vol.12 65–73 (Aug. 2019)

[DOI: 10.2197/ipsjtsldm.12.65]

Regular Paper

Communication-Aware Scheduling of Data-Parallel Tasks on Multicore Architectures

Kana Shimada[1,a)] Ittetsu Taniguchi[2,b)] Hiroyuki Tomiyama[1,c)]

Received: November 30, 2018, Revised: March 8, 2019,
Accepted: April 22, 2019

Abstract: Task scheduling has a significant impact on multicore computing systems. This paper studies scheduling of data-parallel tasks on multicore architectures. Unlike traditional task scheduling, this work allows individual tasks to run on multiple cores in a data-parallel fashion. In this paper, the inter-task communication overhead is taken into account during scheduling. The communication happens if main threads of two tasks with data-dependencies are mapped onto the different processors. This paper proposes two methods for data-parallel task scheduling with communication overhead. One is two-step method, which schedules tasks without communication and then assigns threads in the task on cores. The other is integrated method, which performs task scheduling and thread assignment simultaneously. Both of the two methods are based on integer linear programming. The proposed methods are evaluated through experiments and encouraging results are obtained.

Keywords: task scheduling, integer linear programming, multicore

1. Introduction

Multicore computing attracts an increasing attention because of its better power/performance efficiency than single-core computing. In multicore computing, task scheduling, which assigns tasks to cores and decides the execution order of the tasks on each core, has a significant impact on the system performance. Therefore, multicore task scheduling has been extensively studied in several decades. A traditional task scheduling problem assumes that each task is executed on a single core. Prior algorithms for the problem try to execute as many tasks as possible in parallel on multiple cores, in order to minimize the overall schedule length (a.k.a. makespan). Since task scheduling is an NP-hard problem, it is difficult to find exact solutions in a practical time and it consumes a large amount of memory and computing resources [1], [2], [3]. Therefore, many heuristic approaches for task scheduling have been proposed [4], [5], [6], [7], [8]. Some researchers have extended the scheduling problem in such a way that a task may use multiple cores [9]. Liu et al. proposed heuristic algorithms to minimize the scheduling length of a given set of dependent data-parallel tasks [10], [11], [12]. They allow individual tasks to use multiple cores. Turek et al. also allows tasks to run on multiple cores [13]. The work developed polynomial time algorithms for a multiprocessor scheduling problem of parallelizable task. Yang and Ha's work proposed a multi-task mapping/scheduling technique for scalable MPSoC [14]. The solution is based on integer linear programming (ILP). They aim at minimization of hardware cost, while satisfying the deadline constraints of the tasks. In Ref. [15], the task scheduling technique based on ILP was proposed. Each task is executed on multiple cores and many tasks are executed on multiple cores in parallel. Their work in Ref. [15] allows a task to run on multiple cores at a synchronous manner, where the task starts and finishes the execution on the multiple cores at the same time. They aim at minimization of schedule length. The ILP-based task scheduling on multicore was also proposed in Ref. [16]. Different from the work in Ref. [15], tasks are split into sub-tasks and they are scheduled independently. The objective in Ref. [16] is the minimization of schedule length. Chen and Chu developed an approximation algorithm for multicore task scheduling [17]. Their algorithm found feasible scheduling results for a set of dependent data-parallel tasks. The work considers the intra-task overhead such as communication and synchronization inside the tasks.

The communication overhead among tasks is not taken into account in Refs. [10], [11], [12], [13], [14], [15], [16], [17]. However, the overhead happens in reality. There are some other works which consider the inter-task communication overhead. Roel et al. describe a design flow to map a throughput-constrained application on MPSoC [18]. SDF modelling in their work includes communication via the interconnect. They implemented the practical application on MPSoC. Hwang et al. assume the scheduling problem for non-preemptive tasks on identical processors subject to inter-processor communication delays [19]. This work designed and analyzed a new heuristic, called Earliest Task First (ETF). They assumed that each task is executed on one processor in n identical processors. In the work of Ref. [20], A* scheduling

1 Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
2 Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
a) kana.shimada@tomiyama-lab.org
b) i-tanigu@ist.osaka-u.ac.jp
c) ht@fc.ritsumei.ac.jp

algorithm is proposed in order to minimize the schedule length. The algorithm can find optimal schedules in reasonable time for task graphs with 40 nodes. If two tasks with data dependencies are mapped onto the same processor, the communication between them is implemented by data sharing in local memory and no communication delay is incurred. Venugopalan et al. proposed an ILP based approach to find exact results for a task scheduling problem with communication delays [21]. Communication delay in Ref. [21] is based on Ref. [20]. The work also aims at the minimization of schedule length. In Refs. [20], [21], no multitasking or parallelism is permitted within a task. The work in Ref. [22] proposes the list scheduling algorithm taking into account communication overhead in distributed memory machine. A task is executed on a processor, and communication time is required if two task which have dependent relation between them are assigned on the different processors. Fujita et al. propose two techniques for obtaining a sharp lower bound on makespan for multicore task scheduling with communication delay [23]. In their work, multiprocessor systems consist of identical processors, and a unit of task is executed on a processor. It takes communication delay between tasks with dependent flow if they are assigned to different processors. The work in Ref. [24] presents a heuristic algorithm for obtaining satisfactory suboptimal schedules. Their algorithm is based on a classical list scheduling. The aim in the work is the minimization of completion time. Each task is executed on a processor and the communication channels are to be half-duplex. An ILP based approach is proposed for integrated hardware/software partitioning and pipelined scheduling of embedded systems in Ref. [25]. Computation tasks are mapped to hardware or software components of the target architecture. The problem is to map computation tasks to SW or HW components. Communication time is set to zero if tasks with data dependence are mapped to the same component. As described above, many researchers have studied multicore task scheduling problems with communication overhead. However, they assume that each task is executed on a single processor without data parallelism.

This paper studies scheduling of data-parallel tasks with taking into account communication time. We propose two scheduling approaches. Based on ILP, this work schedules the tasks in such a way that the overall schedule length is minimized. To our knowledge, this is the first paper to address scheduling of dependent data-parallel tasks with communication overhead.

This paper is organized as follows. Section 2 defines the scheduling problem solved in this paper. Section 3 and 4 present our scheduling methods and an ILP formulation of our task scheduling problem. Section 5 evaluates our scheduling technique. Finally, Section 6 concludes this paper.

2. Scheduling Problem of Data-Parallel Tasks with Communication Time

2.1 Problem Definition

In this section, we define the scheduling problem for data-parallel with communication time.

