一次元アレイプロセッサにおける タスクマッピング手法

川口　剛　　田村　吉紀

（宮崎大学工学部）

あらまし　一次元アレイは、ＶＬＳＩ化に最も適した並列処理アーキテクチャの一つである。　本論文では、木構造を持つプログラムのモジュールを、プロセッサ間の通信時間も含めた総実行時間が最小となるよう、一次元アレイプロセッサ上へ配置するアルゴリズムを提案する。　シミュレーション実験の結果、提案アルゴリズムによるプログラムの実行時間の、既存のアルゴリズムによるプログラムの実行時間の比は、実験に用いた１２０個のテスト問題の平均として、約７１％であった。

A TASK MAPPING ALGORITHM FOR LINEAR ARRAY PROCESSORS

Tsuyoshi Kawaguchi　　Yoshinori Tamura
Miyazaki University

ABSTRACT The linear array processor architecture is an important class of interconnection structures that are suitable for VLSI. In this paper we present a heuristic algorithm for mapping a task tree onto a linear array to minimize the total execution time. The algorithm partitions the node set of a task tree into clusters and maps these clusters onto processors. Simulation experiments showed that the total execution time of a task tree obtained by the proposed algorithm was about 71% of that obtained by a conventional algorithm on the average of 120 test problems.

1. INTRODUCTION

The mapping of task graphs onto target architectures is an essential subject for the design of VLSI algorithms such as systolic array algorithms. Rewini et al.[1] and Kawaguchi [2] presented heuristic algorithms for mapping an arbitrary task graph onto a parallel machine with an arbitrary interconnection network topology. Koren et al.[3] considered the problem of mapping an arbitrary task graph onto two-dimensional array processors. Cappello [4] proposed a systolic array which can realize a processor-time-minimal mapping for a task graph with cubical mesh structure.

The linear array processor architecture is an important class of interconnection structures that are suitable for VLSI. The simplicity of the linear array interconnection provides several practical advantages over higher dimensional multiprocessor arrays. However, since the diameter of a linear array is proportional to the number of processors, the interprocessor communication grows lineally with the number of processors. Thus, the balancing between the amount of computation assigned to each processor and the amount of interprocessor communications is especially important for the design of VLSI algorithms on linear arrays.

Trees play important roles as computational structures for several

applications. In this paper we study the problem of mapping a task tree onto a linear array so as to minimize the total execution time.

Ghosal et al.[5] presented a heuristic algorithm for this problem. The algorithm partitions the node set of the task tree into clusters and allocates these clusters onto processors of a linear array. Ghosal et al. recommended to use $O(\sqrt{n})$ processors and showed that if $O(\sqrt{n})$ processors are used, the computation time and the communication time obtained by their mapping algorithm are both $O(h' \log n)$ and so the total execution time of a task tree is also $O(h' \log n)$ where $h'=\max(h,\sqrt{n})$ and h denotes the height of the task tree.

We first present an optimization algorithm for a message scheduling problem which occurs in the task tree mapping problem stated above. Solutions of this problem play an essential role to find an efficient mapping of a task tree. However, any algorithm for this problem was not explicitly given by Ghosal et al.[5] because they discussed only a rough estimation of the total execution time of a task tree obtained by their algorithm. Next, we present a new algorithm for mapping a task tree onto a linear array.

2. PRELIMINARIES

2.1 Problem Discriptions

In this paper we consider the problem of mapping a task tree onto a linear array so as to minimize the total execution time. As in Ref[5], we assume the processors P_i, $1 \leq i \leq m$, which are numbered according to the left to right ordering, perform computation subject to the following rules.

(1) Computations on all processors are synchronized and communications between all adjacent processor-pairs are synchronized. In addition, any processor cannot communicate with another processor while executing tasks.

(2) Each processor P_i can execute a task in one unit of time and it can also transmit one value to a neighbor P_{i-1} in one unit of time. (Communication between each processor - pair is one directional.)

(3) Each link (P_i,P_{i-1}) can transmit no more than one value at each time unit.

(4) Each non-leaf task cannot be executed before the values of all its children are available to the processor on which the task is placed.

This problem is called MAP_TREE in the remainder of this paper. For each task v, let I(v) denote the index of the processor on which task v is placed. Then, any feasible solution of the problem MAP_TREE has to satisfy the following condition: if a task v is a parent of a task w, then $I(v) \leq I(w)$.

2.2 A Mapping Algorithm

Ghosal et al.[5] presented an algorithm for the problem MAP_TREE. The algorithm consists of the following three steps.

