A Task Duplication Based Scheduling Algorithm with Optimality Condition in Heterogeneous Systems

Tae-Young Choe and Chan-Ik Park
Department of Computer Science and Engineering/PIRL
Pohang University of Science and Technology
Kyungbuk, Republic of Korea
{choety,cipark}@postech.ac.kr

Abstract

The task scheduling problem is NP-hard in heterogeneous systems. We propose a task scheduling algorithm based on task duplication with an optimality condition to determine whether or not the resulting schedule has the shortest schedule length. The optimality condition is that, given any join task, the completion time of a parent task is longer than the maximum message arrival times from the other parent tasks. An illustrative example is given to show how our algorithm differs from existing algorithms.

1 Introduction

With the advance of high speed network technology, parallel computing systems are gaining in importance [7, 1]. Applications which consist of multiple tasks with precedence relations can be expressed as directed acyclic graphs (DAGs) with weighted nodes and weighted edges, where nodes represent tasks and edges represent communication between tasks [9]. The scheduling problem is to allocate nodes of a DAG to processors of a parallel computing system with the minimum schedule length; that is, the completion time of the application. Since the scheduling problem was proven to be NP-hard [6], many heuristic scheduling algorithms have been proposed [3, 10, 4, 12, 11]. Most of these algorithms target for homogeneous systems [9].

We assume that a DAG $G = (V, E, \tau, c)$ represents an application where V represents a set of tasks $\{n_i\}$, τ represents a set of weights of tasks $\{\tau_i | n_i \in V\}$, E represents the communication patterns among tasks $\{e_{i,j} | n_i, n_j \in V\}$, and the set of the weight $c_{i.j}$ of edge $e_{i,j}$ is given by $c = \{c_{i,j} | e_{i,j} \in E\}$. Given any task n_i, $PRED(n_i)$ is the set of parent tasks of n_i; that is, $PRED(n_i) = \{n_j | e_{j,i} \in E\}$, and $SUCC(n_i)$ is the set of child tasks of n_i; that is, $SUCC(n_i) = \{n_j | e_{i,j} \in E\}$. We designate n_i as an *entry task* if $|PRED(n_i)| = 0$, and an *exit task* if $|SUCC(n_i)| = 0$. Without loss of generality, we assume only one entry and exit task, which we will denote as n_α and n_ω, respectively. If a task is allocated to processors, each task in its processor have a start time and a completion time. Accordingly, the problem of task scheduling can be expressed as follows: We define the *schedule length* as the completion time of n_ω.

> Given an application which is expressed as a DAG $G = (V, E, \tau, c)$, the problem is to allocate all tasks to a sufficient number of processors in order to minimize the schedule length.

We focus on scheduling algorithms with optimality conditions, which is defined as follows: If a given DAG satisfies *optimality conditions* of a task scheduling algorithm, the task scheduling algorithm schedules the DAG with the shortest schedule length. Such algorithms have the merit that schedule with the shortest length is guaranteed and no further refinements are required as long as input DAGs satisfy the conditions.

Recently, because heterogeneous systems such as PC clusters [14] and grid computing [5] have become popular, we need efficient scheduling for heterogeneous systems. In this paper, we consider scheduling algorithms in heterogeneous systems given two properties. First, the execution time of a task in a processor is the multiplication of the weight of the task and the execution time of the unit data in the processor [8, 2]; Second, communication overhead does not depend on the source processor or the target processor [12, 10, 13]. The heterogeneous systems are a model of multiprocessor systems where the CPUs differ in terms of processing speed only.

The heterogeneous systems is modeled as follows: There are p processors denoted as $P_1, P_2, \ldots, P_p$, and the execution time required to process a single unit of data in processor P_i is s_i. The set of processor power s_i is $s = \{s_1, s_2, \ldots, s_p\}$. We assume that $s_i \leq s_{i+1}$ for $1 \leq i <$

Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)
1530-2016/02 $17.00 © 2002 IEEE

p, and p is sufficiently large to contain schedules by any scheduling algorithm. In general, p is smaller than the number of tasks. The execution time of task n_i in processor P_j is $\tau_i \cdot s_j$. The problem of task scheduling in heterogeneous systems is the same as that in homogeneous systems.