The tasks that are to be scheduled are represented as a weighted directed acyclic graph such as **Fig. 1**. The nodes represent tasks and are associated with two values, data parallelism and the execution time. The directed edges represent data dependency between tasks. The edge cost is the communication time between two tasks. In this problem, a task consists of one or more thread(s) such as **Fig. 2**. In case of multiple threads, there is a single main-thread and zero or more sub-thread(s). The main-thread partitions data into multiple pieces, and distributes them to the sub-threads for data-parallel execution. There may be inter-thread communication and synchronization during execution of the threads. Then, the computation results of the sub-threads are gathered to the main-thread, and the overall result data is passed to the successor task. Such overheads for data-parallel execution within the task are assumed to be included in the execution time of the task. We also assume that no preemption is allowed, and all of the main-thread and the sub-threads start and finish on multiple cores at the same time in a synchronous manner. Communication overhead happens if the main-thread of the predecessor task and that of the successor task are mapped to different cores.

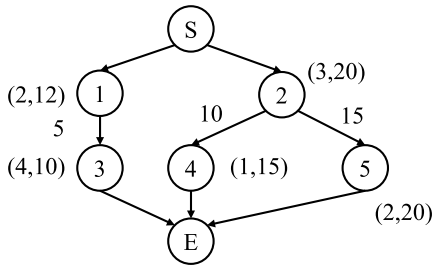

Fig. 1 An example of task-graph.

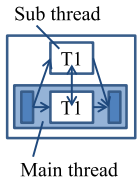

Fig. 2 A data-parallel task.

For simplicity, this paper assumes that cores are fully connected and no conflict happens on the communication network. However, this work can easily extend for shared communication links on which communication traffics conflict with each other, by treating the communication links as shared resources in a similar way to CPU cores during scheduling.

A set of dependent data-parallel tasks and a set of homogeneous cores are given in advance as part of the task graph. This work decides (a) mapping of the tasks onto the cores, and (b) the start time of each task. The aim is the minimization of schedule length. A task consists of one or more thread(s).

2.2 Scheduling Example

An example for scheduling in this work is shown in Figs. 1 and 2. A set of tasks is shown in Fig. 1. A schedule result for the task-graph in Fig. 1 is shown in **Fig. 3**, where main-threads are drawn in blue. Since the main-threads of tasks 1 and 3 are mapped to the same core, there is no communication overhead from tasks 1 to 3. On the other hand, since the main-threads of tasks 2 and 5 are mapped to different cores, task 5 is executed after the execution time of task 2 plus the communication time, and thus the start time of task 5 becomes 47 time-units (= 32 + 15).

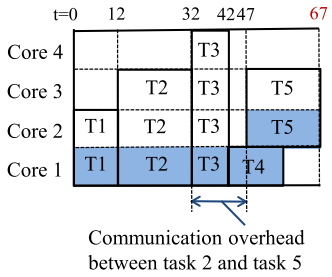

Fig. 3 An example of scheduling result.

Then, the overall length is 67 time-units.

3. Two-Step Approach to Scheduling and Thread Assignment

In this section, we propose a two-step scheduling approach (TSS). There are two steps in TSS. In the first step, TSS schedules tasks without considering communication overhead. In this step, each task is assigned multiple cores, but main-threads and sub-threads are not distinguished. Then, in the second step, main-threads are selected in such a way that the overall communication overhead is minimized.

3.1 Scheduling Example for TSS

Figure 4 shows an example of scheduling result for Fig. 1. The left Gantt chart in Fig. 4 describes a scheduling result in the first-step scheduling. The other represents scheduling result in the second-step scheduling. In the first step, task 5 is mapped to cores 2 and 3. Then, the main threads of all tasks are not distinguished. The main threads are decided in the second step. Task 5 is executed the communication time after the execution of task 2 has been completed. That is because the main threads of tasks 2 and 5 are assigned to different cores.

3.2 Formulation of the First-Step Scheduling

The first-step scheduling technique is based on ILP in Ref. [10], and scheduling solutions are obtained with ILP. We review the ILP formulation in Ref. [10] which is used in the first step in this section.

Let $map_{i,j}$ denote mapping information of task $i \in \{1, \ldots, Ntask\}$, where $Ntask$ represents the number of tasks. $map_{i,j}$ is 1 if task i is assigned to core $j \in \{1, \ldots, Ncore\}$, where $Ncore$ represents the number of cores. Otherwise, $map_{i,j}$ is 0. Let Par_i denote the digree of data parallelism of task i. Par_i is assumed to be given. The mapping constraint for each task is expressed as follows.

$$\forall i, \qquad \sum_j map_{i,j} = Par_i \tag{1}$$

Let $Time_i$ denote the execution time of task i. We assume that $Time_i$ is given. Let $start_i$ and $finish_i$ denote the start time and the finish time of task i. Note that $start_i$ is a decision variable and $finish_i$ is a dependent viriable defined by the following equation.

$$\forall i, \qquad finish_i = start_i + Time_i \tag{2}$$

If two tasks $i1$, $i2 \in i$, are mapped to the same core, the execution of the two tasks cannot be overlapped in the same time. This resource constraint is following formula.

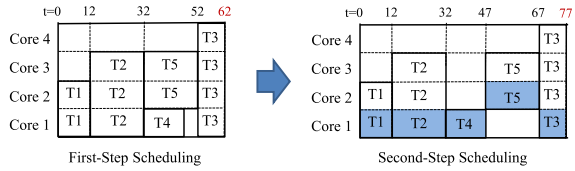

Fig. 4 An example of scheduling result for TSS.

$$\begin{aligned} \forall i1,\ i2,\ j, \qquad & map_{i1,j} + map_{i2,j} \le 1 \\ \vee \quad & finish_{i1} \le start_{i2} \\ \vee \quad & finish_{i2} \le start_{i1} \end{aligned} \tag{3}$$

This work assumes a set of dependent tasks; that is, the tasks may have a precedence dependency. Let $Flow_{i1,i2}$ denote a precedence dependency between tasks $i1$ and $i2$. $Flow_{i1,i2}$ is 1 when task $i1$ must be finished before task $i2$ starts. Otherwise, $Flow_{i1,i2}$ is 0. We assume that $Flow_{i1,i2}$ is given. Then, the precedence constraint is expressed as follows.

$$\forall i1,\ i2, \qquad Flow_{i1,i2} \rightarrow finish_{i1} \le start_{i2} \tag{4}$$

This work aims at minimization of the overall schedule length. Therefore, the objective function of our scheduling problem is given as follows.

$$\max_i \{finish_i\} \tag{5}$$

The first-step scheduling problem for data-parallel tasks without communication overhead is now formally defined: Given $Time_i$, Par_i and $Flow_{i1i2}$, find map_{ij} and $start_i$ which minimize the objective function (5), subject to the four constraints (1)–(4). There are $(3 \cdot Ntask^2 \cdot Ncore + 3 \cdot Ntask^2 + 3 \cdot Ntask)$ constraints and $(Ntask^2 \cdot Ncore + Ntask^2 + Ntask \cdot Ncore + 2 \cdot Ntask + 1)$ variables in total.

