Research Article

Multistep Scheduling Algorithm for Parallel and Distributed Processing in Heterogeneous Systems with Communication Costs

Hitoshi Yamazaki,¹ Katsumi Konishi,² Seiichi Shin,¹ and Kenji Sawada¹

¹ Department of Mechanical Engineering and Intelligent Systems, The University of Electro-Communications, 1-5-1 Chofugaoka, Chofu-shi, Tokyo 182-8585, Japan

² Department of Computer Science, Kogakuin University, 1-24-2 Nishishinjuku, Shinjuku-ku, Tokyo 163-8677, Japan

Correspondence should be addressed to Hitoshi Yamazaki; hitoshi-yamazaki@uec.ac.jp

Received 16 August 2013; Revised 22 October 2013; Accepted 23 October 2013

Academic Editor: Chin-Chia Wu

Copyright © 2013 Hitoshi Yamazaki et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We discuss a task scheduling problem in heterogeneous systems and propose a multistep scheduling algorithm to solve it. Existing scheduling algorithms formulated as 0-1 integer linear programming can be used to consider the optimality of task scheduling. However, they cannot address complicated relations among tasks or communication costs among processors. Therefore, we propose a scheduling algorithm that formulates communication costs as 0-1 integer linear programming. On the other hand, 0-1 integer linear programming takes a long time to calculate the scheduling results because it is NP-complete. Thus, scheduling time also needs to be decreased. A solution for decreasing scheduling time is a graph clustering which decomposes a large task graph into smaller subtask graphs (clusters). Also, it is important for parallel and distributed processing to find task parallelism in a task graph. Then, we also propose a *clustering algorithm based on SCAN*. SCAN is an algorithm for finding clusters in a network. The *clustering algorithm based on SCAN* can find task parallelism in a task graph. In numerical examples, we argue the following two points. First, our multistep scheduling algorithm resolves the scheduling problem in heterogeneous systems. Second, it is superior to the existing scheduling algorithms in terms of calculation time.

1. Introduction

Computer simulation is useful for various calculations, for example, for developing medicine [1], analyzing the orbit of rockets [2], and developing various other products [3]. Computer simulation can reduce the time and costs required to make prototypes and tell us in advance the amount of deterioration and risk in what we do. Hence, developing computers for conducting simulations is important. Today, a single processor, which is the main calculating element in a computer, has performance limitations. Accordingly, research and development on a multicore CPU and GPU accumulated processors as cores [4] and grid computing [5] to reduce computation time has been extensive. This research is in the field of parallel and distributed processing. It is important for increasing the computation speed for assigning tasks

to appropriate processors or computers. This is called task scheduling [6].

In parallel and distributed processing, a program is composed of a set of tasks, which is regarded as a graph in which tasks are bridged. The graph is called a task graph [7]. One of the most active research areas is on the task scheduling problem, which is how to assign tasks to processing elements (PEs), for example, CPU, GPU, and DSP, appropriately so that certain performance indices are optimized. There are many algorithms for task scheduling [6–10]. Focusing on the optimality of task scheduling, Beaumont et al. [9] and Shioda et al. [10] have proposed scheduling algorithms within 0-1 integer linear programming framework. Beaumont et al.'s algorithm [9] is applied to task graphs for multiround algorithms. On the other hand, Shioda et al.'s algorithm [10] is used to execute more complicated task graphs that have tasks

FIGURE 1: (a) Task graph (50 tasks), (b) results of *clustering algorithm based on SCAN* ($\epsilon = 0.3$, $\mu = 2$).

with priority orders to be executed by PEs compared with Beaumont et al.'s [9]. In addition, it has been reported through numerical experiments that Shioda et al.'s algorithm can solve the scheduling problem more optimally compared with Critical Path/Most Immediate Successors First (CP/MISF) [11]. CP/MISF is one of the most efficient heuristic scheduling algorithms and can be applied to a task graph which has tasks with priority orders to be executed. However, Shioda et al.'s algorithm [10] cannot be used to take into account communication costs among PEs. Consider multicore CPUs and GPUs composed of multiple cores that have the same functions as PEs. In parallel and distributed processing, communication among PEs generally occurs when data are transferred from one PE to another.