(I) Partition the node set of the task tree T into several clusters $G_1,\cdots,G_r$. Let TC denote the tree obtained from T by combining the nodes of each G_k $1 \leq k \leq r$, into one node. We call TC the cluster tree associated with T and $(G_1,\cdots,G_r)$.

(II) Map clusters G_k, $1 \leq k \leq r$, onto processors $P_1,\cdots,P_m$ of a linear array where P_i, $1 \leq i \leq m$, are numbered according to the left to right ordering.

(III) Let level(G_k) denote the levels of nodes G_k in the cluster tree TC, and l_{max}=maximum of level (G_k), $1 \leq k \leq r$.. (If the distance between a node G_k and the root of TC is j, the level of G_k is (j+1).)

For j=l_{max} down to 1 do

(1) In parallel, evaluate all clusters G_k with level (G_k)=j.

(2) For each evaluated cluster G_k, let PR(G_k) denote the parent of G_k in the cluster tree TC. Transmit the output data of G_k to the processor on which PR(G_k) is placed.

Let $T_E(j)$, $1 \leq j \leq l_{max}$, denote the time needed to evaluate clusters at level j and let $T_C(j)$, $1 \leq j \leq l_{max}$, represent the time

needed to transmit the output data of clusters at level j. Then, the execution time of the task tree is given by:

$$T = \Sigma(T_B(j)+T_C(j)) \qquad (1)$$

where the summation is taken over all levels j, $1 \leq j \leq l_{max}$.

Below, we show the procedure used in Ref.[5] for partitioning the node set of the task tree into clusters. This procedure is recursively executed from the root of the tree T. For each node v in T, let ST(v) denote the subtree with root v. Any subtree ST(v) has a node α whose removal disconnects ST(v) into smaller trees such that the number of nodes in each smaller tree is smaller than or equal to $|ST(v)|/2$. The node α is referred to as a centroid of ST(v).

To find a centroid of ST(v) , we can use the following method. Start from the root v. If v is a centroid of ST(v), we are done. Otherwise, move to a child of v whose subtree contains more than $|ST(v)|/2$ nodes. If the current node is a centroid, we are done. Otherwise, repeat this search until a centroid is found.

<< The procedure PARTITION (v)>>

If the number of nodes in ST(v) is smaller than or equal to $h'=\max(h, \lceil n/m \rceil)$ where h is the height of the tree T, make the node set of ST(v) a cluster. Otherwise, let α be a centroid of ST(v). If P represents the path connecting v and α, make the node set of P a cluster. Further, let $ST(v_i)$, $1 \leq i \leq r$, denote the subtrees obtained by the removal of the nodes lying on P. For each node v_i, $1 \leq i \leq r$, apply the procedure PARTITION(v_i).

<< The procedure CLUSTER_MAPPING >>

The procedure used in Ref.[5] for mapping clusters onto processors $P_1, \cdots, P_m$ of a linear array repeats the following operation while there exist unscheduled clusters: after selecting a cluster G_k according to a preorder traversal, assign G_k to the left most processor with a smaller number of tasks than $h'=\max(h, \lceil n/m \rceil)$.

As an example, we consider the problem of mapping the task tree of Fig.1 into three processors. We have h'=4 because n=10, h=4 and m=3. The algorithm of Ref.[5] proceeds as follows.

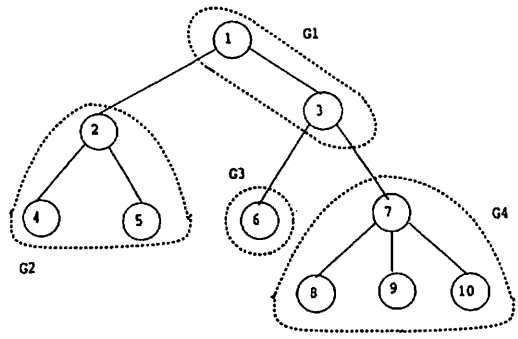

Fig.1 A task tree

(I) Since $|ST(1)| > h'$, the procedure PARTITION divides ST(1) into smaller trees. A centroid of ST(1) is node 3 and so the procedure makes the node set {1,3} a cluster. The removal of {1,3} divides ST(1) into three subtrees ST(2), ST(6) and ST(7) whose sizes are smaller than or equal to h'. Thus, the tree of Fig.1 is finally divided into four clusters : $G_1=\{1,3\}$, $G_2=\{2,4,5\}$, $G_3=\{6\}$, $G_4=\{7,8,9,10\}$.

