アルゴリズム 23－17
（1991．9．20）

任意のプロセッサ間結合網をもつ並列マシン上へのタスクグラフの配置手法

川口 剛　　増山 博

（宮崎大学工学部）

あらまし　各リンクが丁度一個のメッセージを記憶できるメモリをもつ直接結合網型マルチプロセッサシステム上への、タスクグラフの静的配置手法を提案する。　評価尺度は、通信時間も含めたタスクの最大完了時間である。　まず、与えられたトラフィック条件の下で、与えられたプロセッサ対に対して、通信遅延最小の経路を求めるための手法を示す。　次に、この経路選択手法を利用するタスクグラフの静的配置手法を示す。

MAPPING A TASK GRAPH ONTO A PARALLEL MACHINE WITH AN ARBITRARY TOPOLOGY

TSUYOSHI KAWAGUCHI and HIROSHI MASUYAMA

Department of Electronics Engineeing

Miyazaki University

Miyazaki, 889-21 Japan

ABSTRACT In this paper we study the problem of scheduling a task graph onto an MPS(message passing multiprocessor system) so as to minimize the total execution time. Each link in the interconnection network of the MPS has a memory which can store one message. An algorithm for finding a route which minimizes the communication delay between processors is first given. Next, we present a heuristic scheduling algorithm which uses the above routing algorithm. The performance of the proposed algorithm is estimated by using simulation experiments.

1. INTRODUCTION

Execution of a parallel program requires a partition of the program into modules and a schedule of these modules on multiprocessors.

In this paper we study the problem of scheduling program modules onto a multiprocessor system so as to minimize the total execution time. This problem is known to be strongly NP-hard even if communication delays between processors are assumed to be infinitely small [1]. Kruatrachue et al. [2] presented two approximation algorithms for the case when all processors are fully connected, that is, all processors can communicate directly with all others without contentions.

The scheduling problem on an MPS(message passing multiprocessor system) is considered in this paper. An MPS is assumed to consist of m homogeneous processors interconnected in an arbitrary way. (The most typical topologies used in MPS's are binary tree, hypercube, mesh, and torus.)

If there exists a data dependency between two modules placed on different processors, communication is needed between these modules. A message between these modules is transmitted along a route on the interconnection network of an MPS. More than one messages cannot pass through a common link of the network at the same time. Thus,

the communication delay between two processors is not always minimized by the shortest path between them.

A routing algorithm is shown in [3]. However, it is assumed in this algorithm that each link of the network has a queue of an infinitely large capacity to store messages waiting for the transmission on the link. Further, although the communication delay associated with a route depends on both the queuing delay and the length of the route, the algorithm estimates the communication delay by only the former.

In this paper, an optimal routing algorithm is presented for the case when each link has a memory which can store one message. The time complexity of this algorithm is $O(e \cdot C)$ where e is the number of links in the network and C denotes the minimum communication delay obtained by this algorithm.

Next we present an approximation algorithm for scheduling program modules onto an MPS with an arbitrary interconnection network topology. The algorithm uses the routing algorithm described above. Simulation experiments showed that, for the most of the test problems, the total execution times obtained by the proposed algorithm were considerably smaller than those obtained by an algorithm in which the communication between each processor-pair is performed by using the shortest path between them.

2. SCHEDULING MODEL

2.1 Task Graph

Given a program P which is already partitioned into modules, the task graph G(P) is a digraph defined as follows.

(1) Each node in G(P), which is called a task, corresponds to a module of P. Weight of a task i denotes the number of instructions to be executed in task i.

(2) G(P) has an arc (i,j) if and only if the processing of task j needs the data of task i. Weight of the arc (i,j) denotes the size of the data to be sent from task i to task j.

2.2 MPS

We make the following assumption about an MPS.

(1) The MPS consists of m homogeneous processors PE(i), $1 \leqq i \leqq m$.

(2) Each processor has an I/O processor and so each processor can execute tasks and communicate with another processor at the same time.

(3) Some pairs of processors are directly connected each other by links. The network constructed by these links is called the interconnection network of the MPS. The link directed from PE(i) toward PE(j) is called link (i,j). Further, if a link (i,j) exists in the interconnection network, we say that PE(j) is adjacent to PE(i).

