Pergamon

Computers Math. Applic. Vol. 28, No. 4, pp. 85–92, 1994

0898-1221/94 $7.00 + 0.00

0898-1221(94)00129-4

Scheduling Algorithm for Nonpreemptive Multiprocessor Tasks

J.-F. LIN AND S.-J. CHEN
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 106, R.O.C.

(Received May 1993; accepted June 1993)

Abstract—This paper considers the problem of scheduling nonpreemptive multiprocessor tasks in a homogeneous system of processors. The problem proposed in this paper is different from the conventional scheduling problem, where each task requires only "one" processor whenever it is in processing. In our multiprocessor task scheduling problem, tasks can require more than one processor at a time for their processing, that is, tasks in our problem require simultaneously "j" arbitrary processors when they start to be processed, where $1 \leq j \leq k$ and k is a fixed number. Though multiprocessor tasks considered in this paper are assumed to be independent, i.e., the precedence relation does not exist among them, such a problem of scheduling nonpreemptive multiprocessor tasks is NP-complete. Therefore, a heuristic algorithm is investigated for this kind of problem and the worst performance bounds of the algorithm are also derived.

Keywords—Multiprocessor task, Largest processing time first scheduling rule, Performance bound.

1. INTRODUCTION

An assumption in conventional scheduling problems [1–3] is that each task is processed on only "one processor" at a time. The assumption was modified by Blazewicz, Drabowski, and Weglarz [4,5] such that different tasks are to be processed on a "different number of processors." They considered a set $\mathbb{T}$ of n independent tasks which are to be processed on a set of m identical processors $P = \{P_1, P_2, \ldots, P_m\}$. This set of n independent tasks, $\mathbb{T}$, can be decomposed into k subsets, such that $\mathbb{T} = T^1 \cup T^2 \cup \cdots \cup T^k$ where $T^1 = \{T_1^1, T_2^1, \ldots, T_{n_1}^1\}$, $T^2 = \{T_1^2, T_2^2, \ldots, T_{n_2}^2\}$, $\ldots$, $T^k = \{T_1^k, T_2^k, \ldots, T_{n_k}^k\}$ and $n = n_1 + n_2 + \cdots + n_k$. That is, each task in T^1 requires one arbitrary processor for its processing and each task in T^j requires j arbitrary processors simultaneously for its processing. In summary, for a task T_i^j, $i = 1, 2, \ldots, n_j$, exactly j arbitrary processors (here, j can be 1 to k) are allocated to it during a period of t_i^j, which is the computation time of task T_i^j. Note that in the defined multiprocessor task scheduling problem, a schedule is feasible if each task T_i^j can be exactly processed by j processors simultaneously, and the performance of a feasible schedule is measured by its schedule length such that a feasible schedule is called an optimal schedule if it is of minimum schedule length.

In solving the above problem, polynomial time algorithms were proposed by Blazewicz, Drabowski, and Weglarz [5]. But these optimal algorithms were only designed for two constrained cases of independent multiprocessor task schedulings: one case was that multiprocessor tasks are with "unit" computation time; the other one was that multiprocessor tasks are with arbitrary computation time, but they need to be "preemptive."

In this paper, we will pay attention to the problem of scheduling independent "nonpreemptive" multiprocessor tasks with "arbitrary" computation times, which is NP-complete because the scheduling of the independent nonpreemptive tasks with arbitrary computation time of T^1, a special case of this problem, was already known to be NP-complete [1,3]. Therefore, we are interested in polynomial time approximation algorithms, that is, we try to get an approximate solution using heuristic algorithms. We say that a heuristic algorithm has a performance bound β if $(S_H/S_O) \leq \beta$ for all problem instances, where S_H and S_O denote the heuristic schedule length and the optimal schedule length, respectively.

Remember that the Largest Processing Time first scheduling rule (LPT) was a well-known rule for the conventional scheduling of independent tasks nonpreemptively on identical processors. In other words, the LPT is a rule for scheduling those independent nonpreemptive tasks T^1 in a homogeneous system of processors. The principle of LPT is that whenever "a" processor becomes free, select a task whose computation time is the largest of those not yet assigned tasks to assign to it. Graham analyzed [2] that the ratio of the LPT and the optimal schedule lengths is bounded by $(\frac{4}{3} - \frac{1}{3m})$, where m is the number of processors in a system. Based on the principle of LPT, we develop a heuristic algorithm for the problem of scheduling independent nonpreemptive multiprocessor tasks and derive the performance bounds for two cases of this algorithm.