(II) The procedure CLUSTER_MAPPING assigns G_1 and G_2 to processor P_1 and the remaining clusters to processor P_2.

(III) In the cluster tree, $level(G_i)=1$ and $level(G_i)=2$ for $2 \leq i \leq 4$. Thus, when j=2, G_2 is executed on P_1 and G_3 and G_4 are processed on P_2. The execution time is 5 time units. Next, the output data of G_3 and that of G_4 are sent from P_2 to P_1. This communication requires 2 time units. Finally, when j=1, G_1 is executed by P_1 in 2 time units. Thus, the total execution time of the task tree is 9 time units.

3. AN OPTIMIZATION ALGORITHM FOR SCHEDULING MESSAGES

In this section we consider a message scheduling problem which occurs in the problem MAP_TREE. The message

scheduling problem, which we call MSCH, is stated as follows.

<< The problem MSCH >>

Given a linear array N with nodes P_i, $1 \leq i \leq m$, and links (P_i, P_{i-1}), $1 < i \leq m$, and a set of messages M_k, $1 \leq k \leq p$, with senders $S(M_k)$ and receivers $R(M_k)$ such that $I(S(M_k)) > I(R(M_k))$ where $I(v)$, $v = S(M_k)$ and $R(M_k)$, denotes the index of v, schedule M_k, $1 \leq k \leq p$, on the path connecting $S(M_k)$ and $R(M_k)$ to minimize the maximum completion time of M_k, $1 \leq k \leq p$. (The completion time of M_k means the time when M_k reaches its receiver.)

The problem MSCH occurs in the step(III) of the task tree mapping algorithm shown in subsection 2.2. Therefore, algorithms for the problem MSCH are needed to be on-line and parallel. Fig.2 shows a parallel algorithm for the problem MSCH which is

```
procedure MSO
begin
  for all nodes Pi, 1≤i≤m, do parallel
    begin
      make Ω(Pi) empty;
      if Pi has messages Mk whose
      senders are Pi then
        insert these messages Mk into
        Ω(Pi) according to a
        nondecreasing order of I(R(Mk));
    end;
  t:=0;
  while there exist node Pi such that
  Ω(Pi) are nonempty do begin
    for all nodes Pi, 1≤i≤m, do parallel
      begin
        select a message Mk of Ω(Pi)
        with the smallest I(R(Mk));
        place Mk on link (Pi,Pi-1) during
        the time interval [t,t+1];
        delete Mk from Ω(Pi);
        if R(Mk)≠Pi-1 then
          insert Mk into Ω(Pi-1)
      end;
    t:=t+1;
  end;
end;
```

Fig.2 The procedure MSO

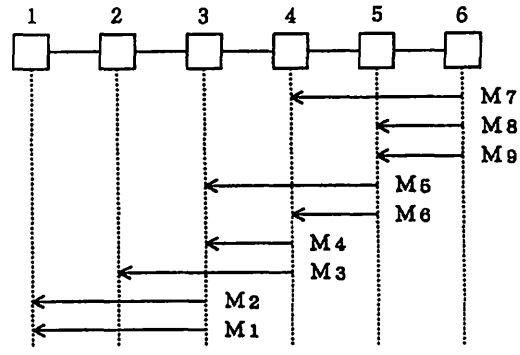

Fig.3 An instance of the problem MSCH

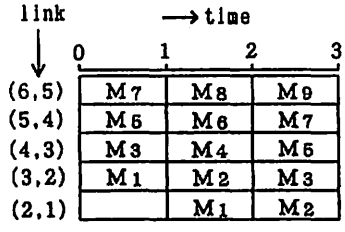

Fig.4 The schedule obtained by applying the procedure MSO to the message set of Fig.3

also on-line. The point of this algorithm is to select a message M_k of $\Omega(P_i)$ with the smallest $I(R(M_k))$ for the transmission on link (P_i, P_{i-1}) when $\Omega(P_i)$ has more than one messages.

If we apply the procedure MSO for the message set of Fig.3, we have the schedule shown in Fig.4.

We have the following theorem.

[Theorem 1] The procedure MSO finds an optimal schedule for any instance of the problem MSCH.

Let T denote the maximum completion time of messages obtained by the procedure MSO. Since T corresponds to the communication time in step(III) of the task tree mapping algorithm, we want an algorithm with time complexity O(T) for the problem MSCH. However, the time complexity of the procedure MSO is $O(T \cdot \log p)$ where p is the number of messages. In addition, to achieve the time complexity of

$O(T \cdot \log p)$, each $\Omega(P_i)$ needs to be implemented by a balanced tree which is a fairly complex data structure.