3.3 Formulation of the Second-Step Scheduling

In the second step, main-threads are decided in such a way that the overall communication overhead is minimized. Then, mapping information and the execution order of tasks in Section 3.2 are given.

Let $main\_map_{i,j}$ denote a 0-1 decision variable. $main\_map_{i,j}$ becomes 1 if the main-thread of task i is mapped to core j. There is only a main-thread in each task. The constraint for each task is expressed as follows.

$$\forall i, \qquad \sum_j main\_map_{i,j} = 1 \tag{6}$$

Let $sub\_map_{i,j}$ denote a 0-1 decision variable. $sub\_map_{i,j}$ becomes 1 if a sub-thread of task i is mapped to core j. Let Par_i be the degree of parallelism (the number of cores) of task i. Par_i is assumed to be given. The total of threads is equal to the degree of parallelism. The constraint is formulated by the following formula.

$$\forall i, \qquad \sum_j \{main\_map_{i,j} + sub\_map_{i,j}\} = Par_i \tag{7}$$

Mapping information is given by the first-step scheduling as $map_{i,j}$. In this section, $map_{i,j}$ is expressed as $Map_{i,j}$. If $Map_{i,j}$ is 1, the main-thread or sub-thread of task i mapping to core j is selected. The selection for threads is expressed as follows.

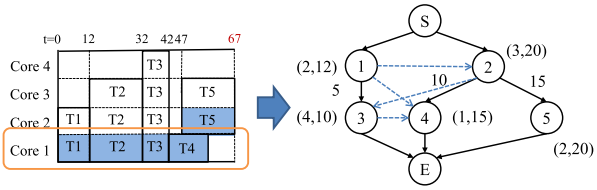

Fig. 5 An example of the order constraint.

$$\forall i,\ j, \qquad main\_map_{i,j} + sub\_map_{i,j} = Map_{i,j} \tag{8}$$

Let $Time_i$ denote the execution time of task i. $Time_i$ is assumed to be given. Let $start_i$ and $finish_i$ denote the start time and the finish time of task i, respectively. Note that $start_i$ is a decision variable and $finish_i$ is a dependent variable defined by the following equation.

$$\forall i, \qquad finish_i = start_i + Time_i \tag{9}$$

If two tasks, $i1$ and $i2$, are executed at the same time, the two tasks is mapped to the different cores. Otherwise, the execution of the two tasks cannot be overlapped. The resource constraint is formulated by the following formula.

$$\begin{aligned}&\forall i1,\ i2,\ i' \in \{i1, i2\},\ j,\\ &\sum\nolimits_{i'} (main\_map_{i',j} + sub\_map_{i',j}) \le 1\\ &\vee \quad finish_{i1} \le start_{i2}\\ &\vee \quad finish_{i2} \le start_{i1}\end{aligned} \tag{10}$$

Let $e_{i1,i2}$ be a 0-1 decision variable, which is 0 if main-threads of two tasks, $i1$ and $i2$, are mapped to the same core. In other words, $e_{i1,i2}$ becomes 1 when communication between tasks $i1$ and $i2$ happens. $e_{i1,i2}$ is given by:

$$\begin{aligned}&\forall i1,\ i2,\\ &e_{i1,i2} = \begin{cases} 0 & \text{if } max_j\{main\_map_{i1,j} + main\_map_{i2,j}\} = 2 \\ 1 & \text{otherwise}\end{cases}\end{aligned} \tag{11}$$

This work assumes a set of dependent tasks, and the tasks may have a precedence dependency. Let $C_{i1,i2}$ denote communication time between tasks $i1$ and $i2$. $C_{i1,i2}$ is assumed to be given. $C_{i1,i2}$ is 0 if there is no data dependency from tasks $i1$ to $i2$. $C_{i1,i2}$ is positive when there is the precedence constraint between two tasks. $C_{i1,i2}$ is positive when task $i1$ must be finished before task $i2$ starts. Then, the precedence constraint is expressed as follows.

$$\forall i1,\ i2, \qquad C_{i1,i2} > 0 \rightarrow finish_{i1} + C_{i1,i2} \cdot e_{i1,i2} \le start_{i2} \tag{12}$$

The execution order of tasks is given as $start_i$ in Section 3.2. It is expessed as $Start_i$ in this section. If task $i1$ is executed on core j before task $i2$ executed on core j, two tasks have a precedence dependency. **Figure 5** shows an example of the order constraint. In this figure, tasks are executed on core 1 in the order of tasks 1, 2, 3, and 4. This schedule is obtained from the first-step scheduling. Then, task 1 is executed on core 1 before task 2. Therefore, it is assumed that tasks 1 and 2 have a precedence dependency in the second-step scheduling. Similarly, data dependency between other tasks is added to task-graph such as dotted edge in Fig. 5.

The order constraint is formulated by the following.

$$\begin{aligned}&\forall i1,\ i2,\ j,\\ &Start_{i1} < Start_{i2} \cdot Map_{i2,j} \rightarrow finish_{i1} \cdot Map_{i1,j} \le start_{i2}\end{aligned} \tag{13}$$

This work aims at the minimization of the overall schedule length. The objective function of our scheduling problem to be minimized is given as follows.

$$\max_i \{finish_i\} \tag{14}$$

The second-step scheduling problem is now formally defined: Given $Time_i$, Par_i and $C_{i1,i2}$, $Map_{i,j}$ and $Start_i$ find $main\_map_{i,j}$, $sub\_map_{i,j}$ and $start_i$ which minimize the objective function (14), subject to the eight constraints (6)–(13).

Although formulas (10)–(14) are not in a linear form, they can be easily linearized by a simple transformation technique.