The remainder of this paper is organized as follows: The basic concept of Largest-Processing-Time first scheduling rule is presented and based on this LPT rule, the algorithm and the performance bound of scheduling one set of independent nonpreemptive multiprocessor tasks, say T^j, are derived in Section 2. Section 3 illustrates how the derived algorithm can be extended to schedule the complete task set $\mathbb{T}$. The performance bound of this heuristic algorithm is proved to be bounded by $\left(\frac{4}{3}k - \lfloor k(k+1)/6m \rfloor\right)$ for the case in which both the number of processors in a system and the number of processors required by each task are arbitrary. Furthermore, the performance bound for a special case, in which both of the number of processors in a system and the number of processors required by each task are power of 2, is shown to become $\left(2 - \frac{1}{m}\right)$. Finally, some concluding remarks are given in Section 4.

2. THE LARGEST PROCESSING TIME FIRST SCHEDULING RULE

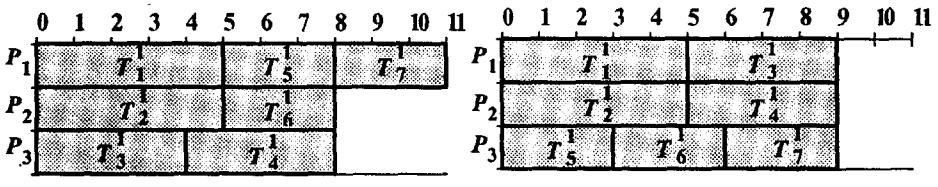
The largest processing time first scheduling rule (LPT) is a popular heuristic algorithm for the conventional nonpreemptive scheduling of independent tasks with arbitrary computation times. This LPT rule was first proposed by Graham in [2] to nonpreemptively schedule independent tasks which require only "one" processor whenever they are in processing. A formal description of LPT is given as follows:

ALGORITHM LPT.
Begin
Input task set T^1;
While (there exist tasks unassigned in T^1) **do**
    Begin
      If (there exists a free processor $P_\alpha, \alpha = 1, 2, \cdots = 1, 2, \ldots, m$) **then**
        Begin
          Assign task T_i^1 whose computation time is the largest among those tasks in T^1 to processor P_α; Remove task T_i^1 from T^1;
        Endif
    EndWhile
End

From the above algorithm, it is obvious that tasks have to be sorted in nonincreasing order of their computation times. Then, tasks are assigned, according to this sorting sequence, to the first free processor they meet. Graham has shown that the performance bound of LPT is $\left(\frac{4}{3} - \frac{1}{3m}\right)$.

That is, let S_H denote the schedule length of a corresponding LPT schedule and S_O denote the schedule length of an optimal schedule. Then, $(S_H/S_O) \leq \left(\frac{4}{3} - \frac{1}{3m}\right)$, where m is the number of processors in a system. A simple example will be given to illustrate the above result.

EXAMPLE 1. Assume that there are 7 tasks $T_1^1, T_2^1, T_3^1, T_4^1, T_5^1, T_6^1$, and T_7^1 submitted to a system of 3 identical processors. The computation times of these tasks are 5, 5, 4, 4, 3, 3, and 3, respectively. Figures 1 (a) and (b) illustrate the LPT schedule and an optimal schedule, respectively. From the figure, we observe that $\frac{S_H}{S_O} = \frac{11}{9} = \frac{4}{3} - \frac{1}{3\times 3}$. ∎



(a) (b)

Figure 1. The LPT schedule and an optimal schedule.

In the following, we will apply the concept of LPT [2] to the nonpreemptive scheduling of independent mutiprocessor task set T^j, where each task in T^j requires exactly j arbitrary processors for its processing, and derive its performance bound.

THEOREM 1. *The performance bound of scheduling one set of independent nonpreemptive multiprocessor tasks* $T^j = \{T_1^j, T_2^j, \ldots, T_n^j, T_{nj}^j\}$, *based on the rule of LPT, on* m *identical processor systems is bounded by* $\left(\frac{4}{3} - \frac{1}{3c_j}\right)$, *where* $j \leq m$ *and* $c_j = \left\lfloor \frac{m}{j} \right\rfloor$.

PROOF. Assume that the independent nonpreemptive multiprocessor tasks in $T^j = \{T_1^j, T_2^j, \ldots, T_{n_j}^j\}$ have been sorted in nonincreasing order of their computation times. By using the LPT rule, tasks in $T^j = \{T_1^j, T_2^j, \ldots, T_{n_j}^j\}$ are assigned to a system of m identical processors. Let S_H^j and S_O^j denote the LPT schedule length and the optimal schedule length of the task subset T^j, respectively.

