

A Task Duplication Based Scheduling Algorithm for Avoiding Useless Duplication

Koichi ASAKURA, Bing SHAO and Toyohide WATANABE

Department of Systems and Social Informatics,
Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8603 JAPAN
{[asakura](mailto:asakura@watanaabe.ss.is.nagoya-u.ac.jp),[fox_shao](mailto:fox_shao@watanaabe.ss.is.nagoya-u.ac.jp),[watanabe](mailto:watanabe@watanaabe.ss.is.nagoya-u.ac.jp)}@watanaabe.ss.is.nagoya-u.ac.jp

Abstract: *In this paper, we propose a task duplication based scheduling algorithm for avoiding useless duplication. In our algorithm, task duplication is divided into two phases: a task fill phase and a task duplication phase. In the task fill phase, tasks are duplicated and allocated to idle time slots in used processors. Namely, the task fill phase does not require additional idle processors. The task duplication phase is invoked only if there are idle processors to be left for a new allocation. Thus, our algorithm can consume computing facilities effectively. Experimental results show that our algorithm can achieve less computation time and less use of idle processors.*

Keywords: scheduling algorithm, task duplication, cluster computing, parallelizing compiler

1 Introduction

In parallel processing, a task scheduling algorithm is one of the most important topics for the sake of effective use of computing facilities. Especially, as the number of processors in a system increases, management of idle processors is very essential. Task duplication based (TDB) scheduling algorithms[1, 2] are the ideas for decreasing the number of idle processors. In task duplication based algorithms, a task is duplicated and allocated to plural processors. Namely, the same task is executed in plural processors simultaneously.

By simultaneous execution of the same task, other tasks which have dependency relationships with the duplicated task can be executed earlier, and thus the processing time of whole tasks can be decreased.

Many task duplication based scheduling algorithms have been proposed[3, 4, 5]. In these algorithms, all tasks are examined whether it can be duplicated or not, namely this duplication induces decrease of computation time of the whole tasks or not. In other words, it is assumed that there exist a lot of idle processors so that duplicable tasks can be allocated to new idle processors. Therefore, these algorithms have such disadvantages that too many tasks are duplicated and many idle processors have to be consumed. Furthermore, in limited number of processors environment, inadequate tasks are duplicated and the computation time of the whole tasks cannot be decreased.

In this paper, we propose a task duplication based scheduling algorithm for avoiding useless duplication. In our algorithm, tasks are duplicated if the duplication leads decrease of computation time. Namely, only effective duplication is achieved in our algorithm. So, our algorithm is suitable for the environment in which there are not enough idle processors.

The rest of this paper is organized as follows. In Section 2, a DAG and other notations

are introduced as a task scheduling model, and task duplication based scheduling algorithms are described in detail. In Section 3, our proposed algorithm is described. In Section 4, experimental results are explained in order to confirm the advantage of our algorithm. In Section 5, our conclusion and future work are described.

2 Task Duplication Based Algorithms

In this section, we describe current task duplication based scheduling algorithms, and the problems in these algorithms. Before the detailed description of algorithms, we explain a model of task scheduling. Namely, a DAG and other notations are described.

2.1 Task Scheduling Model

Generally, in task scheduling, an actual parallel application is transformed into a Directed Acyclic Graph (DAG). In DAG, a task is denoted as a node, and a dependency relationship between tasks is denoted as an edge. Each node has a computation cost, and each edge has a communication cost. Figure 1(a) shows a typical DAG. The following is the notation of our task scheduling model.

- DAG: $G = (V, E)$.
- V : set of task T_i .
- E : set of dependency (T_j, T_k) .
- $cost(T_i)$ [ut]: computation cost of a task T_i .
- $comm(T_i, T_j)$ [ut]: communication cost between two tasks T_i and T_j .
- P : set of available processor p_i .

The [ut] stands for a unit time.