Formula (10) can be replaced with (15)–(21) as follows. Let $x_{i1,i2,j}$ be a 0-1 variable, being 1 if $\sum_{i' \in \{i1,i2\}} (main\_map_{i',j} + sub\_map_{i',j}) < 2$. A part of formula (11) is expressed by the following inequality.

$$\begin{aligned}&\forall i1,\ i2,\ i' \in \{i1, i2\},\ j,\\ &\sum\nolimits_{i'} (main\_map_{i',j} + sub\_map_{i',j}) - 2 \ge -U \cdot x_{i1,i2,j}\end{aligned} \tag{15}$$

$$\begin{aligned}&\forall i1,\ i2,\ i' \in \{i1, i2\},\ j,\\ &\sum\nolimits_{i'} (main\_map_{i',j} + sub\_map_{i',j}) - 2 < U \cdot (1 - x_{i1,i2,j})\end{aligned} \tag{16}$$

Here, U is large positive constant. U is sufficiently large if $U > \sum_k max_k(Time_i)$. Next, let $x_{i1,i2}$ be a 0-1 variable, being 1 if $finish_{i1} \le start_{i2}$. A part of formula (11) is linearized as follows.

$$\forall i1,\ i2, \qquad (finish_{i1} - start_{i2}) + U \cdot x_{i1,i2} > 0 \tag{17}$$

$$\forall i1,\ i2, \qquad (finish_{i1} - start_{i2}) - U \cdot (1 - x_{i1,i2}) \le 0 \tag{18}$$

Similarly, let $x_{i2,i1}$ be a 0-1 variable, being 1 if $finish_{i2} \le start_{i1}$. A last of formula (11) is linearized as follow.

$$\forall i1,\ i2, \qquad (finish_{i2} - start_{i1}) + U \cdot x_{i2,i1} > 0 \tag{19}$$

$$\forall i1,\ i2, \qquad (finish_{i2} - start_{i1}) - U \cdot (1 - x_{i2,i1}) \le 0 \tag{20}$$

Then, formula (11) can be replaced with a linear inequality as follows.

$$\forall i1,\ i2,\ j, \qquad x_{i1,i2,j} + x_{i1,i2} + x_{i2,i1} > 0 \tag{21}$$

Formula (11) can be replaced with Eqs. (22), (23) and (24) as follows. Let $y_{i1,i2,j}$ be a 0-1 variable, being 0 if main-threads of two tasks, $i1$ and $i2$, are mapped to the core j. This variable is expressed by the following inequality.

$$\begin{aligned}&\forall i1,\ i2,\ i' \in \{i1, i2\},\ j,\\ &\left(\sum\nolimits_{i'} main\_map_{i',j} - 2\right) + 2 \cdot y_{i1,i2,j} \ge 0\end{aligned} \tag{22}$$

$$\begin{aligned}&\forall i1,\ i2,\ i' \in \{i1, i2\},\ j,\\ &\left(\sum\nolimits_{i' \in \{i1,i2\}} main\_map_{i',j} - 2\right) - 2(1 - y_{i1,i2,j}) < 0\end{aligned} \tag{23}$$

Figure 6 shows an example of mapping threads to four cores. The Gantt chart on top of Fig. 6 shows the main threads of two tasks are not executed on the same core, core 1. In other words,

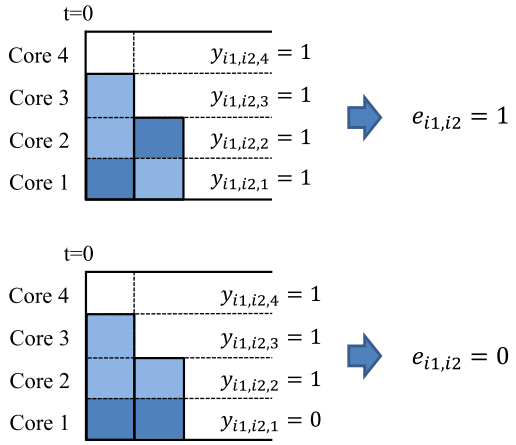

Fig. 6 Mapping main threads to cores.

there is communication overhead between tasks. $e_{i1,i2}$ is 1 if main-threads of two tasks, $i1$ and $i2$, are mapped to the different cores. Therefore, $e_{i1,i2}$ becomes 1 in the case of the top in Fig. 6. The other of Fig. 6 shows the main threads of two tasks are executed on the core 1; there is no communication overhead between tasks. Therefore, $e_{i1,i2}$ becomes 0 in the case of the other in Fig. 6. $e_{i1,i2}$ in formula (11) expressed by the following equation, using *Ncore* which is the number of cores in the target systems.

$$\forall i1,\ i2, \qquad e_{i1,i2} = \sum_j y_{i1,i2,j} - Ncore + 1 \tag{24}$$

Formula (12) can be replaced with a linear inequality as follows.

$$\forall i1,\ i2, \quad C_{i1,i2} \cdot (finish_{i1} - start_{i2} - C_{i1,i2} \cdot e_{i1,i2}) \leq 0 \tag{25}$$

Formula (13) can be replaced with Eqs. (26), (27) and (28) as follows. $S_{i1,i2,j}$ is positive if task $i2$ starts after the start time of task $i1$. $S'_{i1,i2,j}$ is positive if task $i2$ is executed after the finish time of task $i1$. When task $i2$ is executed on core j before task $i1$ has been executed on core j, the following inequality as well as inequality (28) is satisfied.

$$\forall i1,\ i2,\ j, \qquad S_{i1,i2,j} = Start_{i2} \cdot Map_{i2,j} - Start_{i1} \tag{26}$$

$$\forall i1,\ i2,\ j, \qquad S'_{i1,i2,j} = finish_{i1} \cdot Map_{i1,j} - start_{i2} \tag{27}$$

$$\forall i1,\ i2,\ j, \qquad S_{i1,i2,j} \cdot S'_{i1,i2,j} \leq 0 \tag{28}$$

The objective function (14) can be replaced with Eqs. (29) and (30) as follows. *finmax* is the objective valuable. The aim of our research is the minimization of *finmax*, together with an additional constraint (30).

$$finmax \tag{29}$$

$$\forall i, \qquad finmax - finish_i \geq 0 \tag{30}$$

Formulas (10)–(14) can be linearized as formulas (15)–(30). Thus, the task scheduling problem leads to an integer linear programming (ILP) problem. There are $(8 \cdot Ntask^2 \cdot Ncore + 6 \cdot Ntask^2 + Ntask \cdot Ncore + 4 \cdot Ntask)$ constraints and $(4 \cdot Ntask^2 \cdot Ncore + 2 \cdot Ntask \cdot Ncore + 2 \cdot Ntask^2 + 2 \cdot Ntask + 1)$ variables in total.

4. Integrated Approach to Scheduling and Thread Assignment

In this section, we propose an integrated scheduling approach (IS). Unlike TSS, scheduling of tasks and mapping of main threads are optimized simultaneously in IS. It is theoretically possible for IS to find better than (or at least equal to) the optimal solution of TSS because the solution space of IS completely covers that of TSS. However, it is difficult to find the good solutions of the IS method within a practical time due to the wider solution space.

