An Optimal Scheduling Algorithm based on Task Duplication

Chan-Ik Park Tae-Young Choe
Department of Computer Science and Engineering,
POSTECH, San 31, HyoJa Dong,
Pohang, KOREA 790-784
{cipark,choety}@postech.ac.kr

Abstract

The task scheduling problem in distributed memory machines is to allocate the tasks of an application into processors in order to minimize the total execution time. This is known as an NP-complete problem. Under the condition where the communication time is relatively shorter than the computation time for every task, the task duplication based scheduling (TDS) algorithm proposed by Darbha and Agrawal generates an optimal schedule. In this paper, we propose an extended TDS algorithm whose optimality condition is less restricted than the TDS algorithm. Given a DAG where the condition is met, our algorithm has the time complexity of $O(|V|^2d^2)$ where $|V|$ represents the number of tasks, and d represents the maximum degree of tasks.
keywords: *schedule, parallel process, task duplication, task clustering*

1. Introduction

An application can be expressed as a directed acyclic graph (DAG) $G = (V, E, \tau, c)$, where V represents the set of $|V|$ tasks, and E, the set of $|E|$ directed edges. The size of task n_i is τ_i. An edge from task n_i to task n_j is expressed as e_{ij}, and the size of the edge is expressed as c_{ij}. Given two tasks n_i and n_j, if there is an edge c_{ij}, n_i is a parent task of n_j and n_j is a child task of n_i. Given task n_a, the set of parent tasks is denoted as $pred(n_a)$; that is, $pred(n_a) = \{n_i | e_{ia} \in E\}$, and the set of children tasks is denoted as $succ(n_a)$; that is, $succ(n_a) = \{n_i | e_{ai} \in E\}$. A task n_a is called a join task if $|pred(n_a)| \geq 2$, an entry task if $|pred(n_a)| = 0$, and an exit task if $|succ(n_a)| = 0$. Without loss of generality, we assume only one entry and exit task denoted as n_α and n_ω, respectively.

The term *cluster* refers to a set of tasks where the start and completion time of each task in the cluster are fixed. That is, task n_i in cluster C has $st_C(n_i)$ and $ct_C(n_i)$ as the start and completion time, respectively. Since we assume that tasks are nonpreemptable, $ct_C(n_i) = st_C(n_i) + \tau_i$. Moreover, the interval $(st_C(n_i), ct_C(n_i))$ of task n_i in cluster C should not be overlapped by the interval of any other tasks. The start time of a task is determined by the completion time of its parent tasks and the maximum communication time (i.e., edge size) between the task and its parent tasks. Such properties are expressed in equations as follows: n_i and n_j are tasks, and C_i and C_j are clusters;

- if $n_i, n_j \in C_i$, then either $ct_{C_i}(n_i) \leq st_{C_i}(n_j)$ or $ct_{C_i}(n_j) \leq st_{C_i}(n_i)$;
- if $n_i, n_j \in C_i$ and $e_{ij} \in E$, then $ct_{C_i}(n_i) \leq st_{C_i}(n_j)$;
- if $n_i \in C_i - C_j$, $n_j \in C_j$, and $e_{ij} \in E$, then $ct_{C_i}(n_i) + c_{ij} \leq st_{C_j}(n_j)$.

The communication time among tasks in the same cluster is assumed to be negligible because a cluster is mapped to one processor in task scheduling. Given task n_i, $est(n_i)$ represents the earliest possible start time of n_i. Therefore, $est(n_i)$ will always be less than or equal to $st_C(n_i)$ for any cluster C. $ect(n_i)$ represents the earliest possible completion time of n_i.

The task scheduling problem is to allocate tasks to clusters (i.e., processors) in order to minimize the completion time of the exit task, where tasks can be duplicated over multiple clusters. It is shown in [1] that task duplication based scheduling algorithms always perform better than any other scheduling algorithms with no task duplication.