We can assume that $m = (c_j \times j) + r_j$, where c_j and r_j are positive integers. If $r_j \neq 0$, then the m-processor system can be seen as a $(c_j \times j)$-processor system, because the number of processors can be used by the set of task T^j is $(c_j \times j)$ and there would always exist r_j free processors. Thus, we can consider this system as a $(c_j \times j)$-processor or an $(m - r_j)$-processor system.

(1) If $S_H^j = S_O^j$, then it is obvious that the theorem holds.

(2) For the case that $S_H^j > S_O^j$, there exists a largest integer x_j, such that the x_j longest tasks of the task set T^j are assigned to $(c_j \times j)$ identical processors by the LPT scheduling rule with an optimal schedule. Then, if the longest computation time of the remaining $(n_j - x_j)$ unassigned tasks is $\tau = \max_{x_j < i \leq n_j} \{t_i^j\}$, where t_i^j is the computation time of task T_i^j, it is obvious that the above $(c_j \times j)$ processors cannot be idle before time $(S_H^j - \tau)$.

Hence,

$$j \sum_{i=1}^{n_j} t_i^j \geq (m - r_j) \times (S_H^j - \tau) + (j \times \tau). \tag{2.1}$$

From equation (2.1), we have equation (2.2):

$$S_O^j \geq \frac{j}{(m - r_j)} \sum_{i=1}^{n_j} t_i^j \geq S_H^j - \left(\frac{m - r_j - j}{m - r_j}\right) \tau. \tag{2.2}$$

With $(m - r_j) = (c_j \times j)$, and from equation (2.2), we derive equation (2.3):

$$S_O^j \geq S_H^j - \left(\frac{c_j - 1}{c_j}\right)\tau. \tag{2.3}$$

Since there are at least $(x_j + 1)$ tasks whose computation time are greater than or equal to τ, the $(m - r_j)$ processors must be assigned with at least $\left(1 + \left\lceil \frac{x_j}{c_j} \right\rceil\right)$ of these $(x_j + 1)$ longest tasks. This implies:

$$S_O^j \geq \left(1 + \left\lceil \frac{x_j}{c_j} \right\rceil\right)\tau \tag{2.4}$$

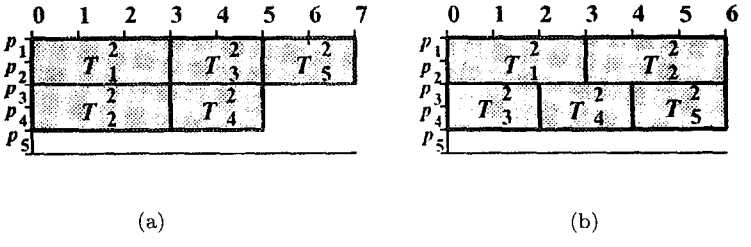
From equation (2.3) and (2.4), we can derive equation (2.5) which is the performance bound equation of the task set T^j on an m identical processor system.

$$\frac{S_H^j}{S_O^j} \leq \left(1 + \frac{1 - \frac{1}{c_j}}{1 + \left\lceil \frac{x_j}{c_j} \right\rceil}\right). \tag{2.5}$$

Since x_j must be greater than c_j (otherwise it would be an optimal schedule), the worst case occurs when $\lceil (x_j)/(c_j) \rceil = 2$. Therefore, the worst performance bound is $\left(\frac{4}{3} - \frac{1}{3c_j}\right)$. ∎

The following simple example will show that the above performance bound is tight.

EXAMPLE 2. Consider a system in which there are 5 identical processors, $m = 5$, and a set of tasks $T^2 = \{T_1^2, T_2^2, T_3^2, T_4^2, T_5^2\}$ are submitted to this system. The computation times of $T_1^2, T_2^2, T_3^2, T_4^2$, and T_5^2 are 3, 3, 2, 2, and 2, respectively. The LPT schedule and an optimal schedule are illustrated in Figures 2 (a) and (b), respectively. Since $m = 5$ and $j = 2$, we can calculate that $c_2 = \lfloor \frac{5}{2} \rfloor = 2$. Thus, we conclude that the performance bound $(S_H^2)/(S_O^2) = \frac{7}{6} = \left(\frac{4}{3} - \frac{1}{3\times 2}\right)$ is tight. ∎



(a) (b)

Figure 2. The LPT schedule and an optimal schedule of task set T^2.

