Mapping a Chain Task to Chained Processors

*Y. Han** *B. Narahari*** H-A. Choi**

*Department of Computer Science
University of Kentucky
Lexington, KY 40506

**Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC 20052

Abstract

We present algorithms for computing an optimal mapping of a chain task to chained processors. Our algorithm has time complexity no larger than $O(m + p^{1+\epsilon})$ for any small $\epsilon > 0$. This represents an improvement over the recent best results of an $O(mp)$ algorithm and an $O(m+p^2 \log^2 m)$ algorithm. We extend our algorithm to the case of mapping a ring task to a ring of processors with time complexity $O(mp^{\epsilon})$ and the case of mapping multiple chain tasks to chained processors with time complexity $O(mn + p^{1+\epsilon})$.

Keywords: Parallel computation, task mapping, task allocation, task scheduling, linear processor array.

1 Introduction

In parallel computation it is often desired to map a computation task to a parallel machine. A computation task is usually modeled as a weighted graph where nodes in the graph are task modules and edges represent communication requirement among task modules. Each node in the graph is labeled with a number indicating the execution time (by a single processor) of the task module. Each edge is also labeled with a number indicating the communication time required for the intercommunication

between the two task modules incident with the edge. The objective sought is a mapping which minimizes the parallel execution time of the computation task.

The general mapping problem is known to be intractable[1]. In reality a complicated computation task is often decomposed into subtasks with simple structures and these simple structured subtasks are then mapped to parallel machines. Thus the knowledge of mapping simple structured computation tasks to parallel machines is very important in achieving good speedup on parallel computers.

In this paper we will study the problem of mapping a chain task to chained processors. The computation task is represented by a connected linear graph and the parallel machine architecture is a one dimensional linear array. Chain tasks are frequently encountered in parallel computation, they found applications in image processing, signal processing and scientific computing [2][10][11]. The problem of mapping chain tasks to chained processors is studied by several researchers[1][5][6][9][10]. The best algorithms known so far[5] have time complexity $O(mp)$[5] and time complexity $O(m + p^2 \log^2 m)$ [9] for mapping m modules to p processors. The algorithm in [5] uses the dynamic programming technique. In this paper we present an algorithm for the problem with time complexity $O(m + p(\log p/\log\log p)^{1/2} c^{\sqrt{\log p \log\log p}} \log^2 p)$, where c is a constant. The time complexity is less than $O(m + p^{1+\epsilon})$ for any small $\epsilon > 0$. Our algorithm is based on a recursive allocation technique which gains its efficiency by exploiting recursion in the process of mapping task modules to processors.

Because a chain can be embedded with unit dilation cost in networks such as meshes and hypercubes[3], our algorithm can be used for mapping chain tasks to these networks according to the embedding functions.

We adapt our algorithm to the case of mapping a ring task to a ring of processors. The algorithm for mapping a chain task to chained processors can not be used for this case because an arbitrary cutting of the ring task into a chain task can result in a nonoptimal mapping. We thus have to examine several possible cuttings to make sure that we will always find the optimal mapping. Our algorithm for mapping a ring task to a ring of processors has time complexity $O(mp^{\epsilon})$.

Another case we shall consider is that of mapping several chain tasks to chained processors with the constraint that a processor can not be allocated task modules of different chains. This problem models partitionable parallel architectures which can simultaneously execute many tasks with different computational requirements. Such an architecture can be partitioned into subsets of processors where each subset operates independently of others and computes a task. Examples of partitionable architectures include the PASM system[12], and hypercube based systems such as Intel's iPSC[7]. The general problem of allocating and scheduling partitions of processors to achieve

optimal system performance is known to be NP-complete[8]. For the problem of mapping n chain tasks each containing m task modules to a chain of p processors we show an algorithm with time complexity $O(mn + p^{1+\epsilon})$.

2 Preliminary

A chain task is represented by a weighted graph $G = (V, E)$ where $V = \{v_1, v_2, ..., v_m\}$ and $E = \{(v_i, v_{i+1}) | 1 \leq i \leq m-1\}$. The set of vertices V represents the task modules and the edges E represent communication between modules. Each node v_i in G is assigned a weight w_i which is the execution time of the module on a single processor. Each edge (v_i, v_{i+1}) is also assigned a weight c_i which is the amount of communication on the edge if the two modules are mapped to adjacent processors. It is assumed that $c_0 = 0$ and $c_m = 0$. A subchain of $G = (V, E)$ is any set of contiguous modules and the subchain consisting of modules $\{v_i, v_{i+1}, ..., v_j\}$ is denoted $< i, j >$.

Chained processors are represented by a graph $G_p = (V_p, E_p)$ where $V_p = \{P_1, P_2, ..., P_p\}$ are the processors and $E_p = \{(P_i, P_{i+1}) | 1 \leq i \leq p-1\}$ are the communication links. We assume that all processors are homogeneous, i.e., a module will have the same execution time on any processor.