To tell the truth, any instance $M=\{M_1, \cdots, M_p\}$ of the problem MSCH which occurs in the execution step of the task tree mapping algorithm proposed in Ref.[5] and that proposed in this paper has the following helpful property :

(#) for any message-pair M_i and M_j, if $I(R(M_i))<I(R(M_j)$ then $I(S(M_i))\leqq I(S(M_j))$.

For any instance of the problem MSCH satisfying the above condition, we have the following simple optimization algorithm.

<< The procedure MS1 >>

This procedure is the same as the procedure MS0 except the followings :

(1) each $\Omega(P_i)$ is implemented by a first-in first-out queue instead of a priority queue such as a balanced tree ;

(2) each node P_i selects the first message of $\Omega(P_i)$ for the transmission on link (P_i,P_{i-1}) instead of selecting a message M_k with the smallest $I(R(M_k))$.

The key point of the procedure MS1 is to transmit messages of $\Omega(P_i)$ according to a first-come first-served rule.

Thus, it is clear that the procedure MS1 can be executed in time $O(T)$.

We have the following corollary.

[Corollary 1] The procedure MS1 finds an optimal schedule for any instance $M=\{M_1, \cdots, M_p\}$ of the problem MSCH satisfying the condition (#).

4. MAPPING A TASK TREE ONTO A LINEAR ARRAY

4.1 Partitioning a Task Tree into Clusters

For each level j, $1\leqq j\leqq l_{max}$, in the cluster tree, let $\Omega(j)$ denote the set of clusters at level j and let β_j, $1\leqq j\leqq l_{max}$, denote the maximum of $|G_k|$ over all clusters G_k of $\Omega(j)$. Then, since $T_z(j)\geqq\beta_j$ in Eq.(1), the minimization of $\Sigma\beta_j$ is needed to minimize the execution time of the task tree.

Our algorithm starts from the partition obtained by the algorithm of Ref.[5] and, for j from $(l_{max}-1)$ down to 1, divides some leaf cluster of level j into two parts, one of

```
procedure PARTITION
begin
  partition the node set of the task tree T
  into clusters by using the procedure
  shown in Ref.[5];
  let Ω(j), 1≦j≦l_max, denote the set of
  clusters at level j in the resultant cluster
  tree;
  for j:=l_max-1 down to 1 do
   while | Ω(j+1) | <m do begin
     β:=max{ | G_k | | G_k∈Ω(j+1)};
     select a leaf cluster G_x of Ω(j) with the
     greatest | G_k | ;
     if | G_x | ≦ β then begin
       move G_x from Ω(j) to Ω(j+1);
       make a dummy cluster G_x' with no
       tasks;
       insert G_x' into Ω(j);
     end
     else begin
       let ST(G_x) denote the subtree of the
       task tree T whose node set is G_x;
       for each arc e in ST(G_x) do begin
         let ST_1(e) and ST_2(e), where ST_1(e)
         includes the source of e, denote
         the subtrees of ST(G_x) obtained
         by the removal of e;
         Δ(e):=the number of tasks in ST_1(e);
       end;
       find an arc e of ST(G_x) with the
       greatest Δ(e) such that Δ(e)≦ β;
       let ST_1(e) and ST_2(e), where ST_1(e)
       includes the source of e, denote the
       subtrees of ST(G_x) obtained by the
       removal of e;
       let G_x1 and G_x2 denote the clusters
       corresponding to the node sets of
       ST_1(e) and ST_2(e), respectively;
       after deleting G_x from Ω(j), insert
       G_x1 and G_x2 into Ω(j+1) and Ω(j),
       respectively;
     end;
   end;
```

Fig.5 The procedure PARTITION

which is moved to level (j+1). The division is performed so that the size of the subcluster moved to level (j+1) does not exceed β_{j+1}. The purpose of such division is to reduce β_j without increasing β_{j+1}. The formal description of our partition algorithm is shown in Fig.5 .

The task tree of Fig.6 is the one used by Gohsal et al.[5] for the illustration of their algorithm. Fig.6 also shows the partition of the node set of this tree obtained by our algorithm. On the other hand, the algorithm of Ref.[5] generates the partition obtained by combining each pair of (G_2,G_{16}), (G_3,G_{19}), (G_6,G_{17}) and (G_8,G_{18}) into one cluster. In addition, while the algorithm of Ref.[5] treats G_5 and G_{14} as the clusters at the third level, our algorithm executes these clusters together with the fourth level clusters.