Motivated by the above, we propose a multistep scheduling algorithm with considering communication costs for PEs. The proposed algorithm is also based on 0-1 integer linear programming similar to Shioda et al.'s [10]. However, 0-1 integer linear programming is NP-complete [12] and there is a problem such that scheduling time increases exponentially with increasing task-graph size. A solution for decreasing scheduling time is graph clustering [13] which decomposes a large task graph into smaller subtask graphs (clusters). Also, it is important for parallel and distributed processing to find task parallelism in a task graph. For task parallelism, we use a structural clustering algorithm for networks (SCAN) [14], which finds clusters in a network. SCAN is for networks not for task graphs. Therefore, we also propose a *clustering algorithm based on SCAN*, which is modified for task graphs. This algorithm can find task parallelism.

Our proposed multistep scheduling algorithm consists of the following three steps. The first is the clustering step for a task graph that uses our *clustering algorithm based on SCAN*. The second is the task scheduling step, which involves assigning tasks of each cluster to cores in a multicore CPU or GPU.

FIGURE 2: Sample of *clusters scheduling results*.

Shioda et al.'s algorithm [10] is used for task scheduling. The third is the *clusters scheduling* step, which involves assigning clusters to PEs by taking into account communication costs among PEs.

We argue that the proposed multistep scheduling algorithm is efficient in various numerical experiment environments. In Section 4, we discuss the following two special numerical experiments to examine the effectiveness of our multistep scheduling algorithm. The first such environment is of a homogeneous distributed system in which the following condition is assumed. A computer consisting of multicore CPUs shared memories connected to cores in a multicore CPU and a main memory connected to all multicore CPUs. Communication costs are considered only when data are transferred from one multicore CPU to another. The second environment is of a heterogeneous distributed system in which the following condition is assumed. A computer consisting of multicore CPUs, GPUs shared memories connected to cores in a multicore CPU or GPU and a main memory connected to all PEs. Communication costs are considered only when data are transferred to a PE from another one.

FIGURE 3: (a) Task graph (24 tasks), (b) results of applying original SCAN ($\epsilon = 0.3, \mu = 2$), and (c) results of applying clustering algorithm based on SCAN ($\epsilon = 0.3, \mu = 2$).

FIGURE 4: Series of tasks in task graph.

This paper is organized as follows. In Section 2, we describe the problem defined in this paper. In Section 3, we introduce two algorithms; an optimal scheduling algorithm [10] and SCAN [14] for use with our algorithms. In Section 4, we introduce our proposed algorithms and experimental results to show the efficiency of the proposed algorithms. In Section 5, we present the results of numerical experiments and compare the proposed multistep scheduling algorithm and Shioda et al.'s algorithm [10]. We give the conclusion in Section 6.

FIGURE 5: Order of clusters c' .

2. Problem Definition

We address the problem of assigning computing tasks to multiple PEs. The proposed multistep scheduling algorithm is used for static scheduling [7]. We consider parallel programs described by a task graph consisting of a directed acyclic graph (DAG), where vertices represent computing tasks and edges represent data dependencies between two tasks. Each task describes a sequence of instructions to be computed and each data dependence describes a communication of data between two tasks. We assume that communication cost among cores is negligible compared with the communication cost among PEs.

FIGURE 6: Results of *clusters scheduling* when (a) $w = 0$, (b) $w = 1$, (c) $w = 2$, (d) $w = 3$, (e) $w = 4$, and (f) $w = 5$.

We assume the following conditions to make it simple to examine the effectiveness of the proposed multistep scheduling algorithm.

- (i) In a homogeneous distributed system, a computer consisting of multicore CPUs shared memories connected to cores in a multicore CPU and a main memory connected to all multicore CPUs.
- (ii) In a heterogeneous distributed system, a computer consisting of a multicore CPU multiple GPUs, shared memories connected to cores in a multicore CPU or GPU and a main memory connected to all PEs (a multicore CPU and multiple GPUs).
- (iii) There are tasks that cannot be executed until other tasks have been executed (we call this a task priority problem).
- (iv) There is a task graph with priority orders among tasks to be executed, as shown in Figure 1(a).
- (v) Communication costs only occur among PEs.

3. Existing Algorithms

3.1. Optimal Task Scheduling Algorithm. The algorithm developed by Shioda et al. [10] is based on 0-1 integer linear programming. It is used to take into account a task priority problem and is superior to CP/MISF in terms of optimization focusing on makespan, that is, the total execution time. The setting of numerical experiments conducted by Shioda et al. [10] is for parallel and distributed processing with the task priority problem. The algorithm has the following two issues. First, it cannot be used to take into account communication costs among PEs. Second, it takes a long time to calculate task

scheduling because 0-1 integer linear programming is NP-complete.