Some task scheduling algorithms that present optimality conditions are proposed in homogeneous systems [3, 4, 11]. We previously proposed a task scheduling algorithm based on task duplication in homogeneous systems [11]. The algorithm constructs clusters in top-down style and presents an optimality condition such that extracting tasks from a cluster increases the completion time of the cluster.

Ranaweera and Agrawal proposed a scheduling algorithm called the STDS algorithm [12], which expands the TDS algorithm [4] in homogeneous systems to heterogeneous systems. Given a join task, the STDS algorithm allocates the join task to the same processor where a parent task sending the last message to the join task is assigned. The parent task is called the *fpred* task of the join task. If the *fpred* task is already allocated to a processor, another parent task with the smallest execution time is selected. After all tasks are allocated, the STDS algorithm tries to reduce the schedule length by duplicating a fpred chain of each join task. Duplicating fpred chain of a join task in a processor requires the tasks allocated in the processor to be moved to an idle processor. Note that available idle processors are slower than any processors currently holding tasks. Therefore, processor selection due to task duplication in the STDS is quite inefficient, overlooking global structures of processors.

We propose a scheduling algorithm which runs in heterogeneous systems and an optimality criterion to determine whether or not the resulting schedule by the proposed algorithm is optimal. The key point of our scheduling algorithm is that tasks are duplicated before tasks are allocated to processors during scheduling, which takes advantage of the global structures of DAGs. The remainder of this paper is organized as follows: the proposed scheduling algorithm and the derivation of optimality criteria are described in Section 2, and an illustrative example is shown in Section 3. Finally, Section 4 presents conclusions.

2 Proposed Algorithm

2.1 Descriptions of the proposed algorithm

Figure 3 (a) shows our proposed scheduling algorithm. The algorithm schedules a DAG in three steps: First, it computes values *fpred*, *CPRED*, *fpred'*, *est*, *ect*, and *rdy* and the cluster of each task. Second, it maps clusters to processors. Finally, it computes the start time and completion time of tasks in processors.

Figure 3 (a) Line 1∼6 shows the first step of the algorithm. For each task n_a, the following values are computed as shown in Figure 1. From a given task to the entry task n_α, the $fpred$ relation generates a path, which is called a *fpred chain*. A *cluster* is a set of tasks which will be assigned to a processor. For each task n_a, we define a cluster $C(n_a)$ as the fpred chain of n_a, as follows:

$$\begin{aligned} C(n_\alpha) &= \{n_\alpha\} \\ C(n_a) &= C(fpred(n_a)) \cup \{n_a\} \end{aligned}$$

In order to compute the cluster of a task, the $fpred$ task of the task should be available, which requires est of all its parent tasks. A task where all the ests of the parent tasks are available and its est is not yet computed is called 'ready,' a task where the est is available is called 'finished,' and other tasks are called 'unready.' Initially, n_α is ready and the other tasks are unready. After the values of a ready task are computed, the task is set as 'finished,' and child tasks are checked to determine whether or not they are 'ready.'

In the second step, redundant clusters are deleted, as shown in Line 7. A redundant cluster is a proper subset of another cluster. Next, clusters are mapped to processors in Line 8∼11: $C(n_\omega)$ is mapped to the fastest processor, and the cluster with a larger ready time is mapped to a faster processor. The ready time of a cluster $C(n_a)$ is $\max_{n_i \in SUCC(n_a)} rdy(n_a, n_i)$. The motivation for the mapping policy is that the cluster with a larger ready time has a higher influence on the schedule length and allocating the cluster to a faster processor can reduce the schedule length.