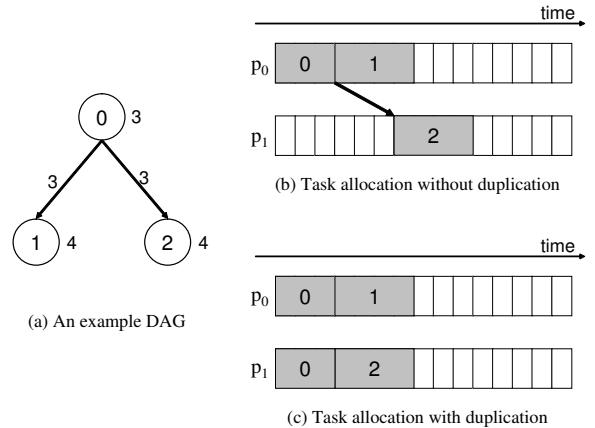


Figure 1: DAG and task duplication based scheduling

In our task scheduling model, if two depended tasks are allocated to the same processor, the communication cost between two tasks is regarded as zero. Figure 1(b) shows an example task scheduling. Since T_1 and T_2 are allocated to the same processor p_0 , the communication cost between two tasks becomes zero and T_2 is allocated just after the completion of execution T_1 .

2.2 Duplication Based Algorithms

As shown in Figure 1(b), the execution time for the whole tasks is 10[ut] without task duplication. This is because communication between T_0 and T_2 occurs and the start time of T_2 is 6[ut]. When T_0 is duplicated and allocated to both p_0 and p_1 , the communication cost can be ignored. Figure 1(c) shows this situation. No communication overhead occurs and thus the whole execution time becomes 7[ut]. This is a basic concept of task duplication based scheduling algorithms.

Several task duplication based scheduling algorithms have been proposed. First, in the algorithm, all the tasks are grouped into some task clusters, and each task cluster is allocated to a processor. Then, if there exists idle processors, tasks are duplicated and allocated to the idle processors in order to minimize the

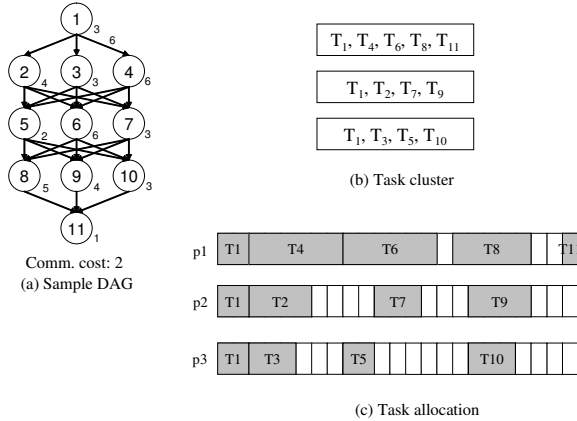


Figure 2: Task duplication based scheduling

total computing time of tasks.

The processing steps of the algorithm is described with an example DAG shown in Figure 2(a). First, a task cluster is generated. For a DAG in Figure 2(a), task clusters are organized as follows: $\{T_1, T_4, T_6, T_8, T_{11}\}$, $\{T_1, T_2, T_7, T_9\}$ and $\{T_1, T_3, T_5, T_{10}\}$. Task clusters are allocated to processors individually. The result of allocation for task clusters is shown in Figure 2(b). In generation of task clusters, the number of available processors are not taken into account. Thus, when the size of DAG is large, the number of generated task clusters also increases. This is one of the disadvantage in algorithms.

After allocating task clusters, task duplication is achieved. In the task duplication phase, tasks which are allocated to the tail of each processors are focused on. For these tasks, if the predecessor task does not has the highest value of *level*, this task cluster is duplicated and tasks in another task cluster are assigned to the front of the task. Here, *level* is defined as the computation cost to the exit node. For example, we focus on T_9 in Figure 2(b). T_7 is the predecessor task of T_9 . As shown in Figure 2(a), T_9 depends on T_5 , T_6 , T_7 , and T_6 has the highest value of *level* ($T_6 \rightarrow T_8 \rightarrow T_{11}$). So, the task cluster is duplicated and the other task cluster which includes T_6 is assigned. This operation