(4) Let M be the data of task α needed by task β for its processing. If tasks α and β are placed on processors PE(i) and PE(j), respectively, PE(i) is called the sender of M and PE(j) is called the receiver of M. Further, M is called a message to be sent from PE(i) to PE(j). Each message is sent from its sender to its receiver along a route on the interconnection network of the MPS.

(5) Each processor PE(i) has a memory for each entering link (k,i), which stores messages entering to PE(i) through link (k,i). The memory, which is called the memory of link (i,j), cannot store more than one message at a time. If the receiver of a message M stored in the memory of a link (k,i) is PE(i) itself, PE(i) receives the message. Otherwise, PE(i) selects one of its outgoing links according to the information about the route of M. Let (i,j) be the selected link. If the link (i,j) is idle, PE(i) sends M to PE(j) through the link, and otherwise PE(i) keeps M in the memory of link (k,i) until link (i,j) becomes idle.

Fig.1 illustrates the communication mechanism described above. It is assumed in this figure that a processor PE(i) has entering links from $PE(N_i)$, $PE(E_i)$, $PE(W_i)$ and $PE(S_i)$ and has outgoing links to them. SN denotes a switching network.

2.3 System Parameters

The following notation is used in the remainder of

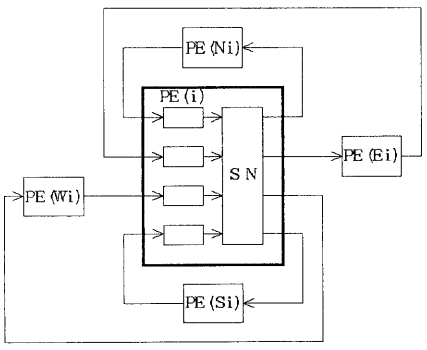

Fig.1 The communication mechanism of the MPS considered in this paper

this paper.

n: the number of tasks,

m: the number of processors,

E: the set of links on the interconnection network,

| E | : the cardinality of E,

INS(i): the number of instructions to be executed in task i,

p(i): the execution time of task i on each processor, which is given by INS(i)/S if the speed of each processor is S,

data(i,j): the data of task i needed by task j,

size(i,j): the size of data(i,j),

R: the transmission rate over a link of the interconnection network,

I: the time to initiate message passing on a link,

d(i,j): the time needed by data(i,j) to pass through a link,

LENGTH(x,y): the length of the shortest path between processors x and y in the interconnection network, where the length of a path denotes the number of links on the path.

p(i) depends on only INS(i) since all processors have the same speed S and p(i) is given by INS(i)/S. Further, d(i,j) depends on only size(i,j) because R and I are both constants and d(i,j) is given by (size(i,j)/R+I). Thus, under an appropriate normalization, we can view p(i) as the weight of task i and d(i,j) as the weight of arc (i,j) in the task graph.

2.4 An Example

As an example, we consider the problem of scheduling the task graph shown in Fig.2 onto the MPS shown in Fig.3. In Fig.2, the number put by the left side of each task i, which is underlined, denotes the parameter p(i) and the number put on each arc (i,j) represents the parameter d(i,j).

First, task T_1 is put in the interval [0,1] on PE(1). T_2 to T_6 need the data of T_1 for their processing. Since T_2 is placed on the same processer as T_1, communication between T_1 and T_2 is not needed. But, T_3 to T_6 have to communicate with T_1 because they are not placed on the same processor as T_1. We assume the routes between T_1 and the tasks T_3 to T_6 are constructed in the order of their indicies.

The data of T_1 is sent to T_3 through link (1,2) and to T_4 through link (1,4). As the result, links (1,2) and (1,4) becomes busy during the interval [1,2].