3. HEURISTIC ALGORITHM

In this section, we will propose a heuristic algorithm for the problem of scheduling independent nonpreemptive multiprocessor tasks which is based on the concept of LPT rule. Before giving a formal description of our heuristic algorithm, the multiprocessor task scheduling problem is stated as follows.

Consider a set of n independent nonpreemptive tasks, $\mathbb{T}$, to be processed on a set of m identical processors $P = \{P_1, P_2, \ldots, P_m\}$. Suppose that this set of n independent multiprocessor tasks can be decomposed into k task subsets $T^1 = \{T_1^1, T_2^1, \ldots, T_{n_1}^1\}$, $T^2 = \{T_1^2, T_2^2, \ldots, T_{n_2}^2\}$, $\ldots$, $T^k = \{T_1^k, T_2^k, \ldots, T_{n_k}^k\}$ such that $\mathbb{T} = T^1 \cup T^2 \cup \cdots \cup T^k$, $n = n_1 + n_2 + \cdots + n_k$, and each task T_i^j, $i = 1, 2, \ldots, n_j$, requires "j" arbitrary processors at a time for its processing during a period of time t_i^j, which is the computation time of task T_i^j. Here, tasks are said to be independent which means that there are no precedence constraints among them. Scheduling of tasks is termed nonpreemptive if a task cannot be interrupted once it has begun execution; that is, it must be

allowed to run to completion. The problem of scheduling independent nonpreemptive tasks is an NP-complete problem. In this section, we will devote our attention to developing a heuristic algorithm for such a problem and deriving the performance bounds for two cases of problems: one is that both the number of processors in systems and the number of processors required by each task are arbitrary; the other one is that both the number of processors in systems and the number of processors required by each task are power of 2.

The formal description of our scheduling algorithm is illustrated below:

```
HEURISTIC ALGORITHM.
Begin
 Input task set 𝕋;
 Divide task set 𝕋 into k task subsets T^1, T^2, ..., and T^k;
 time_free = 0      /* Initialize that processors were free at time 0. */
  For j = k downto 1 do
   Sort tasks in T^j in nonincreasing order;
    Begin
     For i = 1 to n_j do      /* Assume that there are n_j tasks in T^j. */
      Begin
       While (there do not exist j free processors at time time_free) do
        Begin
         time_free = time_free + 1;
        EndWhile
       Assign task T_i^j to these j free processors;
      EndFor
    EndFor
End
```

In the above algorithm, the task set $\mathbb{T}$ is first divided into k task subsets $T^1, T^2, \ldots,$ and T^k. Then, the tasks in each task subset are scheduled according to the LPT scheduling rule. The major policy of our algorithm is that the more number of processors a task requires the earlier our scheduling algorithm takes it into consideration.

In the following theorems, we will show that the performance bounds of our heuristic algorithm for the above two cases of problems.

THEOREM 2. *The performance bound of our scheduling algorithms is bounded by* $\left(\frac{4}{3}k - \frac{k(k+1)}{6m}\right)$, *in the case that the number of processors in a system and the number of processors required by each task are arbitrary.*

PROOF. Let S_H and S_O represent, respectively, the heuristic schedule length and the optimal schedule length of the task set $\mathbb{T}$. Since $S_H = S_H^1 \cup S_H^2 \cup \cdots \cup S_H^k$, where S_H^j represents the schedule length of the task subset T^j using the LPT scheduling rule on an m-processor system, for $j = 1, 2, \ldots, k$. Therefore,

$$\begin{aligned}
&\frac{S_H}{S_O} \leq \frac{S_H^1 + S_H^2 + \cdots + S_H^k}{S_O} \\
\Rightarrow &\frac{S_H}{S_O} \leq \frac{S_H^1}{S_O} + \frac{S_H^2}{S_O} + \cdots + \frac{S_H^k}{S_O} \\
\Rightarrow &\frac{S_H}{S_O} \leq \frac{S_H^1}{S_O^1} + \frac{S_H^2}{S_O^2} + \cdots + \frac{S_H^k}{S_O^k},
\end{aligned}$$

because $S_O^j \leq S_O$, where S_O^j represents the optimal schedule length of any task subset T^j on an m-processor system for $j = 1, 2, \ldots, k$.