Let $main\_map_{i,j}$ denote a 0-1 decision variable. $main\_map_{i,j}$ becomes 1 if the main-thread of task i is mapped to core j. Let $sub\_map_{i,j}$ denote a 0-1 decision variable. There is only a main-thread in each task. The constraint for each task is expressed as follows.

$$\forall i, \qquad \sum_j main\_map_{i,j} = 1 \tag{31}$$

$sub\_map_{i,j}$ becomes 1 if a sub-thread of task i is mapped to core j. Let Par_i be the degree of parallelism of task i. Par_i is assumed to be given. The mapping constraint for each task is expressed as follows. The total of threads is equal to the degree of parallelism. The constraint is formulated by the following formula.

$$\forall i, \qquad \sum_j \{main\_map_{i,j} + sub\_map_{i,j}\} = Par_i \tag{32}$$

The thread of task i mapped to core j is a main-thread or a sub-thread. The selection for threads is expressed as follows.

$$\forall i,\ j, \qquad main\_map_{ij} + sub\_map_{ij} < 2 \tag{33}$$

Let $Time_i$ denote the execution time of task i. $Time_i$ is assumed to be given. Let $start_i$ and $finish_i$ denote the start time and the finish time of task i, respectively. Let $C_{i1,i2}$ denote communication time between tasks $i1$ and $i2$. $C_{i1,i2}$ is assumed to be given. $C_{i1,i2}$ is 0 if there is no data dependency from tasks $i1$ to $i2$. Note that $start_i$ and $e_{i1,i2}$ are a decision variable and $finish_i$ is a dependent variable defined by the following equation.

$$\forall i, \qquad finish_i = start_i + Time_i \tag{34}$$

If two tasks, $i1$ and $i2$, is executed at the same time, the two tasks is mapped to the different cores. Otherwise, the execution of the two tasks cannot be overlapped in the same time. This resource constraint is formulated by the following formula.

$$\begin{aligned} &\forall i1,\ i2,\ i' \in \{i1, i2\},\ j, \\ &\sum_{i'} (main\_map_{i',j} + sub\_map_{i',j}) \leq 1 \\ &\vee \quad finish_{i1} \leq start_{i2} \\ &\vee \quad finish_{i2} \leq start_{i1} \end{aligned} \tag{35}$$

Let $e_{i1,i2}$ be a 0-1 decision variable, which is 0 if main-threads of two tasks, $i1$ and $i2$, are mapped to the same core. In other words, $e_{i1,i2}$ becomes 1 when communication between tasks $i1$ and $i2$ happens. $e_{i1,i2}$ is given by:

$$\forall i1,\ i2, \quad e_{i1,i2} = \begin{cases} 0 & \text{if } max_j\{main\_map_{i1,j} + main\_map_{i2,j}\} = 2 \\ 1 & \text{otherwise} \end{cases} \tag{36}$$

This work assumes a set of dependent tasks, and the tasks may

have a precedence dependency. Let $C_{i1,i2}$ denote communication time between tasks $i1$ and $i2$. $C_{i1,i2}$ is assumed to be given. $C_{i1,i2}$ is 0 if there is no data dependencies from tasks $i1$ to $i2$. $C_{i1,i2}$ is positive when there is the precedence constraint between two tasks. $C_{i1,i2}$ is positive when task $i1$ must be finished before task $i2$ starts. Otherwise, $C_{i1,i2}$ is 0. Then, the precedence constraint is expressed as follows.

$$\forall i1,\ i2, \qquad C_{i1,i2} > 0 \rightarrow finish_{i1} + C_{i1,i2} \cdot e_{i1,i2} \leq start_{i2} \tag{37}$$

This work aims at minimization of the overall schedule length. The objective function of our scheduling problem to be minimized is given as follows.

$$\max_i \{finish_i\} \tag{38}$$

Our scheduling problem with task communication time is now formally defined: Given $Time_i$, Par_i and $C_{i1,i2}$, find $main\_map_{i,j}$, $sub\_map_{i,j}$ and $start_i$ which minimize the objective function (38), subject to the seven constraints (31)–(37). Although formulas (35)–(38) are not in a linear form, they can be easily linearized by a simple transformation technique as well as Eqs. (15)–(25) and (29)–(30). There are $(5 \cdot Ntask^2 \cdot Ncore + 6 \cdot Ntask^2 + Ntask \cdot Ncore + 4 \cdot Ntask)$ constraints and $(2 \cdot Ntask^2 \cdot Ncore + 2 \cdot Ntask \cdot Ncore + 2 \cdot Ntask^2 + 2 \cdot Ntask + 1)$ variables in total.

5. Experiments

5.1 Experimental Setups

In order to test the effectiveness of this work, a set of experiments is conducted with fifteen task-graphs generated by TGFF [26]. Each task-graph contains up to 30 tasks. TGFF generates task graphs and resource parameters in accordance with the user's parameterized graph. In the experiments, we set the execution time parameter to 50 ± 20 and the edge cost parameter to 8 ± 5. In addition, we selected a task-graph of the realistic application from the Standard Task Graph (STG) set developed at Waseda University [27]. The task-graph contains 88 tasks. The task-graph is not taken account into communication cost. Therefore, we assume that the communication overhead between two tasks is 10% of the predecessor execution time.

Since no previous work is directly comparable to this work, we compared our two methods with a simple scheduling method as follows.

- *SS*: The sequential schedule when each task is assigned sequentially. **Figure 7** shows an example of SS result for the task graph shown in Fig. 1. Note that no communication is necessary with the SS method. The schedule length of the SS method is $\sum_i Time_i$.
- *EPCS*: The heuristic algorithm based on the PCS algorithm proposed in Ref. [10].
- *TSS*: ILP-based two-step scheduling approach presented in Section 3.
- *IS*: ILP-based integrate scheduling approach presented in Section 4.

EPCS is an extended version of the PCS algorithm proposed in Ref. [10]. The PCS algorithm is based on list scheduling. Since the original PCS algorithm does not take into account the communication overhead among tasks, we have extended the PCS algorithm for the communication overhead. When a task is scheduled, the main thread of the task is mapped to the core where the main thread of its predecessor task is mapped, if the core is available. **Figure 8** shows an example of EPCS. TSS and IS are ILP-based scheduling proposed in this paper. In order to solve the ILP problems, we used IBM ILOG CPLEX 12.8 on Intel Core i9-7980XE processors. Since ILP is very time-consuming, optimal solutions are not sometimes found in a practical time. In our experiments for TGFF and STG, CPU runtime of CPLEX is limited up to 2 hours and 24 hours, respectively, and the best solutions found at that time are used for evaluation.