For a chain task $G = (V, E)$ and chained processors $G_p = (V_p, E_p)$, a mapping π is a function: $V \mapsto V_p$. We assume the contiguity constraint[2][5][6][10]. The contiguity constraint stipulates that either $\pi(v_i) = \pi(v_{i+1})$ or if $\pi(v_i) = P_l$ then $\pi(v_{i+1}) = P_{l+1}$.

Because of the contiguity constraint the mapping π always maps a subchain $< i_k, j_k >$ to a processor k. The time taken by processor P_k, when assigned the subchain $< i_k, j_k >$, is $T_k = (\sum_{l=i_k}^{j_k} w_l) + c_{i_k - 1} + c_{j_k}$. Under mapping π, the time taken by the chained processors is the maximum among the time taken by each processor. We define this as the cost of the mapping $C(\pi) = \max_{k \in V_p}\{T_k\}$. Our objective is to find the optimal mapping which minimizes $C(\pi)$ among all π's.

We now discuss two important special cases. The first is the case of a monotonic chain. A chain task is said to be monotonic if for all i, $1 \leq i \leq m$, $c_i < w_{i+1} + c_{i+1}$ and $c_i < w_i + c_{i-1}$. Thus a nonmonotonic chain represents the situation there are two neighboring modules which takes less time when assigned to a single processor than assigned to two neighboring processors because of large communication cost. A nonmonotonic chain can always be condensed to a monotonic chain[6] in time $O(m)$ without affecting the cost of optimal mapping. Thus without loss of generality we need only consider monotonic chains.

The second is the case where all communication costs (c_i's) are zeros. This case is interesting because it is easy to deal with. This case captures the essential features of the general chain mapping

problem. To simplify the exposition our algorithms will be designed for this special case although they work for any monotonic chains.

3 Probing

A basic routine used in mapping is the routine of probing. Probing is the routine that, given a probe value b, check whether or not there exists a π such that $C(\pi) \leq b$. Probing can be done by allocating to processors one by one the task modules, stipulating that each processor gets the largest number of task modules with total weight no more that b. If we have exhausted all processors with unallocated task modules left then the chain can not be mapped to the processors with cost less or equal to b, otherwise it can. Because each task module is examined only once, we have

Lemma 1: Giving a probing value b it takes $O(m)$ time to check whether there exists a π such that $C(\pi) \leq b$. □

It is obvious that such a probing routine is optimal because every task module has to be examined to reach a correct conclusion. However, in our algorithm for mapping a chain task to chained processors such a probing need to be done repeatedly. We therefore use the idea of grouping to reduce the time complexity for repeated executions of the probing routine.

We assume without loss of generality that m is a multiple of p. We use p groups for the task modules. Group i contains $v_{im/p+1}, ..., v_{(i+1)m/p}$, $0 \leq i < p$. For each group we build a binary search tree of height $\lceil \log(m/p) \rceil$ with m/p leaves. The task modules and their weight are placed at the leaves, from left to right. We label each internal node of the tree with the sum of the weight of the leaves in the subtree rooted at that internal node. We therefore have p binary search trees. It takes $O(m)$ time to build these binary search trees.

Now for a value b and a tree T we can check whether the total weight in T is larger than b or not by simply looking at the root of T. If the total weight in T is larger than b we can search into T to find the leave l such that the total weight of the leaves to the left of l is less than or equal to b, but it is larger than b when the weight of l is included. Such a search into T takes $O(\log(m/p))$ time.

When we allocate task modules to a processor in the process of probing we can now look at a binary search tree. We will either allocate the whole tree to the processor or search into the tree to allocate a subtree to the processor. For each processor we need search into a tree at most once besides allocating some whole trees to the processor. There are a total of p trees and each of them can be allocated only once if it is allocated as a whole tree to a processor. Thus the time for allocating whole trees is $O(p)$. The time taken to search into a tree for a processor is $O(\log(m/p))$.

Therefore the time needed for the probing excluding the time for building the trees is $O(p \log(m/p))$.

Lemma 2: Assume that the binary search trees are prebuilt, giving a probe value b it takes $O(p \log(m/p))$ time to check whether there exists a mapping π with $C(\pi) \leq b$. □

We use $S([i_1, j_1], [i_2, j_2], b)$ to denote allocating task modules $v_{i_1}, v_{i_1+1}, ..., v_{j_1}$ to processors $P_{i_2}, P_{i_2+1}, ..., P_{j_2}$ with probing value b. $S([i_1, j_1], [i_2, j_2], b)$ is 1 if the allocation is successful, that is, there is a mapping π with $C(\pi) \leq b$ for mapping task modules $v_{i_1}, v_{i_1+1}, ..., v_{j_1}$ to processors $P_{i_2}, P_{i_2+1}, ..., P_{j_2}$. $S([i_1, j_1], [i_2, j_2], b)$ is 0 if the allocation is unsuccessful. From Lemmas 1 and 2 we know that the value of $S([i_1, j_1], [i_2, j_2], b)$ can be found out in time $O((j_2 - i_2 + 1) \log((j_1 - i_1 + 1)/(j_2 - i_2 + 1)))$ if the binary search trees are prebuilt.