3.2. Structural Clustering Algorithm for Networks (SCAN). SCAN [14] is a network clustering algorithm that detects clusters, hubs, and outliers in networks. It is used to take into account the structure of vertexes, which are elements of a network. Vertexes are connected by edges. SCAN involves the following steps. At the beginning, the structure of vertexes is described by its neighborhood. At the second step, structural similarity, which is the number of common neighbors by the geometric mean of the two neighborhoods' sizes, is calculated. In the third step, threshold ϵ is applied to the computed structural similarity when assigning cluster membership. In the fifth step, when a vertex shares structural similarity with more than μ vertexes, the vertex becomes a *core vertex*, which is a nucleus or seed for a cluster. Parameters ϵ and μ are used to determine clustering for networks. In this paper, a vertex in SCAN is regarded as a task.

4. Proposed Algorithms

In this section, we propose a multistep scheduling algorithm for parallel and distributed processing with communication costs under the conditions presented in Section 2. Our multistep scheduling algorithm involves the following three steps.

The first step is the clustering step for a task graph. We find clusters in a task graph by using the proposed *clustering algorithm based on SCAN*. Figure 1(b) shows the results of clustering for the task graph shown in Figure 1(a).

The second step is task scheduling. We obtain results on how to assign tasks of each cluster to cores in a PE. Shioda et al.'s algorithm [10] is used for this step.

FIGURE 7: Results of *clusters scheduling* when (a) $w = 0$, (b) $w = 0.1$, (c) $w = 0.2$, (d) $w = 0.3$, (e) $w = 0.4$, and (f) $w = 0.5$.

The third step is *clusters scheduling*. We obtain results on how to assign clusters to PEs with considering communication costs among the PEs. In this paper, the communication cost occurs when a cluster is executed by a PE and then the processing result of cluster is transmitted to another PE through main memory. In Figure 2, communication cost occurs between clusters 2 and 5.

Step 1 (clustering algorithm based on SCAN). SCAN [14] is used to take into account a nondirected graph as a network. A task graph is represented by a DAG then an adjacency matrix because a task priority problem is addressed. Then, it is natural to apply a DAG and an adjacency matrix to SCAN. However, the original SCAN cannot find task parallelism sufficiently since it is for networks not for task graphs. Hence, we add the following two steps to SCAN.

- (i) Before applying SCAN to a task graph, a series of tasks is regarded as one task (this is called *preprocessing in SCAN*).
- (ii) After applying SCAN to a task graph, the resulting isolation task that does not belong to any clusters is included in a cluster that has more than two edges

connected to the isolation task (this is called *post-processing in SCAN*).

Then, our *clustering algorithm based on SCAN* consists of *preprocessing in SCAN*, *SCAN*, and *post-processing in SCAN*. Our *clustering algorithm based on SCAN* is efficient for a task graph that has high task parallelism. Next, we discuss the effectiveness of *preprocessing* and *post-processing in SCAN*. First, we obtain the results shown in Figure 3(b) when the task graph shown in Figure 3(a) is applied to the original SCAN. Figure 3(b) shows that the original SCAN cannot find task parallelism efficiently in a task graph because it involves the following processes of finding clusters for a task graph.

- (i) We apply a threshold ε to the computed structural similarity when assigning cluster membership.
- (ii) When a vertex (task) shares structural similarity with more than μ tasks, it becomes a core task.
- (iii) A cluster consisting of core task v and tasks connected to v is found.

We then add *preprocessing* and *post-processing in SCAN* to the original SCAN. *Preprocessing in SCAN* is for regarding a series of tasks as one task, as shown in Figure 4, before the original SCAN is applied to a task graph.

FIGURE 8: Task graph (240 tasks).

Next, we explain a numerical experiment by using our *clustering algorithm based on SCAN*. We consider the two task graphs that have efficient task parallelism in Figures 1(a) and 3(a).

Figures 1(b) and 3(c) show the results which our *clustering algorithm based on SCAN* can find task parallelism of a task graph.

Step 2 (task scheduling). We obtain results on how to assign tasks of each cluster to cores in a PE. Shioda et al.'s algorithm [10] is used for this step. Table 1 shows the result of task scheduling for clusters in Figure 1(b).

Step 3 (clusters scheduling). In this part, we present *clusters scheduling* algorithm which takes into account communication costs in distributed processing. Communication costs occur among PEs when a PE transfers a data cluster to another PE. Therefore, we use (1)–(16) for *clusters scheduling*

TABLE 1: Makespan for each cluster shown in Figure 1(b).