In the third step, each task in each processor has two values: *start time* and *completion time*, which are computed in procedure ComputeST(), as shown in Figure 3 (b). Given a task n_a in a processor P_j, the start time and completion time of the task are denoted as $st_{P_j}(n_a)$ and $ct_{P_j}(n_a)$, respectively. They are defined as Figure 2.

2.2 Optimality conditions

After schedule of the proposed algoirthm is finished, all tasks in processors have their start and completion time. Given a task n_a, let the set of processors which contain n_a be $\mathcal{P}(n_a)$.

Theorem 1. *If the condition of Equation 3 is satisfied for any join task n_a and any processor $P_k \in \mathcal{P}(n_a)$ in a schedule generated by algorithm* HTschedule()*, then the schedule has the shortest length.*

$$ct_{P_k}(fpred(n_a)) \geq \max_{n_i \in CPRED(n_a)} (\min_{P_j \in \mathcal{P}(n_i)} (ct_{P_j}(n_i)) + c_{i,a}). \quad (3)$$

$$
\begin{aligned}
fpred(n_a) &= n_i | rdy(n_i, n_a) \geq rdy(n_j, n_a), \forall n_i, n_j \in PRED(n_a), \\
CPRED(n_a) &= PRED(n_a) - \{fpred(n_a)\}, \\
fpred'(n_a) &= n_i | rdy(n_i, n_a) \geq rdy(n_j, n_a), \forall n_i, n_j \in CPRED(n_a) \\
est(n_a) &= \begin{cases} 0, & \text{if } PRED(n_a) = \emptyset, \\ \max(ect(fpred(n_a)), rdy(fpred'(n_a), n_a)), & \text{otherwise,} \end{cases} \\
ect(n_a) &= est(n_a) + \tau_a, \\
rdy(n_i, n_a) &= ect(n_i) + c_{i,a}.
\end{aligned}
$$

Figure 1. Properties of each task n_a

$$
\begin{aligned}
st_{P_j}(n_a) &= \max(ct_{P_j}(fpred(n_a)), \max_{n_i \in CPRED(n_a)} (\min_{P_k \ni n_i} rdy_{P_k}(n_i, n_a))), & (1) \\
ct_{P_j}(n_a) &= st_{P_j}(n_a) + \tau_a s_j, & (2) \\
rdy_{P_j}(n_a, n_i) &= ct_{P_j}(n_a) + c_{a,i}.
\end{aligned}
$$

Figure 2. Properties of task n_a in processor P_j

Proof. If a shorter schedule exists than that by the algorithm HTschedule(), the schedule by HTschedule() can be transformed to a shorter schedule by a combination of the following three primitive transform operations: The addition of tasks to processors, the extraction of tasks from processors, and the exchange of processors allocation. If all operations do not reduce the schedule length, any combination of the operations does not reduce the schedule length, which means that the schedule by HTschedule() has the shortest schedule length. We show that each operation does not reduce the schedule length.

- The additions of tasks to processors
  By Equation 3, Equation 4 is satisfied for any join task n_a in processor P_k.

  $$ct_{P_k}(fpred(n_a)) = st_{P_k}(n_a), \quad (4)$$

  which means that no empty slot exists between tasks in the schedule by HTschedule(). Thus the addition of tasks to a processor increase its completion time.

- The extraction of tasks from processors
  An extraction of tasks from processor P_1 increases its completion time, because the extracted tasks cause communication overhead. An extraction of tasks from another processor could reduce its completion time, however, which does not affect the completion time of P_1.

- The processor allocation exchanges
  The operation means that cluster $C(n_i)$ which was mapped to processor P_i immigrates to processor P_j and $C(n_j)$ which was mapped to processor P_j immigrates to processor P_i. P_1 and $C(n_\omega)$ could not be a target of the operation, because its exchange increases the completion time of n_ω. While the exchanges of other processors reduce their completion time, they does not affect the completion time of n_ω.