Fig.7 shows the cluster tree obtained from the task tree of Fig.6 by combining the nodes of each cluster into one node. In this tree, nodes 5* and 14* denote dummy nodes with zero processing times. They do not require any computation. But, when the output data of nodes 5 and 14 are sent to nodes 4 and 7, respectively, the data need to pass through 5* and 14*, respectively, because communications are possible only between nodes at adjacent levels.

4.2 Mapping Clusters onto Processors

In the mapping obtained by the algorithm of Ref.[5], more than one nodes at the same level are often assigned to the same processor and it degrades the efficiency of the algorithm.

Our algorithm uses a level by level mapping strategy to avoid such situations. Let $I(G_k)$ denote the index of the processor on which a cluster G_k is placed. Any feasible mapping has to satisfy the following condition :

(*) $I(G_i) \leq I(G_j)$ for a pair of clusters G_i and G_j such that G_i is a parent of G_j in the cluster tree.

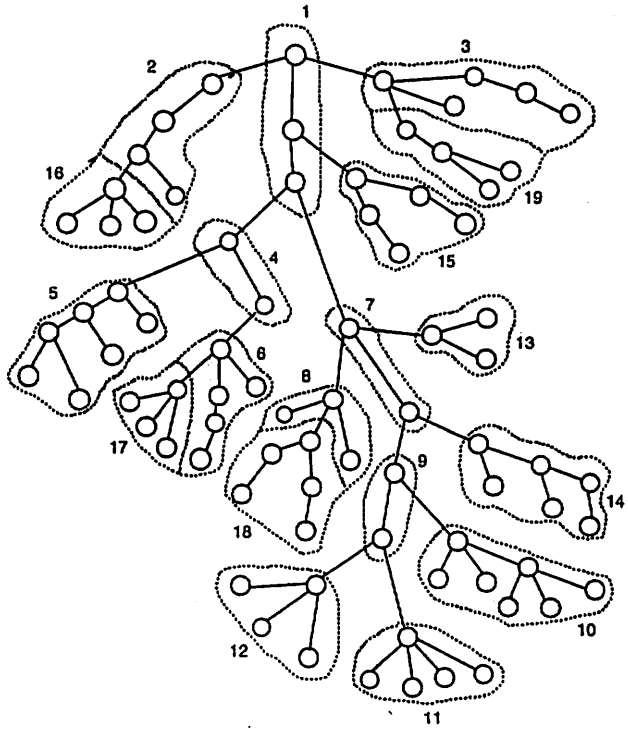

Fig.6 A task tree and a partition of its node set

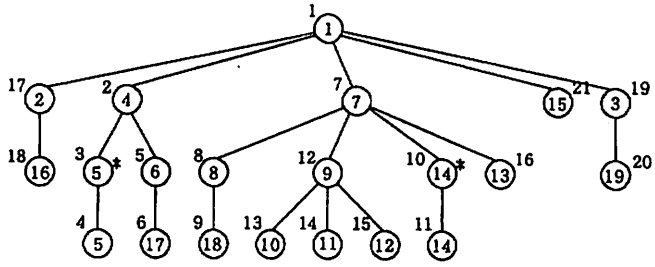

Fig.7 The cluster tree associated with the task tree and the partition shown in Fig.6

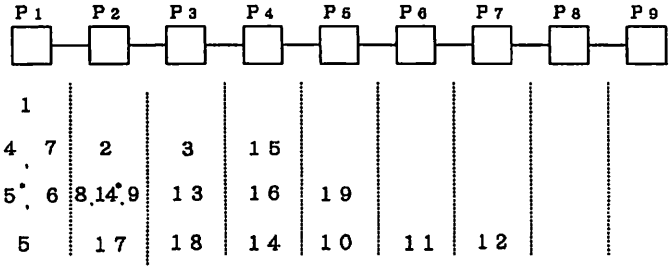

Fig.8 A mapping of the clusters of Fig.7 onto a linear array

The algorithm of Ref.[5] selects clusters according to a preorder traversal on the cluster tree for the assignment of processors. In addition, after some cluster was placed on a processor P_i, none of the remaining clusters are assigned to processors P_j , $1 \leq j < i$. Thus, the above condition always holds in the mapping obtained by the algorithm of Ref.[5] although the algorithm does not explicitly check the condition.