	Cluster number					
	1	2	3	4	5	6
Makespan	4	3	3	4	3	4

with communication costs. Equations (2)–(11) are almost the same as those of Shioda et al.'s algorithm [10] and the other equations are new. Equation (1) is an objective function, and (2)–(16) are constraint conditions. In (1), λ is a weight parameter between the first and second parts of (1). The constants are defined as follows. The term N_{pe} denotes the number of PEs. Each PE is expressed as PE 1, PE 2, ..., PE i . The term $N_{cluster}$ denotes the number of clusters. Each cluster is expressed as clusters 1, 2, ..., cluster c . The term N_{step} denotes the number of time steps. Each step is expressed as steps 1, 2, ..., step k . The terms i , c , and k are subscripts for PEs, clusters, and steps, respectively. *Clusters scheduling* requires

FIGURE 9: Results of clustering algorithm based on SCAN ($\epsilon = 0.1$, $\mu = 3$).

results of the task scheduling for each cluster, as shown in Table 1. Table 1 lists the results of task scheduling for the clusters in Figure 1(b)

$$\text{Minimize } m + \lambda \left(s + \sum_{c \in C_{\text{start}}} w_c + b + \sum_{i=1}^{N_{\text{pe}}} \text{ST}_i \right) \quad (1)$$

$$\text{subject to } x_{ick} = \{0, 1\} \quad \forall i, \forall c, \forall k \quad (2)$$

$$y_{ik} = \{0, 1\} \quad \forall i, \forall k \quad (3)$$

$$\sum_{k=1}^{k'-1} y_{ik} \geq (k' - 1) y_{ik'} \quad (4)$$

$$\forall i, \forall k' \quad (k' \geq 2)$$

$$y_{ik} \geq x_{ick} \quad \forall i, \forall c, \forall k \quad (5)$$

$$\sum_{k=1}^{N_{\text{step}}} \sum_{i=1}^{N_{\text{pe}}} x_{ick} = 1 \quad \forall c \quad (6)$$

$$\sum_{c=1}^{N_{\text{cluster}}} x_{ick} \leq 1 \quad \forall i, \forall k \quad (7)$$

$$\sum_{i=1}^{N_{\text{pe}}} \sum_{c=1}^{N_{\text{cluster}}} \sum_{k=1}^{k'-1} x_{ick} \geq p \sum_{i=1}^{N_{\text{pe}}} x_{ic'k} \quad (8)$$

$$\forall k' \quad (k' \geq 2)$$

$$0 = \sum_{i=1}^{N_{\text{pe}}} x_{ic'k} \quad (k = 1) \quad (9)$$

FIGURE 10: Results of *clusters scheduling* when (a) $w = 0$, (b) $w = 1$, and (c) $w = 14$.

$$\sum_{c=1}^{N_{\text{cluster}}} \sum_{k=1}^{N_{\text{step}}} t_{ic} x_{ick} \leq m \quad \forall i \quad (10)$$

$$\sum_{k=1}^{N_{\text{step}}} f_{ik} y_{ik} \leq s \quad \forall i \quad (11)$$

$$z_{ick} \in \{0, 1\} \quad \forall i, \forall c, \forall k \quad (12)$$

$$z_{ick} \geq x_{ick} \quad \forall i, \forall c, \forall k \quad (13)$$

$$\sum_{c'=c+1}^{N_{\text{cluster}}} \sum_{k=1}^{N_{\text{step}}} w d_{cc'} e_{ii'} (z_{ick} + z_{i'c'k'}) - w N_{\text{step}} \leq w_c \quad (14)$$

$$\forall i, \forall i', \forall k', \forall c' \in C_{\text{start}}$$

$$\text{if } (d_{cc'} = 1) \text{ then } \sum_{i=1}^{N_{\text{pe}}} \sum_{k=1}^{N_{\text{step}}} k x_{ic'k} - \sum_{i=1}^{N_{\text{pe}}} \sum_{k=1}^{N_{\text{step}}} k x_{ick} \leq b \quad (15)$$

$$\forall c' \in C$$

$$\sum_{c=1}^{N_{\text{cluster}}} \sum_{k=1}^{N_{\text{step}}} k x_{ick} \leq ST_i \quad \forall i. \quad (16)$$

Equations (1)–(11) show the scheduling algorithm by considering execution costs of clusters. The term x_{ick} is a binary variable. When cluster c is executed by PE i at step k , x_{ick} equals 1; otherwise, x_{ick} equals 0. The term y_{ik} is a binary variable. When PE i is able to execute a cluster at step k , y_{ik} equals 1; otherwise, y_{ik} equals 0. The term m denotes *clusters-makespan*, that is, the total execution time in *clusters scheduling*. The term s is defined as the maximum number of executed clusters among PEs. The term f_{ik} denotes the number of clusters that is executed by PE i at step k ; f_{ik} continually

TABLE 2: (a) An example of t_{ic} in a homogeneous system with two PEs. (b) An example of t_{ic} in a heterogeneous system with two PEs.