The scheduling problem is known as NP-complete whether or not the tasks can be duplicated [2, 3, 4]. Many scheduling algorithms have been proposed using heuristic methods [1, 4, 5, 6, 7]. Some are able to find an optimal schedule when certain conditions are met for the given task set [7, 8]. For example, Colin and Chretienne proposed the LWB scheduling algorithm [8], which generates an optimal schedule if

$$\min_{n_i \in pred(n_a)} \tau_i \geq \max_{n_i \in pred(n_a)} c_{ia}$$

for any join task n_a. That is, for each join task, the incoming communication time must be shorter than the computation time of each parent task. Darbha and Agrawal proposed a scheduling algorithm based on task duplication in [7]. Their algorithm (called TDS algorithm) produces an optimal schedule if any join task n_a satisfies one of the following two conditions:

$$\text{if } est(n_1) \geq est(n_2),\ \tau_1 \geq c_{2a} \quad (1)$$

$$\text{if } est(n_1) < est(n_2),\ \tau_1 \geq (c_{2a} + est(n_2) - est(n_1)), \quad (2)$$

where task n_1 and n_2 has the highest and the next highest value of $\{ect(n_j)+c_{ja} | n_j \in pred(n_a)\}$ for join task n_a, respectively. Note that the optimality condition of Darbha and Agrawal's TDS algorithm is less restricted than the optimality condition of Colin and Chretienne's LWB algorithm.

In this paper, we propose an optimal scheduling algorithm with a less restricted optimality condition than the TDS algorithm. The rest of this paper is organized as follows; the proposed scheduling algorithm and the optimality condition are described in Section 2, and an illustrative example is shown in Section 3. Finally, Section 4 provides conclusions and directions for future work.

2. Algorithm

To begin with, we introduce some terminology. Given task n_i, $C(n_i)$ is a cluster where $est(n_i)$ and $ect(n_i)$ are determined by the start time and the completion time of n_i in the cluster. In other words, $est(n_i)$ is defined as $st_{C(n_i)}(n_i)$, and $ect(n_i)$ is defined as $ct_{C(n_i)}(n_i)$. Initially, for the entry task n_α, we obtain $C(n_\alpha) = \{n_\alpha\}$ and $est(n_\alpha) = st_{C(n_\alpha)}(n_\alpha) = 0$. The main step of our scheduling process shown in Figure 1 is the procedure Build_C_and_est() which constructs cluster $C(n_i)$ using the clusters of parent tasks of n_i, which we call *parent clusters*. Note that the start time of a task is determined when the task is included in a cluster. If all parent clusters of a task are available, the task is called *ready* task. Since all parent clusters are required to compute *est* of a task, the procedure Build_C_and_est() is applied for ready tasks only. $ReadyTaskSet$ is the set of ready tasks.

Before describing the details of Build_C_and_est(), we explain how a cluster of a given task is constructed. When task n_i is merged into cluster C, it forms a new cluster C' by $C \cup \{n_i\}$. The start time *st* of every task in C' must be computed. To clarify this point, we denote the merge operation as $\overline{\cup}$. Thus, cluster $C' = C \,\overline{\cup}\, \{n_i\}$. Likewise, the merge of two clusters can be defined as $C_i \,\overline{\cup}\, C_j$.

Given task n_a, the cluster $C(n_a)$ is determined according to the number of its parent tasks. If task n_a has a single parent task n_i (See Figure 2 (a)), then $C(n_a)$ should be $C(n_i) \,\overline{\cup}\, \{n_a\}$, as shown in Figure 2 (b). If n_a is not

```
Algorithm schedule(V, E, τ, c, Γ)
Begin
   // input: DAG (V, E, τ, c)
   // output: the set of clusters Γ
   C(n_α) ← {n_α};                                                    1
   est(n_α) ← 0; // entry task n_α                                    2
   ReadyTaskSet ← succ(n_α);                                          3
   while (|ReadyTaskSet| > 0) do
      n_a ← select a task in ReadyTaskSet;                            4
      Build_C_and_est(n_a, C(n_a), est(n_a));                         5
      ReadyTaskSet ← {n_r | For∀n_i ∈ pred(n_r), ∃C(n_i)};            6
   end while
   Γ ← {C(n_i) | n_i ∈ V};                                            7
End
```