Since links (1,2) and (1,4) are both busy during the interval [1,2], communication between T_1 and T_5 starts at time 2 and the data of T_1 is sent to T_5 along the route consisting of links (1,4) and (4,5). As the result, the interval [2,3] on link (1,4) and the interval [3,4] on link (4,5) become busy. Repeating the above procedure, we have the schedule shown in

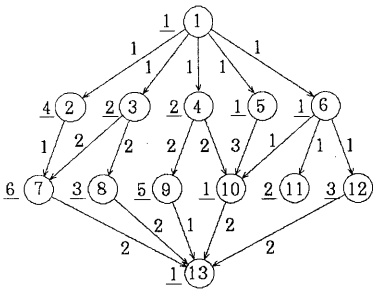

Fig.2 A task graph

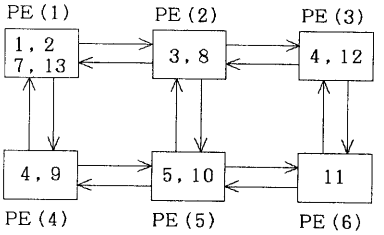

Fig.3 A mapping of the task graph of Fig.2 onto an MPS

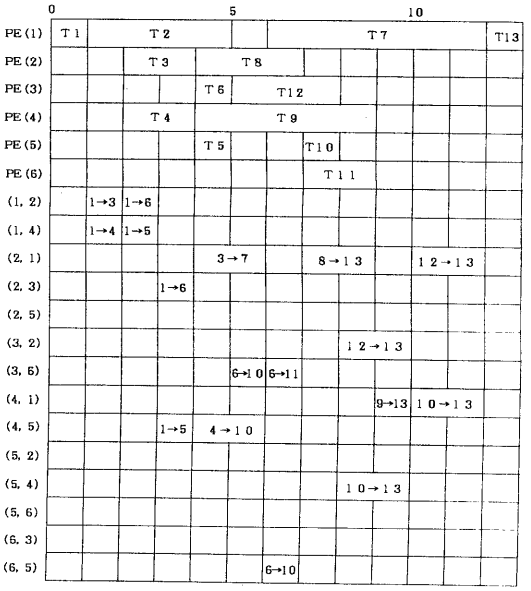

Fig.4 A schedule obtained by the mapping shown in Fig.3

Fig.4.

Note that the communication delay between two processors is not always minimized by the shortest path between them.

3. ROUTING BETWEEN A PROCESSOR-PAIR

In this paper, a route in a network means any sequence of links where the final node of a link is the initial node of the next link. Further, a path is a route which does not use the same node more than once. For example, in the network of Fig.3, a sequence of links ((1,2), (2,5), (5,2), (2,3)) denotes a route. Note that while the shortest route between a processor-pair is always a path, the route which minimizes the communication delay between a processor-pair is not always a path.

3.1 Formulation of the Routing Problem

We assume p(i) and d(i,j) are all integers.(This assumption is valid because the original values of p(i) and d(i,j) can be rounded according to the required accuracy.) Let T denote an upper bound on the length of the schedule obtained by the proposed algorithm.

Assuming that the first task in the schedule starts at time 0, we divide the interval [0,T] into T unit intervals I_t=[t-1,t], $1 \leqq t \leqq T$. Each link $L_k \in E$ is said to be idle during an interval I_t if no messages pass through L_k during I_t, and otherwise L_k is said to be busy during I_t.

For each link $L_k \in E$ and each interval I_t, define TC(k,t) as follows: TC(k,t)=idle if L_k is idle during I_t, and otherwise TC(k,t)=busy. Each TC(k,t) is called a traffic condition of a link L_k during an interval I_t. Further, let TC be a table with $|E|$ rows and T columns whose elements are TC(k,t). TC is called a traffic condition of the network.

Let a message be denoted by M(x,y,d,s) where x is its sender processor, y is its receiver processors, d is the time needed by the message to pass through a link, and s is the time when the transmission of the message becomes ready on processor x.(If M(x,y,d,s) is the data of a task i placed on x which is needed for the processing of a task j placed on y, s is the completion time of task i.) If M(x,y,d,s) is received by y at time t under a traffic condition TC of the network, the difference between t and s is called the communication delay of M(x,y,d,s) under TC, which is denoted by C(TC,x,y,d,s). Further, let a sequence of processors $(v_1,v_2,\cdots,v_{r+1})$, where v_1 is x and v_{r+1} is y, denote a route between x and y in the network. If M(x,y,d,s) can be transmitted from x to y using idle intervals $[t_i,t_{i+1}]$, $1 \leqq i \leqq r$, on links (v_i,v_{i+1}), the sequence $((v_1,t_1),\cdots,(v_{r+1},t_{r+1}))$ is called a transmission pattern for M(x,y,d,s). We consider the following problem in this section.