$$\Rightarrow \frac{S_H}{S_O} \leq \sum_{j=1}^{k} \left(\frac{4}{3} - \frac{1}{3c_j}\right)$$

$$\Rightarrow \frac{S_H}{S_O} \leq \sum_{j=1}^{k} \left(\frac{4}{3} - \frac{j}{3m}\right), \qquad \text{for } c_j = \left\lfloor \frac{m}{j} \right\rfloor$$

$$\Rightarrow \frac{S_H}{S_O} \leq \left(\frac{4}{3}k - \frac{k(k+1)}{6m}\right), \qquad \text{where } k \leq m. \quad \blacksquare$$

In the following, we will consider the case in which both of the number of processors in a system and the number of processors required by each task are power of 2. Such a case can be found in hypercube systems. The performance bound of our heuristic algorithm in this case is shown in Theorem 3.

LEMMA 1. *The number of free processors in a system would always be a multiple of the number of processors required by a task which is the next to be scheduled.*

PROOF. When both of the number of processors in a system and the number of processors required by each task are power of 2, we can assume that the number of processors in the system is $2^{\log_2 m}$ and the task sets are $T^{2^h}, T^{2^{h-1}}, \ldots, T^1$, where $h \leq \log_2 m$. It is clear that if $\left\{2^{\log_2 m} - \sum_{i=u}^{h}(a_i \times 2^i)\right\} > 0$, then it must be a multiple of 2^u, where $0 \leq u \leq h$, and a_i is either a positive integer or a zero. The policy of our algorithm is the more number of processors a task requires, the earlier it will be scheduled. Thus, it is obvious that the lemma holds. ∎

LEMMA 2. *If the task T_x^q is finished at time S_H, then there would be no processors idle before $(S_H - t_x^q)$, where q is an integer of power of 2 and $q \leq m$, $1 \leq x \leq n_q$, t_x^q is the computation time of task T_x^q, and S_H is the schedule length of our heuristic algorithm.*

PROOF. We will prove this lemma by contradiction. By Lemma 1, if there were some processors idle before $(S_H - t_x^q)$, there must be enough free processors for the task T_x^q. Then, T_x^q should be scheduled before $(S_H - t_x^q)$. This leads to a contradiction. Therefore, there would be no processor idle before $(S_H - t_x^q)$. ∎

THEOREM 3. *The performance bound of our heuristic algorithm is bounded by $\left(2 - \frac{1}{m}\right)$, if both the number of processors in a system and the number of processors required by each task are power of 2, where m is the number of processors in a system.*

PROOF. Assume that the task T_x^q is finished at time S_H, where $q \leq k \leq m$, $x \leq n_q$, and S_H is the schedule length of the task set $\mathbb{T}$ by our heuristic algorithm. By Lemma 2, we can get:

$$(m \times S_H) \leq \sum_{j=1}^{k} \sum_{i=1}^{n_j} \left(j \times t_i^j\right) + (m-1) \times t_x^q$$

$$\Rightarrow S_H \leq \frac{\sum_{j=1}^{k} \sum_{i=1}^{n_j} \left(j \times t_i^j\right)}{m} + \frac{(m-1)}{m} \times t_x^q.$$

Since $\left(\left\lfloor \sum_{j=1}^{k} \sum_{i=1}^{n_j} (j \times t_i^j) \right\rfloor / m\right) \leq S_O$ and $t_i^j \leq S_O$, where S_O denotes the optimal schedule length of the task set $\mathbb{T}$, for $j = 1, 2, 4, \ldots, k$, $i = 1, 2, \ldots, n_j$.

$$\Rightarrow S_H \leq S_O + \left(\frac{m-1}{m}\right) S_O$$

$$\Rightarrow \frac{S_H}{S_O} \leq \left(2 - \frac{1}{m}\right). \quad \blacksquare$$

We will give a simple example to show that the performance bound is almost tight.

EXAMPLE 3. Let $t_1^j = m-2$ for $j = 2^1, 2^2, 2^3, \ldots, 2^{\log_2 m - 1}$, $t_i^2 = 1$ for $i = 1, 2, \ldots, m-2$, $t_1^1 = m$, and $t_2^1 = m$. From Figure 3, we observe that the heuristic schedule length $S_H = (2m-2)$ and an optimal schedule length $S_O = m$. Thus, $(S_H/S_O) = (2m - 2/m) = \left(2 - \frac{2}{m}\right)$. ∎