(a)						
t_{ic}	c					
	1	2	3	4	5	6
i						
1	4	3	3	4	3	4
2	4	3	3	4	3	4

(b)						
t_{ic}	c					
	1	2	3	4	5	6
i						
1	4	3	3	4	3	4
2	8	6	6	8	6	8

equals 1. The term t_{ic} is the time when PE i executes cluster c . As shown in Table 2, we obtain t_{ic} from the results of task scheduling, which is presented in Table 1. Using t_{ic} , this paper distinguishes between homogeneous systems and heterogeneous systems. Table 2(a) shows an example of t_{ic} in a homogeneous system with two PEs. In homogeneous systems, t_{ic} equals $t_{i'c}$ ($\forall i \neq i'$). Table 2(b) presents an example of t_{ic} in a heterogeneous system with two PEs. In heterogeneous systems, t_{ic} does not necessarily equals $t_{i'c}$ ($\forall i \neq i'$). In Shioda et al.'s algorithm [10], t_{ic} continually equal f_{ik} . In this paper, t_{ic} does not always equal f_{ik} .

Equation (4) enables PEs to execute clusters in series. This leads to the minimization of *clusters-makespan* and processing load, which is the time to execute clusters of each PE i . When y_{ik} satisfies (4) and (5), $y_{ik} = 0$ means that execution of PE i is finished at step k . Such conditions enable us to minimize *clusters-makespan*. Equation (6) means that every cluster is certainly executed by any PE i . Equation (7) means that every PE i is able to execute less than or equal to one cluster in a step. Similar to tasks, there are clusters that cannot be executed until other clusters are executed. We call this the cluster priority problem. Equations (8) and (9) address this problem, as shown in Figure 5. If the number of clusters that have been already executed is less than P (by using (8) and (9)), cluster c' will not be executed at step k .

FIGURE 11: Results of *clusters scheduling* when (a) $w = 0$, (b) $w = 1$, and (c) $w = 14$.TABLE 3: $d_{cc'}$ for Figure 1(b).

$d_{cc'}$	c'					
	1	2	3	4	5	6
c						
1	0	0	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

TABLE 4: $e_{ii'}$ for two PEs.

$e_{ii'}$	i'	
	1	2
i		
1	0	1
2	1	0

TABLE 5: Experimental preconditions.

Multicore CPU i	Cluster c	Step k	Communication costs w
1, 2	1, 2, ..., 6	1, 2, ..., 6	0, 1, 2, 3, 4, 5

Mathematical formulas of minimizing communication costs among PEs are given as (12), (13), and (14).

The term w denotes the communication cost among PEs, w_c denotes the communication cost that occurs when a data cluster c is executed by a PE i and then the resultant data is transmitted to another PE i' through main memory, and C_{start} is the set of clusters that has at least one cluster. In Figure 1(b), C_{start} consists of clusters 1, 2, and 3. A cluster that belongs to C_{start} is called a parent cluster. Also, a cluster which a cluster of C_{start} has is called a child cluster. When a PE i can execute cluster c at step k , z_{ick} equals 1; otherwise, z_{ick} equals 0. If cluster c has child cluster c' , $d_{cc'}$ equals 1; otherwise, $d_{cc'}$ equals 0. For Figure 1(b), $d_{cc'}$ is shown in Table 3. When PE i is different from PE i' , $e_{ii'}$ equals 1; otherwise, $e_{ii'}$ equals 0. Table 4 lists $e_{ii'}$ when there are two PEs.

By (6) and (13), there are PE i and step k , such that z_{ick} equals 1 for each cluster c . The left first part of (14) means that w occurs between PE i and PE i' when $d_{cc'}$ and $e_{ii'}$ equal 1. $d_{cc'} = 1$ and $e_{ii'} = 1$ mean the following case. Two clusters that have order to be executed are dealt by different PEs. When z_{ick} and $z_{i'c'k'}$ equal 1, it takes $w(z_{ick} + z_{i'c'k'})$ as communication costs. In this case, the costs are increased by $2wN_{\text{step}}$. Then, there is the left second part of (14) which is minus the costs by wN_{step} .