Problem RF. Given a message M(x,y,d,s) and a traffic condition TC of the network, find a transmission pattern which minimizes the communication delay of M(x,y,d,s) under TC.

Specially, a transmission pattern $((v_1,t_1),\cdots,(v_{r+1},t_{r+1}))$ is said to be a no-wait transmission pattern if $t_{i+1}-t_i=d$ for all i, $1 \leqq i \leqq r$. As described in section 2, each link in the network has a memory to store a message. Thus, if a link (v_i,v_{i+1}) is busy when the message M reaches v_i, M can wait on link (v_{i-1},v_i) until link (v_i,v_{i+1}) becomes idle. Therefore, we need not to impose no-wait conditions on transmission patterns. But, if our final goal is to schedule the whole task graph onto the network, no-wait transmission patterns have a desirable property. For a transmission pattern P, let l(P) denote the sum of lengths of the time intervals used in P. Then, a no-wait transmission pattern P satisfies $l(P) \leqq l(P')$ for any other transmission pattern P' which uses the same route as P. Since no-wait transmission patterns have such a desirable property, we consider the following problem in addition to the problem RF.

Problem NWRF. Given a message M(x,y,d,s) and a traffic condition TC on a network, find a no-wait transmission pattern which minimizes the communication delay of M(x,y,d,s) under TC.

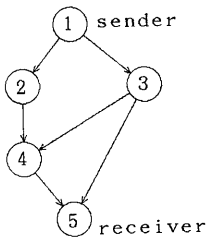

Fig.5 An interconnection network among processors

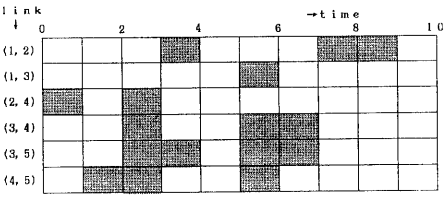

Fig.6 A traffic condition of the network shown in Fig.5

Below, we will explain these problems using an example. Fig.5 shows a network where each node i, $1 \leqq i \leqq 5$, denotes a processor PE(i). Fig.6 gives a traffic condition TC of the network over the time interval [0,10], where the clear segments mean the link is idle and the dark segments denote the link is busy.

Let M be a message to be sent from PE(1) to PE(5) which needs two time units to pass through a link. Further we assume the transmission of M becomes ready at time 0.

The route consisting of links (1,2), (2,4) and (4,5) connects between PE(1) and PE(5). If M departs from PE(1) at time 4, M can pass through each link of this route during its idle interval. Thus, the sequence ((1,4), (2,6), (4,8), (5,10)) is a transmission pattern for M, and the communication delay of M determined by this pattern is 10 time units. Further, this transmission pattern is a no-wait one. We can show that if no-wait conditions are imposed on transmission patterns for M, the minimum communication delay of M is 10 time units. Therefore, the above transmission pattern is an optimal no-wait one.

On the other hand, if M is allowed to wait on link (2,4) after its arrival at PE(4) until link (4,5) becomes idle, M can be placed on links (1,2), (2,4) and (4,5) during the intervals [1,3], [3,6] and [6,8], respectively. Thus, the sequence ((1,1), (2,3), (4,6), (5,8)) is a transmission pattern for M, which is not a no-wait one. Using an optimization algorithm for the problem RF which will be shown later, we can show that the minimum communication delay of M is 8 time units. Therefore, the above transmission pattern is an optimal one.

3.2 An Algorithm for Finding an Optimal No-Wait Transmission Pattern

Let a message be denoted by M(x,y,d,s) where the meanings of x, y, d and s are the same as those previously defined.