□

2.3 Time complexity

If the maximum degree of input edges of tasks is d_i, the computation of values $fpred$, est, and cluster requires $O(d_i)$ steps, respectively. Such computations occur for each task; that is, $|V|$ times. Thus, the $O(d_i|V|)$ steps are required for the computation of the values. Next, checking the proper subset cluster requires $O(|V|^2)$ steps, because the size of the largest cluster is $|V|$ and a subset check of two clusters requires $|V|$ steps. Mapping clusters to processors requires no more than $|V|$ steps. In procedure ComputeST(), $|V|^2$ tasks are possible in all processors because tasks can be duplicated. The computation of the start time of each task in each processor requires $d_i|V|$ steps. Thus, the time complexity of ComputeST() is $O(d_i|V|^3)$. Since the time complexity of ComputeST() is the dominating time complexity over the algorithm HTschedule, the time complexity of the algorithm HTschedule is $O(d_i|V|^3)$.

```
Algorithm HTschedule(G = (V, E, τ, c), P)
Begin
   set all tasks 'unready';                                              1
   set n_α 'ready';                                                      2
   foreach (ready task n_a) do
      compute fpred, CPRED, fpred', est, ect, rdy of n_a;                3
      compute cluster C(n_a);                                            4
      check SUCC(n_a) for ready tasks;                                   5
      set n_a 'finished';                                                6
   end foreach
   erase included clusters;                                              7
   map cluster C(n_ω) to processor P_1;                                  8
   p = 1;                                                                9
   for (decreasing order of ready time cluster C_j) do
      p = p + 1;                                                         10
      map cluster C_j to processor P_p;                                  11
   end for
   ComputeST(p);                                                         12
End
```

(a)

```
Procedure ComputeST(p)
Begin
   // compute st and ct for all tasks in processors P_1, ..., P_p.
   set all tasks as 'unready task';                                      1
   set all n_α as 'ready task';                                          2
   foreach (ready task n_i in each processor P_j) do
      if (n_i == n_α) then
         st_{P_j}(n_α) = 0;                                             3
      else
         n_j = fpred(n_i);
         β = max_{n_i ∈ CPRED(n_a)} min_{P_k ∋ n_i} ct_{P_k}(n_i) + c_{i,a};  4
         st_{P_j}(n_i) = max(ct_{P_j}(n_j), β);                         5
      end if
      ct_{P_j}(n_i) = st_{P_j}(n_i) + τ_i s_j;                          6
      check SUCC(n_i) for ready tasks;                                  7
      set n_i in P_j 'finished';                                        8
   end foreach
End
```

(b)

Figure 3. (a) The main task scheduling algorithm HTschedule **in a heterogeneous environment, (b) algorithm** computeST **computes the start time st and the completion time ct of each task allocated in a processor**

3 An Example of the Scheduling

For the DAG given in Figure 4 (a), Figure 4 (b)-(e) shows the operation of the STDS algoirthm and Figure 4 (f)-(g) shows the operation of our scheduling algorithm. The available processors are $P_1 \sim P_6$, and the time to process the unit data of each process is $s = \{1.0, 1.1, 1.2, 1.3, 1.5, 1.6\}$.