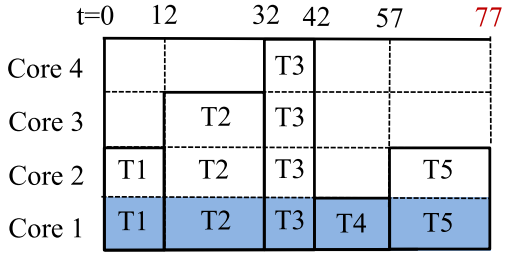

Fig. 7 Sequential scheduling method (SS).

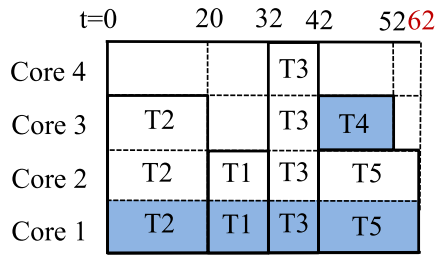

Fig. 8 Extended PCS method (EPCS).

5.2 Experimental Results

The scheduling results on two, four and eight cores are shown in **Figs. 9**, **10** and **11**, respectively. In the figures, a label on X-axis indicates ID of the task-graph and the number of tasks. Y-axis represents the schedule length normalized SS method.

On average, the TSS method achieves 6%, 12% and 14% reduction and the IS method achieves 8%, 13% and 15% reduction in schedule length, compared with the SS method. The TSS and IS methods outperformed the SS method. In the SS method, the communication overhead between tasks does not happen because the main-threads of tasks are mapped to same cores. However, our proposed methods take advantage of the task parallelism. Therefore, the TSS and IS methods can find better solutions than the SS method even if the communication happen.

The TSS method achieves 2%, 4% and 2% reduction, compared with the EPCS method. The IS method achieves 3%, 4% and 3% reduction. The EPCS method is based on list scheduling, which takes advantage of inter-task parallelism. Therefore, the EPCS method finds shorter schedule length than the SS method. However, due to the greedy nature of the EPCS algorithm, EPCS is not as good as the TSS and IS methods.

Let us compare the TSS and IS methods. On average, the IS method achieves 1%, 1% and 1% reduction in scheduling

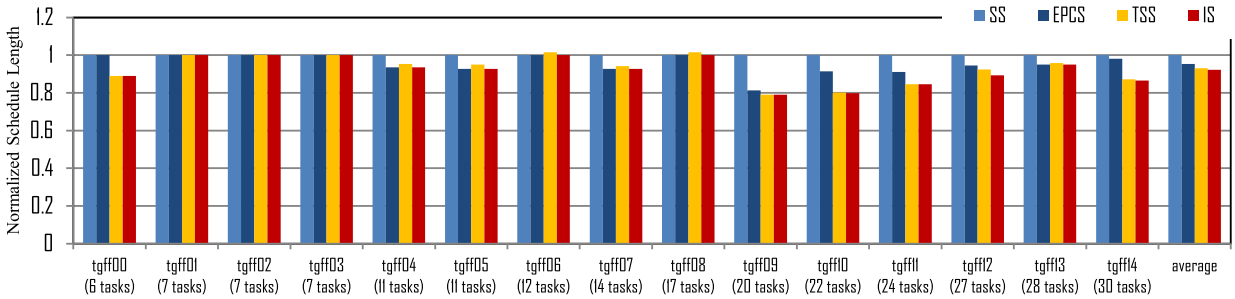

Fig. 9 Scheduling results on 2 cores.

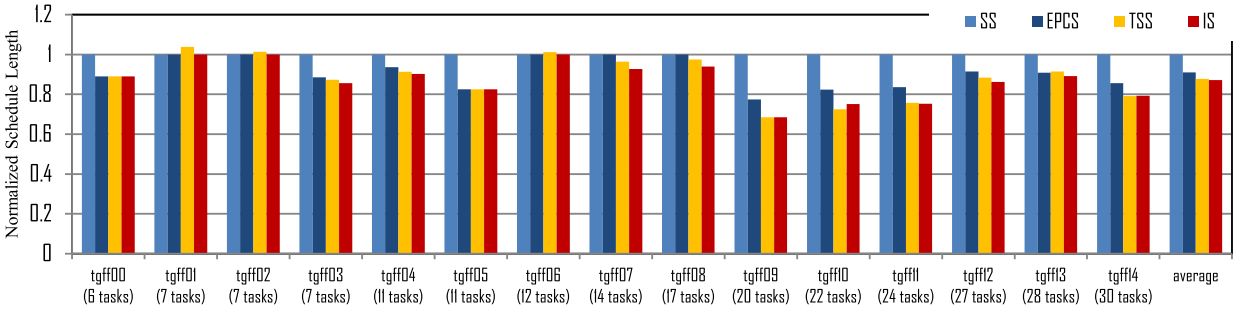

Fig. 10 Scheduling results on 4 cores.

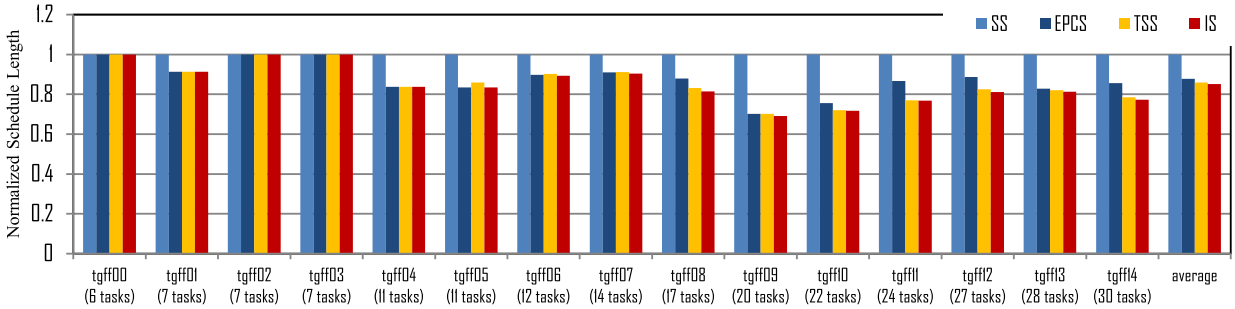

Fig. 11 Scheduling results on 8 cores.

length, compared with the TSS method. Scheduling of tasks and mapping of main threads are optimized simultaneously in the IS method. On the other hand, in the TSS method, each task is scheduled without the communication overhead and main-threads are selected in such a way that the overall communication overhead is minimized. Therefore, the solution space of the IS method completely covers that of the TSS, and the ILP solver found good solutions for the IS method. In this experiments, an improvement in schedule length of the IS method is observed in Figs. 9, 10 and 11. However, the TSS method is better than the IS method for some task-graphs (tgff10 and tgff14 on four cores). The solution space of the IS method is much larger than that of the TSS method. Therefore, it is sometimes difficult for the IS method to find better solutions than TSS in a limited time.