Figure 1. The overall step of the proposed algorithm

merged into $C(n_i)$, i.e., $C(n_a) = \{n_a\}$, the execution of n_a is delayed by $ect(n_i) + c_{ia}$, as shown in Figure 2 (a). We define *ready time* of n_i to n_a, that is, $ect(n_i) + c_{ia}$, as $rdy(n_i, n_a)$, which means that if n_i is not allocated to the same cluster with n_a, n_a must wait until the message arrival time $rdy(n_i, n_a)$ from n_i. That is, execution of n_a is restricted by *ready times* of its parent tasks. If n_a is merged into $C(n_i)$, the communication time c_{ia} is negligible and n_a can start at $ect(n_i)$ as shown in Figure 2 (b). Therefore, we obtain $est(n_a) = ect(n_i)$ and $C(n_a) = C(n_i) \,\overline{\cup}\, \{n_a\}$.

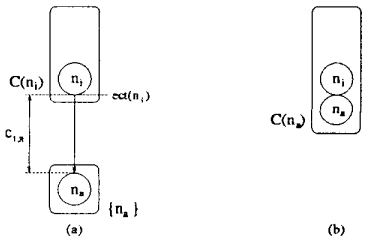

Figure 2. Computation of $C(n_a)$ of task n_a that has the single parent task n_i; (a) before compute $C(n_a)$, (b) compute $C(n_a)$ by merging n_a into $C(n_i)$.

If task n_a has k parent tasks $n_1, n_2, \ldots, n_k$, as shown in Figure 3 (a), where $rdy(C(n_1), n_a) \geq rdy(C(n_2), n_a) \geq \ldots \geq rdy(C(n_k), n_a)$, then $C(n_a) = (\overline{\bigcup}_{i=1}^{l} C(n_i)) \,\overline{\cup}\, \{n_a\}$ such that l is the greatest index that satisfies the following condition for $1 \leq l \leq k$:

$$st_{M_{l-1}}(n_a) \geq st_{M_l}(n_a),$$

where $C(n_{k+1}) = \emptyset$, $M_0 = \emptyset$, $M_j = \overline{\bigcup}_{i=1}^{j} C(n_i)$, and $st_{M_j}(n_a) = \max(ct(M_j), rdy(C(n_{j+1}), n_a))$ for