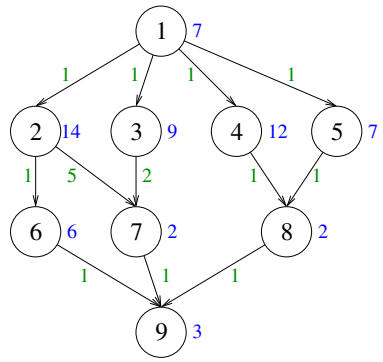
First, we explain the operation of the STDS algorithm. Values est, ect, $fpred$, $last$, $lact$, and $level$ of each tasks are computed. Tasks are inserted into a queue in the order of $level$, as {9,8,7,6,5,3,4,2,1}, as shown in Figure 4 (b). Cluster {1,2,6,9} is extracted from the queue and is allocated to the fastest processor of task 9, P_1. Task 7 is extracted from the queue, and task 3 is selected as the smallest parent task of task 7, because task 2 is already allocated to P_1. Thus, the second cluster {1,3,7} is extracted and is allocated to the next fastest processor, P_2. Through similar steps, cluster {1,4,8} is allocated to P_3, and cluster {1,5} is allocated to P_4, as shown in Figure 4 (c). At this point, the start times and completion times of all tasks in each processor are computed. Since the start time of task 9 is delayed by task 7, the fpred tail of task 7, that is, {1,2,7}, is duplicated to P_2 and tasks 1 and 3 immigrate to the next available processor, P_5, as shown in Figure 4 (d). However, duplication and immigration do not reduce the completion time of task 9. Because tasks 7 and 8 have the same level, the STDS algorithm attempted to allocate cluster {1,3,7} to P_2 and cluster {1,4,8} to P_3. However, it merely reduces the start time of task 9 from 29.4 to 29.2. Also, the duplication of the fpred tail of task 7 and the immigration of {1,3} do not reduce the ready time of task 7, as shown in Figure 4 (e).

Second, we explain the operation of our proposed algorithm. The algorithm computes the cluster of each task in addition to the values required by the STDS algorithm. The proper subset clusters, $C(1)$, $C(2)$, $C(4)$, and $C(6)$, are deleted, and the maximum ready time of each cluster is computed. As the maximum ready time of a cluster is larger, it is mapped to a faster processor. Thus, $C(9)$, $C(7)$, $C(8)$, $C(3)$, and $C(5)$ are mapped to P_1, P_2, P_3, P_4, and P_5, respectively, as shown in Figure 4 (f). Next, the start time and completion time of each task are computed, as shown in Figure 4 (g). Note that, for any join task, the completion time of a parent task is greater than the ready time of any other parent tasks in the figure; that is, Equation 3 is satisfied. Thus, the schedule has the shortest length.

4 Conclusions

In this paper we presented an optimal task scheduling algorithm based on task duplication in heterogeneous systems. In order to overcome the shortcomings of the STDS algorithm that separates task duplication and allocation, our scheduling algorithm integrates them in scheduling. The

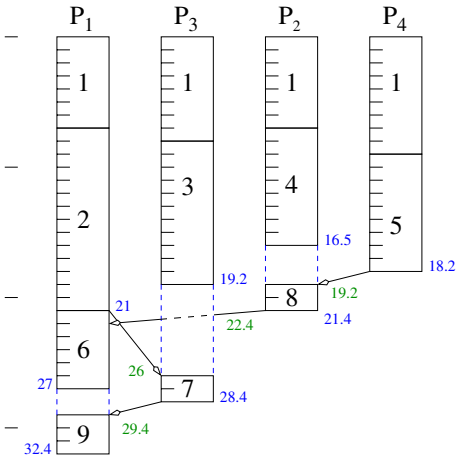
{s} = {1.0, 1.1, 1.2, 1.3, 1.5, 1.6}



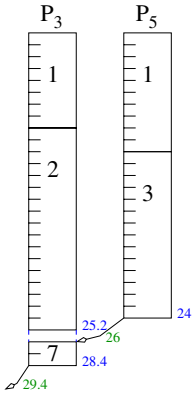
(a)

task	est	ect	fpred	level	last	lact	proc
1	0	7	-	30	0	7	P_1
2	7	21	1	23	7	21	P_1
3	7	16	1	14	13	22	P_3
4	7	19	1	17	12	24	P_2
5	7	14	1	12	16	23	P_4
6	21	27	2	9	21	27	P_1
7	14	16	2	5	24	26	P_3
8	19	20	4	5	24	26	P_2
9	27	30	6	3	27	30	P_1