In the next experiments, we used a task-graph developed from the realistic application, robot control [27]. This task-graph contains 88 tasks and 131 edges. The scheduling results for task set of robot control are shown in **Fig. 12**. In the figures, the schedule lengths are also normalized to the SS method. A label on X-axis indicates the number of cores on which tasks are executed.

In cases of four and eight cores, IS failed to find any feasible solutions. In case of two cores, IS finds a solution, but the solution is worse than that of EPCS and TSS. The results show that the

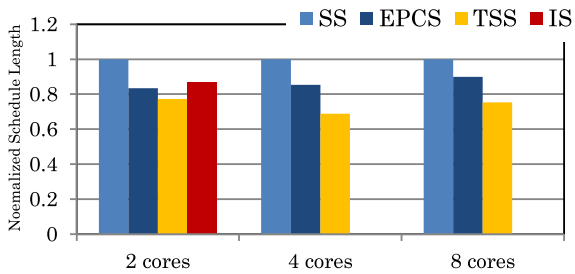

Fig. 12 Scheduling results for robot control.

IS method is not scalable to large task graphs. The TSS method finds shorter schedule length than the IS, EPCS and SS methods. The TSS method achieves 23%, 40% and 74% reduction, compared with the SS method. This experiment shows a significant improvement in scheduling results. The TSS method achieves 7% reduction on two cores, compared with the IS method. In addition, IS method is failed to find solutions on four and eight cores. It is possible for the IS method to find better than or at least equal to the optimal solution of the TSS method. However, it is difficult for ILP solver to find the optimal solution of the IS method within limited time because of the wider solution space. The task-graphs in the experiments are more than twice as large as the task-graphs from TGFF. Therefore, the TSS method outperformed the

Table 1 Runtime of ILP solver [seconds].

	2 cores				4 cores				8 cores			
	TSS			IS	TSS			IS	TSS			IS
	Step 1	Step 2	Total		Step 1	Step 2	Total		Step 1	Step 2	Total	
tgff00	6	<1	7	19	5	<1	6	9	12	<1	13	56
tgff01	6	<1	6	39	6	5	11	11	11	<1	12	26
tgff02	4	<1	4	7	5	<1	6	9	7	<1	8	78
tgff03	6	<1	7	10	10	4	14	10	11	<1	12	80
tgff04	9	<1	10	103	17	5	22	87	27	<1	28	76
tgff05	8	<1	9	11	13	<1	14	171	39	<1	40	304
tgff06	6	<1	7	156	11	<1	12	160	23	5	28	133
tgff07	15	<1	16	308	48	5	53	377	95	8	103	4,137
tgff08	23	<1	24	262	41	<1	42	1,817	44	<1	45	7,200<
tgff09	7,200<	<1	7,200<	7,200<	7,200<	5	7,200<	7,200<	7,200<	6	7,200<	7,200<
tgff10	7,200<	<1	7,200<	7,200<	7,200<	7	7,200<	7,200<	7,200<	5	7,200<	7,200<
tgff11	7,200<	<1	7,200<	7,200<	7,200<	8	7,200<	7,200<	7,200<	8	7,200<	7,200<
tgff12	7,200<	<1	7,200<	7,200<	7,200<	6	7,200<	7,200<	7,200<	8	7,200<	7,200<
tgff13	7,200<	<1	7,200<	7,200<	7,200<	9	7,200<	7,200<	7,200<	9	7,200<	7,200<
tgff14	7,200<	5	7,200<	7,200<	7,200<	6	7,200<	7,200<	7,200<	10	7,200<	7,200<
robot	86,400<	2	86,400<	86,400<	86,400<	6	86,400<	n.a.	86,400<	15	86,400<	n.a.

IS method in the case of task-graphs for robot control. The TSS method achieves 7%, 19% and 16% reduction on four and eight cores. The EPCS method is not as good as the TSS method due to the greedy nature of the EPCS algorithm. That is why the TSS method found shorter scheduling length than the EPCS method.

Our experimental results demonstrate the strength of methods scheduling tasks with communication time in three methods. The IS method finds better scheduling results than the TSS, EPCS and SS methods. However, it is difficult to find solutions quickly in the IS method. On the other hand, the TSS method outperforms other methods when the IS method suffers from a large size of task-graph.

Table 1 shows the CPU runtime of the ILP solver. The CPU runtime of PCS method is less than one second. As described before, we limited the CPU runtime up to two hours for the TGFF benchmarks and up to 24 hours for robot. For large task graphs, the IS method cannot find optimal schedules within the time limit, but still it finds the best schedules of the four methods in most cases. For the TSS method, the first step requires the long runtime, while the second step is fast. It should be noted that existing heuristic algorithms such as Refs. [10], [11], [12] can be used for the first step, in order to accelerate the TSS method.

6. Conclusions

This paper addressed a scheduling problem for data-parallel tasks on multiple cores with communication time. This work presented two solution methods, the TSS and IS methods, for the scheduling problem based on integer linear programming formulation. The TSS method is a two-step approach to scheduling and thread assignment. The IS method is an integrated approach to scheduling and thread assignment. In the experiments, The IS method finds better scheduling results than the other methods. However, it may be difficult to find solutions in the IS method depending on task-graphs. In the case of that, the TSS method finds shorter schedule length than the other methods in practical time. In future, we plan to extend this work to take into account dynamic behaviors of tasks and architectures.

Acknowledgments This work is in part supported by KAKENHI 15H02680.

References

[1] Hironori, K. and Seinosuke, N.: Practical multiprocessor scheduling algorithms for efficient parallel processing, *IEEE Trans. Computers*, Vol.C-33, pp.1023–1029 (1984).

[2] Satoshi, F.: A branch-and-bound algorithm for solving the multiprocessor scheduling problem with improved lower bounding techniques, *IEEE Trans. Computers*, Vol.60, pp.1006–1016 (2010).

[3] Shahul, A.Z.S. and Sinnen, O.: Optimal scheduling of task graphs on parallel systems, *Proc. 9th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp.323–328 (2008).

[4] Kwok, Y.K. and Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Computing Surveys*, Vol.31, pp.406–471 (1999).