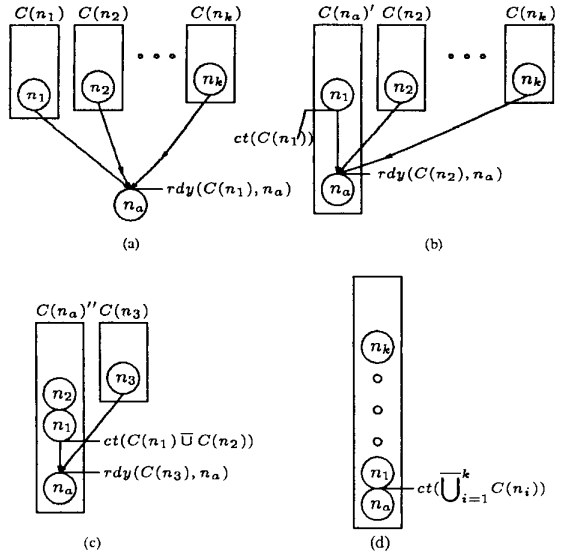

Figure 3. Computation of $C(n_a)$ of task n_a that has multiple parents $n_1, n_2, \ldots, n_k$; (a) join task n_a does not merge with any parent task, (b) n_a is merged into the cluster $C(n_1)$ (c) n_a is merged into $C(n_1) \cup C(n_2)$, and (d) all parent clusters $C(n_1), \ldots, C(n_k)$ are merged.

```
Procedure Build_C_and_est(n_a, C(n_a), est(n_a))
Begin
  // input: n_a, a ready task
  // output: C(n_a), the cluster of n_a
  // output: est(n_a)
  // variable: M_l, clusters that merge parent clusters of n_a
  // side effect: st_{C(n_a)} of tasks in C(n_a) are determined
  // rdy(C(n_1), n_a) ≥ rdy(C(n_2), n_a) ≥ ... ≥ rdy(C(n_k), n_a)
  // rdy(C(n_{k+1}), n_a) = 0 and C(n_{k+1}) = ∅
  M_1 ← C(n_1);                                          1
  l ← 1;                                                 2
  do
    l ← l + 1;                                           3
    M_l ← M_{l-1} ∪ C(n_l);                              4
  while (l ≤ k AND st_{M_{l-1}}(n_a) ≥ st_{M_l}(n_a));
  // l = k + 1 or st_{M_{l-1}}(n_a) < st_{M_l}(n_a)
  C(n_a) ← M_{l-1} ∪ {n_a};                              5
  est(n_a) ← st_{M_{l-1}}(n_a);                          6
End
```

Figure 4. Build_C_and_est() function that constructs cluster and computes *est* of given task

$1 \leq j \leq k$. Therefore, $C(n_a) = M_l \cup \{n_a\}$ and $est(n_a) = \max(ct(M_l), rdy(C(n_{l+1}), n_a))$. Note that the condition is always satisfied in the case of $l = 1$ because $rdy(C(n_1), n_a) \geq \max(ct(C(n_1)), rdy(C(n_2), n_a))$; that is, $st_{M_0}(n_a) \geq st_{M_1}(n_a)$.

Based on this operation, we can always guarantee $st_{C(n_a)}(n_a) \leq \max(ect(n_1), rdy(C(n_2), n_a))$. Hence, our algorithm always produces a better schedule than the TDS algorithm. Figure 4 shows how procedure Build_C_and_est() works. Remember that Build_C_and_est() is applied for ready tasks only. After the sequence of the merge operations, Build_C_and_est() determines which parent clusters are merged with n_a. $C(n_{k+1})$ is assume to be $\emptyset$ in initialization step in order to include the case where merging of all its parent clusters makes $est(n_a)$ minimal.

Let us consider the optimality of the procedure Build_C_and_est(). Since we try to merge the parent clusters from the largest ready time cluster, we must determine whether or not other combinations of merging parent clusters exist, to further reduce the *est* of a join task. Theorem 1 proves this.

Theorem 1 *Consider a join task n_a with k parent tasks $n_1, n_2, \ldots, n_k$, where $rdy(C(n_i), n_a) \geq rdy(C(n_{i+1}), n_a)$ for $i = 1, \ldots, k-1$.* Build_C_and_est() *generates $C(n_a)$ with the smallest $est(n_a)$ among all possible combinations of merging parent clusters. Moreover, we have*

$$est(n_a) = \min_{1 \leq l \leq k} \max(ct(M_l), rdy(C(n_{l+1}), n_a)), \quad (3)$$

where $M_l = \bigcup_{i=1}^{l} C(n_i)$ and $C(n_{k+1}) = \emptyset$.

Proof.
See [9]. □

Theorem 1 indicates that the merging sequence which checks the cluster with the largest ready time task first will be sufficient to obtain $est(n_a)$ in linear time of the number of the parent tasks of each task.

Note that Theorem 1 states only the optimality of the merging sequence in Build_C_and_est(), assuming that the merging operation $\cup$ is optimal, i.e., that the new cluster has the minimum completion time.