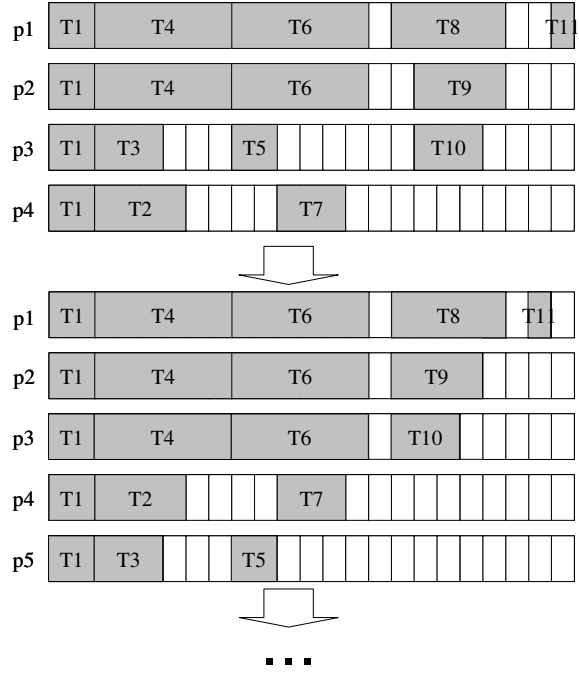


Figure 3: Result of task duplication

is repeated until all the predecessors of the task have the highest value of *level*. Figure 3 shows the processing result of the task duplication phase. This phase also requires that the enough number of idle processors are used for task duplication.

3 Our Proposed Algorithm

3.1 Concept of Our Algorithm

As shown in the previous section, current task duplication based scheduling algorithms requires the enough number of idle processors for generating task clusters and task duplication. As the size of application programs becomes large, the number of tasks in the application programs also increases and thus the number of task clusters increases rapidly. Therefore, effectiveness of task duplication is not so improved in actual systems in which limited number of processors can be used.

In order to solve the above problem, we pro-

pose a task duplication based scheduling algorithm which avoids useless duplication and uses limited number of idle processors effectively. Our algorithm is based on traditional BNP (Bounded Number of Processors) scheduling algorithms[6, 7]. Our algorithm consists of three phases.

1. BNP scheduling phase: tasks are allocated to processors by a traditional BNP algorithm.
2. Task full phase: if idle time slots exist in processors, a task is duplicated and allocated to the time slot.
3. Task duplication phase: the task which has the highest effectiveness of task duplication is duplicated to an idle processor.

In the first and second phases, extra idle processors are not consumed. Namely, in our method, task allocation is achieved in minimum number of processors in the first and second phases. The third phase is invoked only if there exist idle processors in a system. Thus, on small systems which consists of small number of processors, effective task duplication based allocation is achieved in the first and second phases. Additionally, on large systems in which idle processors are left unused, the third phase is invoked and tasks are duplicated and allocated to remaining idle processors. Therefore, our algorithm can be adapted effectively to both small and large systems.

3.2 Task Fill

In the task fill phase, a task is duplicated and allocated to idle time slots in a processor, which makes the execution time of the whole tasks short without using additional idle processors.

The example of task fill is shown in Figure 4. The top figure in Figure 4(b) is a result of task allocation for the DAG shown in

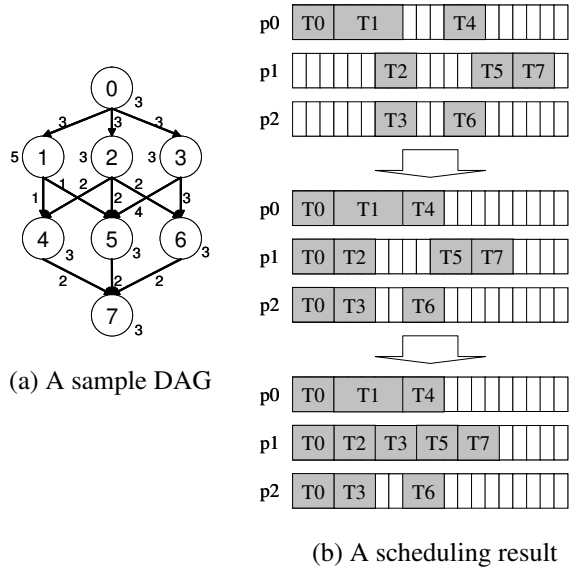


Figure 4: A processing of the task fill phase

Figure 4(a) by a BNP scheduling algorithm. In this result, the execution of T_2 and T_3 is started when the execution of T_0 is finished and communication among them is finished. In the task fill phase, T_0 is duplicated and allocated to the idle time slots p_1 and p_2 , which reduces the waiting time of T_2 and T_3 . Additionally, there is an idle time slot between T_2 and T_5 , and T_3 can be also duplicated to this time slot. By this processing, as shown in Figure 4(b), communication between tasks can be deleted and the start time of tasks can be accelerated.