Given a processor v in the network, let a sequence of processors $(w_1, w_2, \cdots, w_{r+1})$, where w_1 is x and w_{r+1} is v, denote a route between x and v in the network. For an integer time t, $s \leqq t \leqq T$, if each link (w_i, w_{i+1}), $1 \leqq i \leqq r$, is idle during the interval $[t_i, t_{i+1}]$ where $t_i = t - d(r-i+1)$, the sequence of (w_i, t_i), $1 \leqq i \leqq r+1$, is called a no-wait (v,t) pattern. Further, (w_i, t_i), $1 \leqq i \leqq r$, is called the immediate predecessor of (w_{i+1}, t_{i+1}), respectively. Our objective is to find a no-wait (y,t) pattern with the smallest t.

For each integer time t, $s \leqq t \leqq T$, let V(t) be the set of processors v such that there exist no-wait (v, t) patterns. The smallest t such that $y \in V(t)$ gives the minimum communication delay of M(x,y,d,s) for the no-wait case. If $v \in V(t)$ and a link (v,w) is idle during the interval [t,t+d], we have $w \in V(t+d)$. Further, when the above if-condition holds, we say that a no-wait (v,t) pattern can be extended to a no-wait (w,t+d) pattern. In addition, if P can be extended to Q and Q can be extended to R, we say that P can be extended to R.

A procedure for finding a no-wait (y,t) pattern with the smallest t is shown in Fig.7. The procedure continues to construct the sets V(t), $t \geqq s$, in the increasing order of t until it finds the smallest t such that $y \in V(t)$. If a no-wait (v,t) pattern can be extended to a no-wait (y,t) pattern with the smallest t, all no-wait (v,t) patterns can be extended to it. Therefore, for each $v \in V(t)$, the procedure of Fig.7 keeps only one no-wait (v,t) pattern, which is the pattern earliest found. For each integer time t and each $v \in V(t)$, IMP(v,t) denotes the immediate predecessor of (v,t) in the no-wait (v,t) pattern kept by the procedure.

Let CT denote the communication delay obtained

```
procedure NO_WAIT_ROUTING
{this procedure finds an optimal no-wait
transmission pattern for a message M(x,y,d,s)}
begin
  for t:=s to T do
    insert x into V(t)
    after making V(t) empty;
  t:=s-1;
  FOUND:=false;
  repeat
    t:=t+1;
    while V(t) is not empty do begin
      let v be the first element of V(t);
      delete v from V(t);
      for each link (v,w) directed from v do
        if link (v,w) is idle during the interval
        [t,t+d] then
          begin
            IMP(w,t+d):=(v,t);
            if w=y then
              FOUND:=true;
            else
              if w∈V(t+d) then
                insert w into V(t+d)
          end
    end
  until FOUND=true
end;
```

Fig.7 The procedure NO_WAIT_ROUTING

by the procedure NO_WAIT_ROUTING. Further, using IMP(v,t) obtained by the procedure, let (w_1,t_1)=(y,CT) and (w_{i+1},t_{i+1})=IMP(w_i,t_i) for $1 \leqq i \leqq r$, where w_{r+1} is x. Then, the sequence $((w_{r+1},t_{r+1}),\cdots\cdots,(w_1,t_1))$ is an optimal no-wait transmission pattern.

Theorem 1. The procedure NO_WAIT_ROUTING solves the problem NWRF in time $O(m \cdot d \cdot C)$ where C denotes the communication delay obtained by the procedure.

3.3 An Algorithm for Finding an Optimal Transmission Pattern

As in the last subsection, let a message be denoted by M(x,y,d,s). Given a processor v in the network, let a sequence of processors $(w_1,w_2,\cdots\cdots,w_{r+1})$, where w_1 is x and w_{r+1} is v, denote a route between x and v in the network. Further for an integer time t, $s \leqq t \leqq T$, let t_i, $1 \leqq i \leqq r+1$, be times such that $t_{r+1}=t$ and $t_{i+1}-t_i \geqq d$ for all i, $1 \leqq i \leqq r$. If each link (w_i,w_{i+1}), $1 \leqq i \leqq r$, is idle during the interval $[t_i,t_{i+1}]$, the sequence of (w_i,t_i), $1 \leqq i \leqq r+1$, is called a (v,t)-pattern. Further, (w_i,t_i), $1 \leqq i \leqq r$, is called the immediate predecessor of (w_{i+1},t_{i+1}), respectively. Our objective is to find a (y,t)-pattern with the smallest t.