On the other hand, our algorithm repeats the following procedure for each j from l_{max} down to 1, schedule clusters of level j on processors P_i $1 \leq i \leq m$, in such a way that any non-leaf cluster G_i satisfies $I(G_i) \leq I(G_j)$ for all its children G_j.

To minimize the execution time of the task tree, the algorithm first sorts clusters according to a preorder traversal restricted by the following rule.

(*) For each node G_k in the cluster tree, let $SUC(G_k)$ denote the set of successors of G_k with the highest level and let $L(G_k)$ represent that level. Given a node-pair G_i and G_j of the cluster tree with the same parent, G_i precedes G_j if $L(G_i) > L(G_j)$ or $(G_i) = L(G_j)$ and $| SUC(G_i) | \leq | SUC(G_j) |$.

In Fig.7, nodes are labeled according to the above ordering. Fig.8 shows a mapping of the cluster tree onto a linear array with nine processors, which is obtained by the algorithm of this paper. The formal description of the proposed mapping algorithm is given in figures 9 and 10.

The total execution time of the task tree of Fig.6 obtained by our algorithm is 32 time units. On the other hand, the total execution time of the tree obtained by the algorithm of Ref.[5] is 44 time units.

5. PERFORMANCE EVALUATION

We made simulation experiments to evaluate the performance of the mapping algorithm proposed in this paper. For each

```
function PACKING(c)
begin
  i:=1;
  while Ω(j) is nonempty do begin
    let G_k denote the node of Ω(j) with the
    smallest label;
    delete G_k from Ω(j);
    if |P_{i,j}| + |G_k| > k or P_i has a node G_q
    of level j such that the parent of G_q is
    not the same as that of G_k and all
    siblings of G_q are placed on P_i then
     begin
      i:=i+1;
      if i≦m then
        assign G_k to P_i
      else
        return (false)
     end
    else
      assign G_k to P_i
  end
  return(true)
end;
```

Fig.9 The function PACKING

```
procedure CLUSTER_MAPPING
begin
 for j:=l_max down to 1 do begin
   c:=max(max{|G_k| | G_k∈Ω(j)},
   ⌈Σ_{G_k∈Ω(j)} |G_k| /m⌉);
   if PACKING(c)=false then begin
     i:=1;
     while PACKING(2^i*c)=false do
       i:=i+1;
     LB:=2^{i-1}*c+1;
     UB:=2^i*c;
     c:=(LB+UB) div 2;
     while LB<UB do begin
       if PACKING(c)=false then
         LB:=c+1
       else
         UB:=c;
       c:=(LB+UB) div 2;
     end
   end
 end;
```

Fig.10 The procedure CLUSTER_MAPPING

test problem, the execution time of the task tree, T_1, obtained by the proposed algorithm was compared with the execution time of the task tree, T_0, obtained by the algorithm of Ref.[5].

The task graphs used in the experiments were constructed by the following method. Let ϕ_0 denote the task graph of Fig.6. Starting from ϕ_0, construct a new task graph ϕ_{i+1} ($i \geq 0$) by applying the following operation to ϕ_i: after selecting a pair of tasks v and w such that v is neither a predecessor nor a successor of w, exchange their parents (in other words, if x and y are parents of v and w, replace arcs (v,x) and (w,y) by new arcs (v,y) and (w,x)). In addition, the number of processors, m, was varied in the range {5,9,12,15}.

On the average of 30 task graphs, the ratios T_0/T_1 were 1.40 for m=5, 1.39 for m=9, 1.42 for m=12 and 1.42 for m=15.

6. CONCLUSIONS

We presented an algorithm for mapping a task tree onto a linear array. The simulation experiments showed that the algorithm of this paper is much more efficient than the algorithm of Ref.[5].

REFERENCES

[1] 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).

[2] T.Kawaguchi: "Scheduling a task graph onto a message passing multiprocessor system", IEICE Trans., Vol.E75-A, No.6, pp.670-677(1992).

[3] I.Koren, B.Mendelson, I.Peled and G.M.Silberman: "A data-driven VLSI array for arbitrary algorithms", IEEE Computer, Vol.21, No.10, pp.30-43(1988).

[4] P.Cappello: "A processor-time-minimal systolic array for cubical mesh algorithms", IEEE Trans. on Parallel and Distributed Systems, Vol.3, No.1, pp.4-13(1992)

[5] D.Ghosal, A.Mukherjee, R.Thurimella and Y.Yesha: "Mapping a task tree onto a linear array", Proc. 1991 Int. Conf. on Parallel Processing, Vol.1, pp.629-633.