[5] Hwang, J.J., Chow, Y.C., Anger, F.D. and Lee, C.Y.: Scheduling precedence graph in systems with interprocessor communication times, *SIAM Journal of Computing*, Vol.18, pp.244–257 (1989).

[6] Sih, G.C. and Lee, E.A.: Compile time scheduling heuristic for interconnection constrained heterogeneous processor architecture, *IEEE Trans. Parallel and Distributed System*, Vol.4, pp.175–187 (1993).

[7] Hagras, T. and Janecek, J.: A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environment, *Proc. 2003 International Conference on Parallel Processing Workshops*, pp.149–155 (2003).

[8] Palmer, A. and Sinnen, O.: Scheduling algorithm based on force directed clustering, *Proc. 9th International Conference Parallel and Distributed Computing, Applications and Technologies*, pp.311–318 (2008).

[9] Drozdowski, M.: Scheduling multiprocessor tasks: An overview, *European Journal of Operational Research*, Vol.94 (1996).

[10] Liu, Y., Meng, L., Taniguchi, I. and Tomiyama, H.: Novel list scheduling strategies for data parallelism task graphs, *International Journal on Networking and Computing*, Vol.4, No.2, pp.279–290 (July 2014).

[11] Liu, Y., Meng, L., Taniguchi, I. and Tomiyama, H.: A dual-mode scheduling approach for task graphs with data parallelism, *International Journal of Embedded Systems*, Vol.9, No.2, pp.147–156 (2017).

[12] Liu, Y., Meng, L., Taniguchi, I. and Tomiyama, H.: A genetic algorithm for scheduling of data-parallel tasks on multi cores archtectures, *IPSJ Trans. System LSI Design Methodology*, Vol.12 (Aug. 2019).

[13] Turek, J., Wolf, J.L. and Yu, P.S.: Approximate algorithms scheduling parallelizable tasks, *Annual ACM Symposium on Parallel Algorithms and Architectures* (1992).

[14] Yang, H. and Ha, S.: ILP based data parallel multi-task mapping/scheduling technique for MPSoC, *International SoC Design Conference* (2008).

[15] Shimada, K., Kitano, S., Taniguchi, I. and Tomiyama, H.: ILP-based scheduling for parallelizable tasks, *IEICE Trans. Fundamentals*, Vol.E100-A, No.7, pp.1503–1505 (2017).

[16] Shimada, K., Kitano, S., Taniguchi, I. and Tomiyama, H.: ILP-based scheduling for malleable fork-join tasks, to appear in *ACM SIGBED Review* (2019).

[17] Chen, C. and Chu, C.: A 3.42-Approximation algorithm for scheduling malleable tasks under precedence constraints, *IEEE Trans. Parallel and Distributed Systems*, Vol.24, No.8, pp.1479–1488 (2013).

[18] Roel, J., Firew, S., Sander, S., Akash, K. and Henk, C.: An automated flow to map throughput constrained applications to a MPSoC, *Bringing Theory to Practice: Predictability and Performance in Embedded Systems* (2011).

[19] Hwang, J.J., Chow, Y.C., Anger, F.D. and Lee, C.Y.: Scheduling precedence graphs in systems with interprocessor communication times, *SIAM Journal of Computing*, Vol.18, No.2, pp.244–257 (1989).

[20] Shahul, A.Z.S. and Sinnen, O.: Task graphs optimally with A*, *The Journal of Supercomputing*, Vol.51, No.3, pp.310–332 (2010).

[21] Veltman, B., Lageweg, B.J. and Lenstra, J.K.: Multiprocessor scheduling with communication delays, *Parallel Computing*, Vol.16, No.2-3, pp.173–182 (1990).

[22] Kai, M.: Task scheduling algorithm to give a reasonable number of processors taking account of communication overhead, *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Vol.2, pp.950–955 (1997).

[23] Fujita, S. and Nakagawa, T.: Lower bounding techniques for the multiprocessor scheduling problem with communication delay, *Proc. International Conference on Parallel Architectures and Compilation Techniques*, pp.212–220 (1999).

[24] Selvakumar, S. and Murthy, C.S.R.: Scheduling precedence constrained task graphs with non-negligible intertask communication onto multiprocessors, *IEEE Trans. Parallel and Distributed Systems*, Vol.5, pp.328–336 (1994).

[25] Kuang, S.R., Chen, C.Y. and Liao, R.Z.: Partitioning and pipelined scheduling of embedded system using integer linear programming, *Proc. International Conference on Parallel and Distributed Systems*, Vol.2, pp.37–41 (1997).

[26] Dick, R.P., Rhodes, D.L. and Wolf, W.H.: TGFF: Task graph for free, *International Workshop on Hardwere/Software Codesign* (1998).

[27] Tobita, T. and Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *Journal of Scheduling*, Vol.5, No.5, pp.379–394 (2002).

Kana Shimada received her B.E. and M.E degrees in Electronic and Computer Engineering from Risumeikan University in 2017 and 2019. She currently works for Sony Imaging Products & Solutions Inc. respectively. Her research interests include, but not limited to, task scheduling problem for multicore and integer linear programming for embedded systems. She is a member of IEEE.

Ittetsu Taniguchi received B.E., M.E., and Ph.D degrees from Osaka University in 2004, 2006, and 2009, respectively. From 2007 to 2008, he was an international scholar at Katholieke Universiteit Leuven (IMEC), Belgium. In 2009, he joined the College of Science and Engineering, Ritsumeikan University as an assistant professor, and became a lecturer in 2014. In 2017, he joined the Graduate School of Information Science and Technology, Osaka University as an associate professor. His research interests include system level design methodology, design methodologies for cyber-physical systems, etc. He is a member of IEEE, ACM, IEICE, and IPSJ.

Hiroyuki Tomiyama received his B.E., M.E. and D.E. degrees in computer science from Kyushu University in 1994, 1996 and 1999, respectively. He worked as a visiting researcher at UC Irvine, as a researcher at ISIT/Kyushu, and as an associate professor at Nagoya University. Since 2010, he has been a full professor with College of Science and Engineering, Ritsumeikan University. He has served on program and organizing committees for a number of premier conferences including DAC, ICCAD, DATE, ASP-DAC, CODES+ISSS, CASES, ISLPED, RTCSA, FPL and MPSoC. He has also served as editor-in-chief for IPSJ TSLDM, as an associate editor for ACM TODAES, IEEE ESL and Springer DAEM, and as chair for IEEE CS Kansai Chapter and IEEE CEDA Japan Chapter. His research interests include, but not limited to, design methodologies for embedded and cyber-physical systems. He is a member of ACM, IEEE, IEICE and IPSJ.

(Recommended by Associate Editor: *Kenshu Seto*)