Figure 5 shows notations used in the algorithm and Figure 6 shows the task fill algorithm. For each task, the task fill condition is evaluated: there are idle time slots enough to be allocated to the task which is depended upon by the focused task, in front of the focused task. If there are enough idle time slots, the depended task is duplicated and allocated in front of the focused task. By task fill, the communication time can be deleted between two tasks and the computation time of the whole tasks can be reduced even though the number of used processors is fixed.

```

est(i, p) := max(ect(j) + comm(i, j), ect(k)) s.t. j is not in p, but k is in p.
est(i)   := max(est(i,p))
ect(i)   := est(i) + cost(i)
level(i) := cost(i) for exit node,
          max(level(i)+comm(i,j)) + cost(i), j is successor of i.
fpred(i) := k: k is predecessor of i and has the highest value of level.

est: earliest start time
ect: earliest computation time

```

Figure 5: Notation

```

INPUT: queue: task queue

begin
  while queue is not empty do
    ti := top of queue.
    delete ti from queue.
    p := processor which ti is allocated to.
    tj := task which ti is waiting for.
    if (there is idle time slot ts in p s.t. size(ts) > size(tj)) then
      if (est(tj,p) > ect(ti,p) && est(tj,p)+size(tj) < est(ts,p)) then
        duplicate tj and allocate tj to p.
      endif
    endif
  done
end

```

Figure 6: Algorithm in task fill phase

3.3 Task Duplication

In the task duplication phase, it is important to select tasks to be duplicated so that the duplication makes the computation time of the whole tasks short. In order to avoid useless duplications, we adopt a depth first search method in selecting tasks to be duplicated. First, a candidate task for duplication is selected. In this selection, a candidate task is searched from the tail task to the head task. This is because a duplication of head tasks is more effective than that of tail tasks. So, in our algorithm, a head task which satisfies two conditions is selected as a candidate task. These conditions are as follows.

1. The task in front of a candidate task is in a best predecessor group of a candidate task.
2. There are no idle time slots among tasks in a best predecessor group of a candidate task.

```

INPUT: queue: task queue.

begin
  while (there is an idle processor) do
    ti := top of queue.
    delete ti from queue.
    tj := task which ti is waiting for.
    if (tj is null) continue. // this series of tasks are not duplicated.
    while (not(cond1(tj) && cond2(tj))) do
      ti := tj.
      tj := task which ti is waiting for.
      if (tj is null) break. // this series of tasks are not duplicated.
    done.
    if (tj is null) continue. // this series of tasks are not duplicated.

    // here tj is a candidate task for duplication
    Duplicate tj and allocate tj to new idle processor.
  done.
end

cond1(t) := tasks in front of t is best predecessor group of t.
cond2(t) := no idle time slots exist between t and best predecessor group of t.

```

Figure 7: Algorithm in task duplication phase

Here, the best predecessor group is defined as follows: if T_i and T_j are in the best predecessor group of T_k , T_i is the **fpred** of T_j and T_j is the **fpred** of T_k , respectively. If these conditions are satisfied, the tasks in the front of a candidate task cannot be duplicated because current algorithm is the best for these tasks. By this method, we can find a candidate task which exists in the forward position. For this selection, we can achieve effective task duplication. Figure 7 shows the in the task duplication phase.

4 Performance Evaluation

In order to verify the effectiveness of our algorithm, we perform some simulated experiments. In experiments, the Standard Task Graph Set (STG) is adopted as example DAGs[8]. The STG is a kind of a benchmark for evaluation of multiprocessor scheduling algorithms. The STG consists of task graphs generated randomly and modeled from actual application programs. However, communication cost between tasks is not included in STG. Thus, we assign communication costs to communication links randomly. In the first experiment, we compare the computation times of the whole tasks in both a tra-

Table 1: Comparison of BNP and our algorithm

Num. of Tasks	Comm. Cost	BNP Algo. (sec)	Our Algo. (sec)	Speed Up
50	10	95.8	92.9	103.1%
	20	168.4	142.9	117.8%
	50	299.6	214.7	139.5%
100	10	175.0	163.6	107.0%
	20	244.0	211.0	115.6%
	50	314.5	252.0	124.8%
500	10	551.8	526.7	104.8%
	20	765.5	682.7	112.1%
	50	972.6	818.8	118.8%