Equation (15) is a subjective function for reducing the sum of steps (execution time interval) among clusters with priority orders to be executed, where b means the number of steps among clusters with priority orders to be executed and C is a set of all clusters. The first left part of (15) means a sum of steps, at which child clusters are executed by any PE i . The second left part of (15) means the sum of steps at which any PE i executes parent clusters. Equation (16) is a subjective function for minimization the sum of steps at which PE i executes clusters, where ST_i is defined as the sum of steps at which PE i executes clusters. From (10), (11), (14), (15), and (16), (1) means minimization of *clusters-makespan*, the number of clusters which are executed by one PE i , w among PEs, sum of steps among clusters with priority orders to be executed, and sum of steps at which each PE i executes clusters.

4.1. Homogeneous Distributed System. In this section, we explain an experiment of *clusters scheduling* in a homogeneous distributed system by using (1)–(16). We consider clusters in Figure 1(b) and a computer having two identical multicore CPUs. Table 5 lists the preconditions of the experiment. We do *clusters scheduling* under these preconditions.

FIGURE 12: (a) Task graph (50 tasks), (b) results of clustering algorithm based on SCAN ($\varepsilon = 0.1, \mu = 3$).

TABLE 6: Experimental preconditions.

Multicore CPU i	Cluster c	Step k	Communication costs w	Parameter λ
1, 2, 3	1, 2, ..., 6	1, 2, ..., 6	0, 0.1, 0.2, 0.3, 0.4, 0.5	1

Figure 6 shows the following three points. The first point is that *clusters scheduling* algorithm can be used to take into account w among multicore CPUs. In other words, the number of communications among them is decreased by setting larger w . The second point is that *clusters scheduling* algorithm can use main memory. This means that it is able to reduce the sum of execution time interval. In other words, occupation time of main memory is decreased. The third point is that *clusters scheduling* algorithm can minimize the latest stating time of cluster execution.

Next, we assumed the following conditions. There are clusters shown in Figure 1(b) and a computer having three identical multicore CPUs. Table 6 lists the preconditions of this experiment.

Figure 7 shows the following thing. Increasing w decreases the number of communication among multicore CPUs similar to the result of two multicore CPUs (PEs). It is expected that our *clusters scheduling* can adapt to increase the number of multicore CPUs (PEs).

4.2. Heterogeneous Distributed System. In this section, we explain an experiment of *clusters scheduling* in a heterogeneous distributed system by using (1)–(16). We assumed a task

TABLE 7: Experimental preconditions.

Multicore CPU i	GPU i	Cluster c	Step k	Communication costs w
1	2, 3, 4	1, 2, ..., 7	1, 2, ..., 7	0, 1, 14

graph as shown in Figure 8 and a computer having a multicore CPU and three identical GPUs.

Figure 9 shows the results of our *clustering algorithm based on SCAN* for the task graph shown in Figure 8, and Table 7 lists the preconditions of the experiment. A multicore CPU has two identical cores. A GPU has eight identical cores. The executional ability, of a core in a multicore CPU, to execute tasks is two times that of a core in a GPU. A core in a GPU takes one time to execute a task. Therefore, we obtain Table 8, which represents the makespan (equivalently t_{ic}) in a multicore CPU and a GPU for the clusters shown in Figure 9. We do *clusters scheduling* under this condition and obtain the result as shown in Figure 10.

We see the three features from Figure 10. The first point is that *clusters scheduling* algorithm can be used to take into account w which is communication cost among a multicore CPU and GPUs. In other words, the number of communications among them is decreased by setting larger w . The second point is that *clusters scheduling* algorithm can reduce the sum of execution time interval. This means that occupation time of main memory is decreased. The third point is that *clusters scheduling* algorithm can minimize the latest stating time of cluster execution.

FIGURE 13: (a) Task graph (70 tasks), (b) results of clustering algorithm based on SCAN ($\varepsilon = 0.1$, $\mu = 4$).