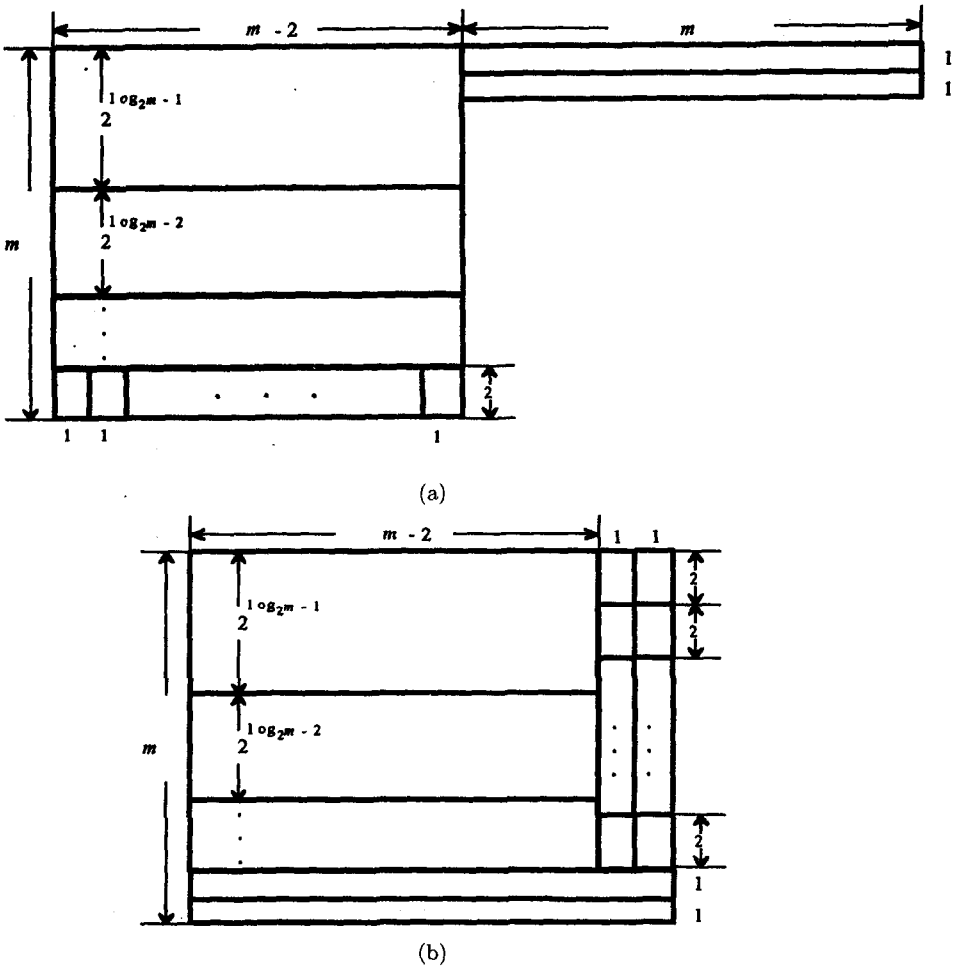


Figure 3. The heuristic schedule and an optimal schedule.

4. CONCLUSIONS

In this paper, we propose a heuristics algorithm for the problem of scheduling independent nonpreemptive multiprocessor tasks and show that its performance bound is bounded by $\left(\frac{4}{3}k - [k(k+1)/6m]\right)$ in case the number of processors in the system and the number of processors required by each task are arbitrary. Besides, we also derive that the performance bound of our scheduling algorithm is bounded by $\left(2 - \frac{1}{m}\right)$, an almost tight performance bound, if both of the number of processors in a system and the number of processors required by each task are power of 2.

Since the performance bound of scheduling nonpreemptive multiprocessor tasks for the case in which the number of processors in a system and the number of processors required by each task are arbitrary is not tight, our future research is to find a tight performance bound. Considering the case in which the precedence constraints among tasks is included will be our next research topic.

REFERENCES

1. E.G. Coffman, Jr., Ed., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, (1976).
2. R.L. Graham, Bounds on Multiprocessing Timing Anomalies, *SIAM J. Appl. Math.* **17**, 416–429 (1969).
3. J.D. Ullman, NP-complete scheduling problem, *Journal of Computer and System Sciences* **10**, 384–393 (1975).
4. J. Blazewicz, M. Drabowski, and J. Weglarz, Scheduling independent 2-processor tasks to minimize schedule length, *Information Processing Letter* **18**, 267–273 (1984).
5. J. Blazewicz, M. Drabowski, and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Trans. on Computers* **C-35** (5), 389–393 (1986).