Function merge() shown in Figure 5 determines the order of tasks in $M \cup C$ according to their *est* values given clusters M and C. If two tasks with the same *est* value existed in the same cluster, merge() preserves orders in their previous cluster. If two tasks with the same *est* existed in the different clusters, merge() gives a priority to the task in the first cluster M. The last if condition checks whether or not the selected task exists among tasks those are already allocated to M'.

In order to define the optimality condition of merge(), we introduce $P(n_a)$ and $S(n_a)$ for a given task n_a. $P(n_a)$ is the set of ancestor tasks, including parent tasks of n_a; that

```
Function merge(M, C)
Begin
  // Input: M = {n_{m_1}, n_{m_2}, ...}, where est(n_{m_i}) ≤ est(n_{m_{i+1}})
  // Input: C = {n_{c_1}, n_{c_2}, ...}, where est(n_{c_i}) ≤ est(n_{c_{i+1}})
  // Output: M' = {n_{m'_1}, n_{m'_2}, ...}
  i ← 1, j ← 1, k ← 1;
  while (there is any task in M or C) do
    if (n_{m_i} = n_{c_j}) then
      n_{m'_k} ← n_{m_i};
      increase i and j;
    else if (est(n_{m_i}) ≤ est(n_{c_j})) then
      n_{m'_k} ← n_{m_i};
      increase i;
    else
      n_{m'_k} ← n_{c_j};
      increase j;
    end if
    if (n_{m'_k} does not exist among n_{m'_1}, ..., n_{m'_{k-1}}) then
      st_{M'}(n_{m'_k}) ← max[ct_{M'}(n_{m'_{k-1}}), est(n_{m'_k})];
      ct_{M'}(n_{m'_k}) ← st_{M'}(n_{m'_k}) + τ_{m'_k};
      increase k;
    end if
  end while
  return(M');
End
```

Figure 5. merge() **function that merges two clusters** M **and** C.

is,

$$P(n_a) = \bigcup_{n_i \in pred(n_a)} (\{n_i\} \cup P(n_i)),$$

and $S(n_a)$ is the set of all children and descendant tasks of n_a, that is,

$$S(n_a) = \bigcup_{n_j \in succ(n_a)} (\{n_j\} \cup S(n_j)).$$

For example, $P(7) = \{1, 2, 3\}$ and $S(2) = \{6, 7, 8, 9\}$ in the DAG of Figure 8 (a).

Theorem 2 *Given clusters $C(n_1)$, $C(n_2)$, ..., $C(n_l)$, let $M = \bigcup_{i=1}^{l-1} C(n_i)$ and assume that $\bigcup$ is an optimal merge operation, i.e., determines the start times of tasks optimally, and M' =*merge*$(M, C(n_l))$. If the following condition is satisfied, then $ct(M') = ct(M \bigcup C(n_l))$, i.e.,* merge() *produces the minimum completion time that $\bigcup$ does.*

$$\min_{n_e \in M'} \left[ect(n_e) + \max_{n_h \in succ(n_e) \cap M'} \gamma_h \right] \geq ct(M'), \qquad (4)$$

where $\gamma_h = c_{eh} + \tau_h + \sum_{n_j \in S(n_h) \cap M'} \tau_j$.

Proof.

Assume a merged cluster M^o such that

$$ct(M^o) < ct(M'),$$

where M^o and M' are subsets of $\bigcup_{i=1}^{l} (P(n_i) \cup \{n_i\})$. We will show that such M^o cannot exist. Four possible ways exist to transform from M' into M^o:

1. changing the start times of tasks in M',
2. inserting some tasks into M',
3. extracting some tasks from M', or
4. combinations of above methods.

We determine if any of these methods can reduce the completion time of M'.

(1) In order to reduce $ct(M')$ using task position (i.e., the start time of task) changes, there should be at least one empty slot in M', as shown in Figure 6 (a), and assume that n_j is the task positioned directly after the last empty slot. Note that $est(n_j)$ and $est(n_i)$ are greater than a in M', where a is the finish time of the last empty slot in M'. Reducing $ct(M')$ by moving n_i above a is feasible only if $est(n_i) < est(n_j)$. However, $est(n_i) \geq est(n_j)$ is guaranteed by the procedure merge().