Table 2: Comparison of TDB and our algorithm

Num. of Tasks	Comm. Cost	Our Algo. (sec)	TDB Algo. (sec)	Speed Up	Num. of Proc. in		Ratio
					our algo.	TDB algo.	
50	10	92.9	89.8	103.5%	13.1	18.6	142.1%
	20	142.9	140.4	101.8%	11.5	14.8	128.9%
	50	214.7	214.1	100.3%	9.4	13.0	138.5%
100	10	163.6	161.3	101.4%	19.2	22.8	118.8%
	20	211.0	208.6	101.2%	18.3	21.9	119.5%
	50	252.0	251.0	100.4%	17.6	20.7	117.6%
500	10	526.7	525.4	100.2%	34.3	36.7	107.0%
	20	682.7	682.1	100.1%	33.9	35.3	104.1%
	50	818.8	818.0	100.1%	33.8	35.5	105.1%

ditional BNP scheduling algorithm and our algorithm. This experiment shows the effectiveness of the task fill phase and the task duplication phase. In the second experiment, we compare the computation times of the whole tasks and the number of processors used in traditional TDB algorithm and our algorithm. This experiment shows whether useless duplication occurs or not.

Table 1 shows the experimental result for comparing the computation times by traditional BNP algorithm and our algorithm. This result shows that computation time of all DAGs by our algorithm is less than that of

the traditional BNP algorithm. From this experimental result, we can conclude that task duplication strategy of our algorithm is correct.

Table 2 shows the computation times and the number of processors used in both the traditional TDB algorithm and our algorithm. This shows that for all DAGs, TDB algorithm consumes more additional processors although the computation time is not so shortened. From this experimental result, we clarify that useless task duplication can be prevented in our algorithm.

From these experimental results, we can

conclude that our task duplication based algorithm prevents useless task duplications and makes the whole computation time of DAGs short.

5 Conclusion

In this paper, we proposed a task duplication based scheduling algorithm. Our algorithm takes reduction of useless task duplication into account. In order to reduce the useless task duplication, the process of task duplication is divided into two phases in our algorithm: the task fill phase and the task duplication phase. In the task fill phase, tasks are duplicated and assigned to idle time slots. In this phase, additional idle processors are not required. In task duplication phase, tasks are duplicated and assigned to idle processors. In this phase, in order to prevent useless task duplication, a depth-first searching is adopted and only tasks required to be duplicated are duplicated. From experimental results, in comparison with other algorithms, our algorithm makes the whole computation time short and makes the number of used processors small.

For future work, we have to evaluate our algorithm with actual application programs. In addition, we have to evaluate our algorithm with an environment in which limited number of computing facilities can be used.

References

- [1] B. Kruatrachue and T.G. Lewis: "Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems", Technical Report OR97331, Oregon University (1987).
- [2] S. Darbha and D.P. Agrawal: "SDBS: A Task Duplication Based Optimal Scheduling Algorithm", *Proc. on Scalable High Performance Computing Conference*, pp. 756–763 (1994).
- [3] S. Darbha and D.P. Agrawal: "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems", *Journal of Parallel and Distributed Computing*, Vol. 46, No. 1, pp. 15–27 (1997).
- [4] S. Darnha and D.P. Agrawal: "Optimal Scheduling Algorithm for Distributed Memory Machines", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 9, No. 1, pp. 87–95 (1998).
- [5] S. Ranaweera and D.P. Agrawal: "A Task Duplication Based Scalable Scheduling Algorithm for Heterogeneous Systems", *Proc. on the 14th Int'l Parallel and Distributed Processing Systems (IPDPS'00)*, pp. 445–450 (2000).
- [6] G.C. Sih and E.A. Lee: "A Compile-time Scheduling Heuristic for Interconnection-constrained Heterogeneous Processor Architectures", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 2, pp. 75–87 (1993).
- [7] Y.K. Kwork and I. Ahmad: "Dynamic Critical-path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506–521 (1996).
- [8] T. Tobita and H. Kasahara: "Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set", *Proc. on the 2000 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 745–751 (2000).