TABLE 8: Makespan in multicore CPU and GPU for clusters shown in Figure 9.

t_{ic}	Cluster number						
	1	2	3	4	5	6	7
Multicore CPU ($i = 1$)	17	5	3	5	17	7	9
GPU ($i = 2, 3, 4$)	10	4	3	4	10	5	6

Clusters scheduling gives us not makespan but *clusters-makespan*. t_{ic} differs from cluster to cluster and from PE to PE. Although this situation causes idle time of PE, *clusters-makespan* does not include the idle time. Then, we get makespan from the result of *clusters scheduling*. From Figure 10, we obtain the scheduling results shown in Figures 11(a), 11(b), and 11(c).

Figure 11 shows that makespan is increased by setting larger w . Also, Figure 11 shows that all PEs are used when w equals 0, and some PEs are used when w is more than 0. Therefore, *clusters scheduling* algorithm can be used to take

TABLE 9: Experimental preconditions.

Task graph	PE i	Task j	Time k
Figure 1(a)	1, 2, 3, 4	1, 2, ..., 50	1, 2, ..., 50
Figure 3(a)	1, 2, 3, 4	1, 2, ..., 24	1, 2, ..., 24
Figure 12(a)	1, 2, 3, 4	1, 2, ..., 50	1, 2, ..., 50
Figure 13(a)	1, 2, 3, 4	1, 2, ..., 70	1, 2, ..., 70
Figure 14(a)	1, 2, 3, 4	1, 2, ..., 100	1, 2, ..., 100

into account a heterogeneous distributed system with w among PEs.

5. Numerical Examples

In this section, we explain and compare the results between our multistep scheduling algorithm and Shioda et al.'s algorithm [10] in terms of makespan and calculation time. In this experiment, we assumed the following two conditions. First, there are five task graphs, as shown in Figures 1(a), 3(a), 12(a), 13(a), and 14(a). Second, there are four identical PEs. Under

FIGURE 14: (a) Task graph (100 tasks), (b) results of clustering algorithm based on SCAN ($\epsilon = 0.1$, $\mu = 4$).

TABLE 10: Calculation time in multistep scheduling algorithm.

	Calculation time (sec)				
	Figure 1(a)	Figure 3(a)	Figure 12(a)	Figure 13(a)	Figure 14(a)
Clustering algorithm based on SCAN	0.26	0.10	0.26	0.52	1.24
Task scheduling	0.66	0.30	0.74	0.92	1.32
Clusters scheduling	0.02	0.03	0.03	0.08	0.08
Sum	0.94	0.43	1.03	1.52	2.64

TABLE 11: Comparison results.

	Calculation time (sec)					Makespan				
	Figure 1(a)	Figure 3(a)	Figure 12(a)	Figure 13(a)	Figure 14(a)	Figure 1(a)	Figure 3(a)	Figure 12(a)	Figure 13(a)	Figure 14(a)
Shioda et al.'s algorithm [10]	7200	3.82	7200	7200	—	15	8	17	26	—
Proposed algorithm	0.94	0.43	1.03	1.52	2.64	23	12	23	34	49

FIGURE 15: Results for Figure 3(a) by using (a) Shioda et al.'s algorithm [10], (b) proposed multistep scheduling algorithm.

FIGURE 16: Results for Figure 1(a) by using (a) Shioda et al.'s algorithm [10], (b) proposed multistep scheduling algorithm.

these conditions, we do scheduling for the five task graphs by using each algorithm. Table 9 shows the preconditions for this experiment. The environment consisted of the following: Intel Core i5-2400 CPU @3.10 GHz, 4.00-GB memory, windows 7 Professional (64 bit), IBM ILOG CPLEX Interactive Optimizer 12.2.0.0.

Table 10 lists the calculation time for the five task graphs shown in Figures 1(a), 3(a), 12(a), 13(a), and 14(a) with our proposed multistep scheduling algorithm. In the proposed multistep scheduling algorithm, the results of applying our *clustering algorithm based on SCAN* to the five task graphs are represented in Figures 1(b), 3(c), 12(b), 13(b), and 14(b).

Table 11 lists the calculation times and makespan with each algorithm. Scheduling for the task graphs in Figures 1(a), 12(a), 13(a), and 14(a) took more than 7200 sec (2 hours) with Shioda et al.'s algorithm [10]. We then stopped the scheduling calculation at 7200 sec and obtained a provisional result for the task graphs in Figures 1(a), 12(a), and 13(a). However, Shioda et al.'s algorithm [10] could not get any provisional results for the task graph in Figure 14(a).