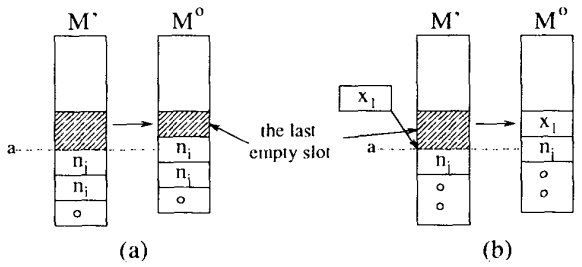

Figure 6. Examples of cluster modification: (a) reduction of completion time by task position change, (b) reduction of completion time by task insertion.

(2) Note that, for this operation to occur, the last empty slot should be generated by some tasks $x_1, \ldots, x_l$ which are not in M', as shown in Figure 6 (b). Assume that n_j is the task positioned directly after the last empty slot. The existence of the empty slot means $est(n_j) = st_{M'}(n_j)$. However, the reduction of $ct(M')$ means that $est(n_j)$ can be further reduced by allocating n_j with $x_1, \ldots, x_l$ in the same cluster, which cannot possibly occur, according to Theorem 1.

(3) Assume that extracting all tasks in $F = \{x_1, \ldots, x_q\}$ from M' will produce an optimal cluster M^o, as shown in Figure 7 (a). Specifically, consider task $x_i \in F$ in Figure 7 (b). The completion time of its successor task n_h in M^o (i.e., $n_h \in succ(x_i) \cap M'$) is at least $ect(x_i) + c_{x_i h} + \tau_h$, where $c_{x_i h}$ is the weight of edge $e_{x_i h}$ between tasks x_i and

n_h. The sum of the execution times of tasks in $S(n_h) \cap M'$ is $\sum_{n_j \in S(n_h) \cap M'} \tau_j$. Thus, the completion time of the last task in $S(n_h) \cap M'$ is at least

$$\alpha_h = ect(x_i) + c_{x_i h} + \tau_h + \sum_{n_j \in S(n_h) \cap M'} \tau_j.$$

Since $|succ(x_i) \cap M'| \geq 1$, the completion time of the last task in $S(x_i) \cap M'$ is at least

$$\begin{aligned} \beta_i &= \max_{n_h \in succ(x_i) \cap M'} \alpha_h \\ &= ect(x_i) + \max_{n_h \in succ(x_i) \cap M'} (c_{x_i h} + \tau_h + \sum_{n_j \in S(n_h) \cap M'} \tau_j). \end{aligned}$$

Since Equation 4 is assumed to be valid,

$$\max_{x \in S(x_i) \cap M'} ct_{M^o}(x) \geq \beta_i \geq ct(M').$$

Next, we consider the case that all tasks in F are extracted. Since $M^o \supset (S(x_1) \cup \ldots \cup S(x_q)) \cap M'$,

$$\begin{aligned} ct(M^o) &= \max_{x \in M^o} ct_{M^o}(x) \\ &\geq \max_{x \in (S(x_1) \cup \ldots \cup S(x_q)) \cap M'} ct_{M^o}(x) \\ &\geq \max_{x \in S(x_i) \cap M'} ct_{M^o}(x) \qquad \text{for any } 1 \leq i \leq q \\ &\geq ct(M'), \end{aligned}$$

which is a contradiction.