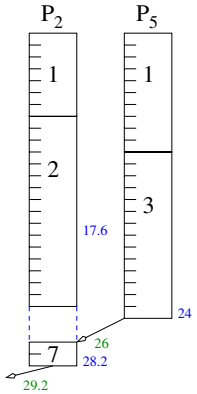
(b)



(c)



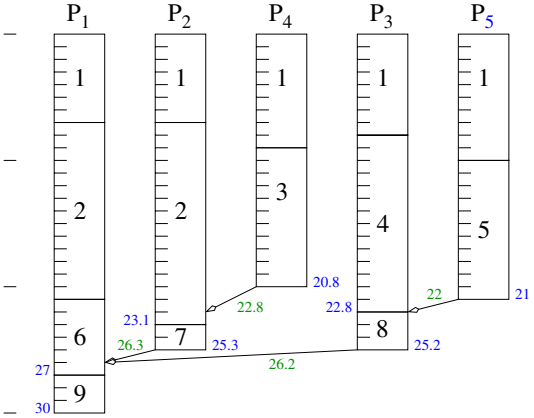
(d)



(e)

task	cluster	remain	rdy	proc
1	$\{1\}$	X	-	-
2	$\{1,2\}$	X	-	-
3	$\{1,3\}$	O	18	P_4
4	$\{1,4\}$	X	-	-
5	$\{1,5\}$	O	15	P_5
6	$\{1,2,6\}$	X	-	-
7	$\{1,2,7\}$	O	23	P_2
8	$\{1,4,8\}$	O	21	P_3
9	$\{1,2,6,9\}$	O	30	P_1

(f)



(g)

Figure 4. Examples of scheduling by the STDS algorithm and ours: (a) a DAG, (b) values of tasks for the STDS algorithm, (c) the first scheduling by the STDS algorithm without duplication, (d) an attempt to duplicate the parent tasks of task '7', (e) {1,2,7} is allocated to P_2 and {1,3} is allocated to P_5, (f) the values of tasks by our algorithm, (g) a scheduling by our proposed algorithm

optimality condition is that arrival times of messages from other processors are always equal to or smaller than the completion time of the previous task in the current processor. We are currently engaged in research on scheduling algorithms with expanded optimality conditions in heterogeneous systems.

acknowledgment

This work has been supported in part by the Ministry of Education of Korea through BK21 program.

References

[1] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *Proceedings of Symposium on High Performance Distributed Computing*, Syracuse, NY, August 1996.

[2] B. Cirou and E. Jeannot. Triplet: a clustering scheduling algorithm for heterogeneous systems. In *International Conference on Parallel Processing Workshops*, pages 231–236, Valencia, Spain, September 2001.

[3] J. Y. Colin and P. Chretienne. C.p.m. scheduling with small computation delays and task duplication. *Operations Research*, pages 680–684, 1991.

[4] S. Darbha and D. P. Agrawal. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems*, 9(1):87–95, January 1998.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.

[6] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[7] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjumin/Cumming, Redwood City, CA, 1994.

[8] Y.-K. Kwok. Parallel program execution on a heterogeneous pc cluster using task duplication. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'2000)*, pages 364–374, Cancun, Mexico, May 2000.

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

[10] H. Oh and S. Ha. A static scheduling heuristic for heterogeneous processors. In *Euro-Par'96*, Lyon, France, August 1996.

[11] C.-I. Park and T.-Y. Choe. An optimal scheduling algorithm based on task duplication. *IEEE Transactions of Computers*, 51(4):444–448, April 2002.

[12] S. Ranaweera and D. P. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of the 14th International Conference on Parallel and Distributed Processing Symposium (IPDPS-00)*, pages 445–450, Los Alamitos, May 1–5 2000. IEEE.

[13] A. Rădulescu and A. J. C. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *The 9th Heterogeneous Computing Workshop (HCW)*, pages 229–238, Cancun, Mexico, May 2000.

[14] T. Sterling. An introduction to PC clusters for high performance computing. *The International Journal of High Performance Computing Applications*, 15(2):92–101, Summer 2001.