For each integer time t, $s \leqq t \leqq T$, let V(t) be the set of processors v such that there exist (v,t)-patterns. The smallest t such that $y \in V(t)$ gives the minimum communication delay of M(x,y,d,s).

Below, we consider an extension of a (v,t)-pattern. Given an integer time t and a processor $v \in V(t)$, let P be a (v,t)-pattern and let u denote the first component of the immediate predecessor of the last element (v,t) in P. If a link (v,w) is idle during an interval $[t_0,t_0+d]$, $t_0 \geqq t$, and link (u,v) is idle during the interval $[t,t_0]$, then we have $w \in V(t_0+d)$. Further, when the above two conditions hold, we say that a (v,t)-pattern P can be extended to a (w,t_0+d)-pattern.

As in the no-wait case, even if some processor v of V(t) has more than one (v,t)-patterns, we need only to keep one of such patterns. But, for solving the problem RF, we need to choose a (v,t)-pattern according to a rule, which is shown below.

Let P_j, $1 \leqq j \leqq q$, denote the (v,t)-patterns, and let (u_j,t_j), $1 \leqq j \leqq q$, represent the immediate predecessor of the last element (v,t) in P_j, respectively. Further for each j, $1 \leqq j \leqq q$, define IDLELENGTH$((u_j,v),t)$ as follows: if $[\alpha_j,\beta_j]$ is the longest idle interval on link (u_j,v) which includes time t, IDLELENGTH$((u_j,v),t)=\beta_j-t$. (For example, we have IDLELENGTH((1,2),5)=2 in Fig.6.) For each processor $v \in V(t)$, we keep only a (v,t)-pattern P_j with the greatest IDLELENGTH$((u_j,v),t)$.

```
procedure ROUTING(M(x,y,d,s))
{this procedure finds an optimal transmission pattern
for a message M(x,y,d,s)}
begin
  for t:=s to T do
    insert x into V(t) after making V(t) empty;
  t:=s;
  FOUND:=false;
  repeat
    t:=t+1;
    if y∈V(t) then
      FOUND:=true;
    while V(t) is not empty do begin
      let v be the first element of V(t);
      delete v from V(t);
      for each link (v,w) directed from v do begin
        let t0 be the smallest k(≧t) such that link
        (v,w) is idle during the interval [k,k+d];
        u:=the first component of IMP(v,t);
        if link (u,v) is idle during the interval [t,t0]
        then
          if w∈V(t0+d) then begin
            insert w into V(t0+d);
            IMP(w,t0+d):=(v,t)
          end
          else begin
            u:=the first component of
            IMP(w,t0+d);
            if IDLELENGTH((v,w),t0+d)>
            IDLELENGTH((u,w),t0+d)
            then
              IMP(w,t0+d):=(v,t)
          end
      end
    end
  until FOUND=true
end.
```

Fig.8 The procedure ROUTING

A procedure which finds a (y,t)-pattern with the smallest t is shown in Fig.8.

Theorem 2. The procedure ROUTING solves the problem RF in time $O(e \cdot C)$ where e is the number of links in the network and C denotes the communication delay obtained by the procedure.

4. A SCHEDULING ALGORITHM

In this section we present an approximation algorithm for scheduling a task graph onto an MPS so as to minimize the total execution time. The algorithm

uses the procedure ROUTING shown in the last section.

In addition to the symbols defined in section 2.2, we use the followings.

PROC(i): the index of the processor assigned to task i in some schedule,

F(i): completion time of task i in some schedule.

The scheduling algorithm proposed in this paper is shown in Fig.9.

The function SELECT_TASK(R) returns a task of R with the highest priority. Priority assignment to tasks results in different schedules because tasks are selected in different order. According to [3], priorities of tasks are given by their levels in the graph. The level of a task i in a task graph, which is denoted by level(i), is given as follows: level(i)=p(i) if task i has no successors, and otherwise level(i)=p(i)+maximum of {level(j)+d(i,j)}, where the maximum is taken over all tasks j which are the immediate successors of task i.