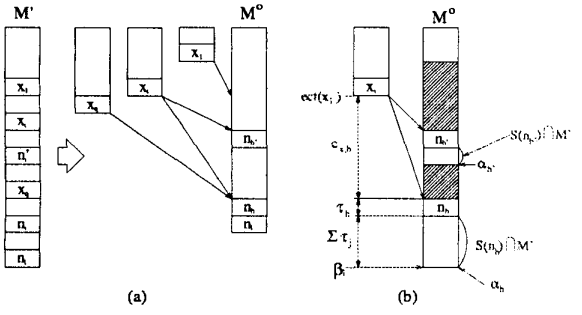

Figure 7. An example of task extraction when two clusters are merged: (a) before extraction, (b) after extraction.

(4) Let the combination of the two methods A and B be applied to M', with M^o generated as a result. The order of methods does not affect M^o. If the modification is $M' \xrightarrow{A} M_1 \xrightarrow{B} M^o$, then $ct(M') \leq ct(M_1)$ and $ct(M_1) \leq ct(M^o)$. Thus, any combination of the above three methods cannot reduce $ct(M')$. □

Theorem 2 explains that if the communication time is too large to extract any task from the current cluster, the function merge() will merge clusters with the minimum completion time.

Procedure schedule() works like the breath-first search, thus the time complexity of ready task search is $O(|E| + |V|)$. For each task n_i, $est(n_i)$ is computed, thus the computation of $est()$ runs $|V|$ times. The time complexity of est is based on procedure Build_C_and_est(). Let d be the maximum degree of a task, then there can be d parent tasks for a join task. Since the size of each parent cluster C_i is $O(|V|)$, the time complexity of merge() is $O(2|V|)$ and that of merge steps $\overline{\cup}_{i=1}^{d} C_i$ is $O(d|V|)$ for a join task. There are d cost comparisons, so $O(|V|d^2)$ is required for a computation of est. After all, $O(|V|^2 d^2)$ is the total time complexity of the algorithm.

3. An Illustrative Example

For the DAG given in Figure 8 (a), Figure 8 (b)-(e) shows how our scheduling algorithm executes. First, $est(0)$ is set as 0. The values of $est(2)$, $est(3)$, $est(4)$, $est(5)$, and $est(6)$ are easily computed, that is, $C(2), C(3), C(4), C(5)$, and $C(6)$ are easily constructed, because each task has only one parent task. Since task '7' is a join task with two parent tasks, its parent clusters $C(2)$ and $C(3)$ are considered. If task '7' merges with $C(2)$, $est(7)$ becomes 17 because of $rdy(C(3), 7) = 17$. $est(7)$ is decreased to 8 by merging '7' into $C(2) \overline{\cup} C(3)$. Because there is no task to be extracted from cluster $C(2) \overline{\cup} C(3)$, the cluster satisfies Equation 4, thus $est(7)$ is optimal and $C(7) = \{1, 2, 3, 7\}$. The TDS algorithm does not generate optimal $est(7)$, because the join task '7' does not satisfy any condition of the TDS algorithm. $est(8)$ is determined as 10 and $C(8) = \{1, 5, 2, 8\}$. Figure 8 (d) shows the merge process of the exit task '9'. The parent clusters are ordered as $C(7), C(8)$, and $C(6)$ by the ready times. When task '9' merges into $C(7)$, $est(9)$ becomes 24 because of $rdy(C(8), 9) = 24$. After '9' merges with $C(7) \overline{\cup} C(8)$, $est(9)$ becomes 21 because of $ct_{C(9)}(8) = 21$. The merge with $C(7) \overline{\cup} C(8) \overline{\cup} C(6)$ is not tried because $rdy(C(6), 9) = 19 < 21 = ct_{C(9)}(8)$. Figure 8 (c) shows that extraction of any clusters from $C(9)$ does not decrease the start time of task '9', which says that Equation 4 is satisfied. Table 1 (b) shows cluster and $est()$ of each task by our scheduling algorithm.

Table 1 (a) shows the est of each tasks generated by the TDS algorithm. Note that the optimal condition for the TDS algorithm is not satisfied in task '7'. Task '7' merges only with task '2' or '3'. Thus $est(7)$ becomes larger than when it merges with '2' and '3'. Note that the start times of task '7' and '9' by the TDS algorithm are greater than those by our algorithm.