The scheduling results of Shioda et al.'s algorithm [10] and the proposed multistep scheduling algorithm for the task graph in Figures 3(a), 1(a), 12(a), 13(a), and 14(a) are represented by Gantt charts and shown in Figures 15, 16, 17, 18 and 19, respectively.

Figures 15–19 show the following facts. The proposed multistep scheduling algorithm is superior to Shioda et al.'s algorithm [10] in terms of calculation time. However, our proposed multistep scheduling algorithm is inferior to Shioda et al.'s algorithm in terms of makespan. The proposed algorithm decomposes a large task graph into smaller subtask graphs (clusters) and solves the problems, which reduces significant calculation time. Because the proposed algorithm provides a solution based on the decomposed problems, the obtained makespan is not necessarily shorter than Shioda et al.'s algorithm. In addition, the proposed multistep scheduling algorithm finds clusters that have task parallelism. Figures 15(b), 16(b), 17(b), 18(b), and 19 suggest that the proposed multistep scheduling algorithm enables us to do parallel processing locally.

FIGURE 17: Results for Figure 12(a) by using (a) Shioda et al.'s algorithm [10], (b) proposed multistep scheduling algorithm.

FIGURE 18: Results for Figure 13(a) by using (a) Shioda et al.'s algorithm [10], (b) proposed multistep scheduling algorithm.

FIGURE 19: Results for Figure 14(a) by using proposed multistep scheduling algorithm.

6. Conclusion

We considered two purposes which are to consider communication costs and speed up of calculation time and proposed a multistep scheduling algorithm for them. The first purpose was to develop a scheduling algorithm by considering communication costs by using 0-1 integer linear programming. Therefore, we proposed *clusters scheduling* algorithm based on Shioda et al.'s algorithm [10] and showed its efficiency through numerical experiments. We discussed our algorithm's efficiency under two special conditions; homogeneous and heterogeneous distributed systems with communication costs among PEs.

The second purpose was to quickly obtain a result for the scheduling problem considering task parallelism. Therefore, we proposed a *clustering algorithm based on SCAN*, which can find task parallelism efficiently in a task graph. The original

SCAN is an algorithm for a nondirected graph; however, our *clustering algorithm based on SCAN* can be applied to a DAG. In Section 5, we argued that the proposed multistep scheduling algorithm is superior to Shioda et al.'s algorithm [10] in terms of calculation time. In addition, the results suggest that the proposed multistep scheduling algorithm enables parallel processing to be done locally.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

- [1] R. Miller, W. Ewy, B. W. Corrigan et al., "How modeling and simulation have enhanced decision making in new drug

- development," *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 32, no. 2, pp. 185–197, 2005.
- [2] G. R. Cates, M. J. Steele, M. Mollaghasemi, and G. Rabadi, "Methods and tools for aerospace operations modeling and simulation: modeling the space shuttle," in *Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers*, pp. 754–762, December 2002.
 - [3] X. Chen, D. Wang, W. Liu, and Y. Han, "Simulation to design car interior trims based on NVH performance," in *Proceedings of the 9th International Conference on Computer-Aided Industrial Design and Conceptual Design: Multicultural Creation and Design (CAID/CD '08)*, pp. 499–503, November 2008.
 - [4] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors: a review of their common attributes," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 26–37, 2009.
 - [5] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, 2005.
 - [6] W. Sun, Y. Zhu, Z. Su, D. Jiao, and M. Li, "A priority-based task scheduling algorithm in grid," in *Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms and Programming (PAAP '10)*, pp. 311–315, IEEE, December 2010.
 - [7] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399–409, 2008.
 - [8] N. A. Bahnasawy, F. Omara, M. A. Koutb, and M. Mosa, "A new algorithm for static task scheduling for heterogeneous distributed computing systems," *African Journal of Mathematics and Computer Science Research*, vol. 4, no. 6, pp. 221–234, 2011.
 - [9] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '03)*, April 2003.
 - [10] H. Shioda, K. Konishi, and S. Shin, "Optimal task scheduling algorithm for parallel processing," in *Proceedings of the 2011 2nd International Congress on Computer Applications and Computational Science*, pp. 79–87, 2011.
 - [11] H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithms," *IEEE Transactions on Computers*, vol. 33, no. 11, pp. 1023–1029, 1984.
 - [12] U. Straccia, "Description logics with fuzzy concrete domains," <http://arxiv.org/abs/1207.1410>.
 - [13] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27–64, 2007.
 - [14] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "SCAN: a structural clustering algorithm for networks," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*, p. 833, USA, August 2007.