For a given task AN, the function SELECT_PROCESSOR selects a processor which can earliest start task AN. Let I_k, $1 \leqq k \leqq r$, denote the indices of the immediate predecessors of task AN in the task graph. Further for each processor PE(j), $1 \leqq j \leqq m$, let ST(j) represent the maximum of $\{F(I_k)+LENGTH(PROC(I_k),j)*d(I_k,AN)\}$ over all I_k, $1 \leqq k \leqq r$, (ST(j) approximates the earliest start time of task AN on PE(j). If all links of the network are idle, ST(j) exactly denote the earliest start time of task AN on PE(j).) The function SELECT_PROCESSOR selects PE(j) with the smallest ST(j).

```
procedure SCHEDULING
begin
  repeat
    let R denote the set of unscheduled tasks
    whose all predecessors have been scheduled;
    AN:=SELECT_TASK(R);
    {the function SELECT_TASK returns a task of R
    with the highest priority}
    PROC(AN):=SELECT_PROCESSOR(AN);
    {the function SELECT_PROCESSOR returns a
    processor on which AN is to be scheduled}
    arrange the parents of AN in nondecreasing
    order of their completion times and let L be the
    obtained list;
    amax(AN):=0;
    while L is nonempty do begin
      v:=the first element of L;
      delete v from L;
      let M(x,y,d,s) denote the data to be sent from
      v to AN where x is PROC(v), y is PROC(AN),
      d is the time needed by the data to pass
      through a link, and s is the completion time
      of v;
      ROUTING(M(x,y,d,s));
      a(v,AN):=the time when M(x,y,d,s) is received
      by AN;
      if amax(AN)<a(v,AN) then
        amax(AN):=a(v,AN)
    end;
    put AN in the interval [amax(AN),amax(AN)+p(AN)]
    on PROC(AN);
  until all tasks are scheduled
end;
```

Fig.9 The procedure SCHEDULING

5. EXPERIMENTAL RESULTS

We performed simulation experiments in order to estimate the performance of the procedure ROUTING shown in FIg.8. The task graph and the MPS shown in [3] were used in these experiments. The task graph has 18 nodes and 36 arcs, and the MPS is a hypercube machine with 8 processors. Further, node weights and arc weights of the task graph were varied throughout the experiments.

Table 1 shows the results of the simulation experiments. Each row of this table corresponds to a problem instance. The columns of this table denote the followings:

$R_{SCH}=(L_{SCH}-LB)/LB$,

$R_{SPT}=(L_{SPT}-LB)/LB$,

where

L_{SCH}: the schedule length obtained by the procedure SCHEDULING, in which the procedure ROUTING is used for finding a communication route between a processor-pair,

Table 1. Performance evaluation of the procedure ROUTING

No.	R_{SCH} [%]	R_{SPT} [%]
1	3.8	11.5
2	6.3	15.6
3	0	5.6
4	5.1	15.4
5	0	11.1
6	3.0	9.1
7	3.8	11.5
8	5.9	11.8
9	3.6	3.6
10	5.9	14.7
Average →	3.7	11.0

L_{SPT}: the schedule length obtained by an algorithm, called SPT, which is the same as the procedure SCHEDULING, except that the communication between a processor-pair is made by using the shortest path between them,

LB: the schedule length obtained by the algorithm SPT on an ideal interconnection network such that an arbitrary number of messages can pass through a common link of the network at the same time.

6. CONCLUSIONS

We presented an approximation algorithm for scheduling a task graph onto an MPS in which each link of the interconnection network has a memory to store one message. A routing algorithm which is used in the above scheduling algorithm always generates an optimal route between any processor-pair under a given traffic condition of the network.

REFERENCES

[1] M.R.Garey and D.S.Johnson, "Computer and Intractability: A Guide to the Theory of NP-Completeness", Freeman, San Francisco (1979).

[2] B.Kruatrachue and T.G.Lewis, "Grain size determination for parallel processing", IEEE Software, 1988, No.1, pp.23-32.

[3] H.E.Rewini and T.G.Lewis, "Scheduling parallel program tasks onto arbitrary target machines", Journal of Parallel and Distributed Computing, Vol.9, No.2, pp.138-153 (1990).