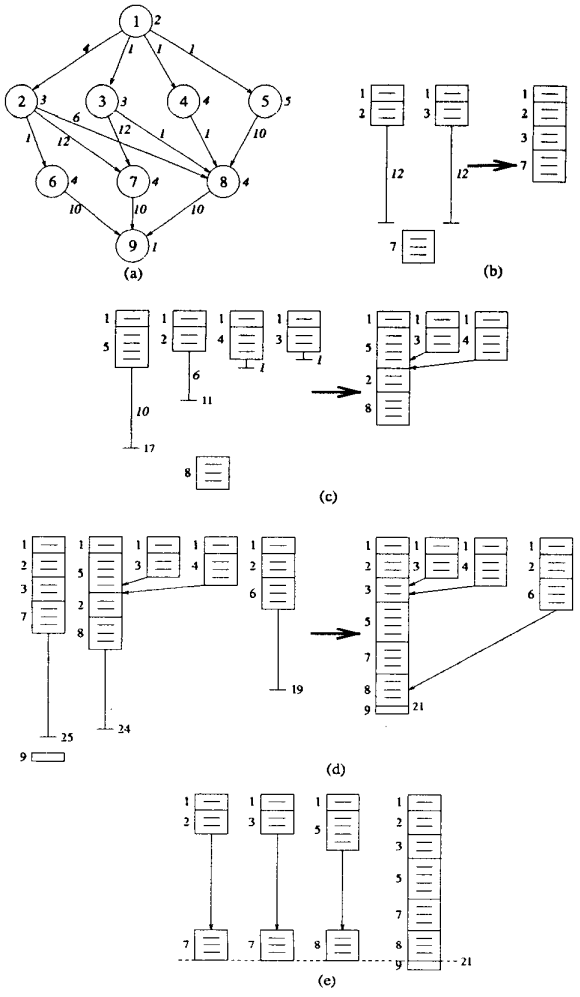

Figure 8. An illustrative example of the proposed algorithm (an optimal schedule)

Table 1. (a) the TDS algorithm [7], and (b) our algorithm

task	est	ect
1	0	2
2	2	5
3	2	5
4	2	6
5	2	7
6	5	9
7	17	21
8	10	14
9	24	25

(a)

task	cluster	est
1	{1}	0
2	{1,2}	2
3	{1,3}	2
4	{1,4}	2
5	{1,5}	2
6	{1,2,6}	5
7	{1,2,3,7}	8
8	{1,5,2,8}	10
9	{1,2,3,5,7,8,9}	21

(b)

4. Conclusions

We proposed an optimal scheduling algorithm based on task duplication. The optimality of the algorithm is guaranteed if the communication overhead is relatively high, that is, Equation 4 is met. The optimal condition of our algorithm extends that of the TDS algorithm in [7]. The proposed algorithm has $O(|V|^2 d^2)$ time complexity where n is the number of tasks, and d is the maximum degree.

References

[1] M. A. Palis, J.-C. Liou, and D. S. L. Wei, "Task clustering and scheduling for distributed memory parallel architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 46–55, January 1996.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, ch. A5.2, pp. 238–241. W.H. Freeman and Company, 1979.

[3] 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*, no. 5, pp. 287–326, 1979.

[4] C. H. Papadimitriou and M. Yannakakis, "Towards an architecture-independent analysis of parallel algorithms," *SIAM Journal on Computing*, vol. 19, pp. 322–328, April 1990.

[5] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *Journal of Parallel and Distributed Computing*, no. 16, pp. 276–291, 1992.

[6] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," *IEEE Software*, pp. 23–32, January 1988.

[7] S. Darbha and D. P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 87–95, January 1998.

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

[9] C.-I. Park and T.-Y. Choe, "An optimal scheduling algorithm based on task duplication." Technical Report, Dept. of CSE, POSTECH, August 2000.