4 Allocation

In this section we give an algorithm for mapping a chain task to chained processors. In order to find the minimum cost mapping we repeatedly allocate task modules to processors and check whether such allocation are valid by invoking the probing routine designed in the last section.

Let g denote the minimum cost for any mapping. Then $S([1, m], [1, p], g) = 1$ and for any $g' < g$ $S([1, m], [1, p], g') = 0$. We describe our first algorithm. We call this algorithm Schedule1. For an i, let $W = \sum_{j=1}^{i} w_j$. We may allocate $v_1, v_2, ..., v_i$ with total weight W to P_1. To make sure that such an allocation is appropriate we may then check $S([i+1, m], [2, p], W)$. That is we probe the value W on mapping $v_{i+1}, ..., v_m$ to processors $P_2, P_3, ..., P_p$. If $S([i+1, m], [2, p], W) = 1$ then we know that $W \geq g$. If $S([i+1, m], [2, p], W) = 0$ then we know that $W < g$. In the case $S([i+1, m], [2, p], W) = 1$ we need find out whether any value smaller than W could still be $\geq g$, therefore we need decrease i. In the case $S([i+1, m], [2, p], W) = 0$ we need find out whether any value larger than W could still be $< g$, therefore we need increase i. Due to the monotonic nature of the chain task there exist an i such that $S([i+1, m], [2, p], W) = 0$ and $S([i+2, m], [2, p], W + w_{i+1}) = 1$. Such an i indicates that $W < g \leq W + w_{i+1}$. A special case is that $i = 0$. In this case we know immediately that $g = w_1$. Otherwise we should consider allocating task modules $v_{i+1}, ..., v_m$ to processors $P_2, P_3, ..., P_p$. This obviously can be done by recursion. The recursion will return a value b which is the minimum cost for mapping $v_{i+1}, ..., v_m$ to processors $P_2, P_3, ..., P_p$. If $b > W + w_{i+1}$ then we should return $W + w_{i+1}$ because we know previously that $g \leq W + w_{i+1}$. If $b \leq W + w_{i+1}$ then we return b because processor P_1 is allocated with weight W. Note that $W < b$ because of the probing we have done. We now give the formal description of Schedule1.

Algorithm **Schedule1**(i_1, j_1, i_2, j_2)
/* Map task modules $v_{i_1}, ..., v_{j_1}$ to processors $P_{i_2}, ..., P_{j_2}$. */

if $i_2 = j_2$ **then** /* One processor left. */

begin

$W := \sum_{j=i_1}^{j_1} w_j$;

return(W);

end

if $S([i_1+1, j_1], [i_2+1, j_2], w_{i_1}) = 1$ **then** return(w_{i_1});

else

begin

Find $i \geq i_1$ such that

$S([i+1, j_1], [i_2+1, j_2], \sum_{j=i_1}^{i} w_j) = 0$ and $S([i+2, j_1], [i_2+1, j_2], \sum_{j=i_1}^{i+1} w_j) = 1$;

b :=Schedule1$(i+1, j_1, i_2+1, j_2)$;

if $b > \sum_{j=i_1}^{i+1} w_j$ **then** return($\sum_{j=i_1}^{i+1} w_j$) **else** return(b);

end

By the analysis above we have,

Theorem 3: Schedule1$(1, m, 1, p)$ returns the minimum cost for mapping the chain task on chained processors. □

Theorem 4: The time complexity of Schedule1 is $O(m + p^2 \log m \log(m/p))$.

Proof: Each recursion has to find out the i satisfying certain conditions. This i can be found by using binary search. Thus we need evaluate S for at most $O(\log m)$ times for locating the right i. Each evaluation (that is, probing) takes $O(p \log(m/p))$ time by Lemma 2. We first take $O(m)$ time for constructing binary search trees. This can be considered as precomputation. Let $T(p)$ be the time needed for scheduling m task modules on p processors. We have the recursion

$T(p) \leq T(p-1) + cp \log m \log(m/p)$, c is a constant.

$T(1) \leq O(m)$.

Which gives $T(p) = O(m + p^2 \log m \log(m/p))$. □

We now generalize algorithm Schedule1. For an i, we map $v_1, v_2, ..., v_i$ to $P_1, P_2, ..., P_t$, for a suitable t. To make sure that such a mapping is appropriate we obtain the minimum cost W for any such mapping. If we choose t to be 1 then W can be easily obtained. If t is not 1 then W can be obtained by invoking recursion if $t < p$. We then check $S([i+1, m], [t+1, p], W)$. That is, we probe the value W on mapping $v_{i+1}, ..., v_m$ to processors $P_{t+1}, P_{t+2}, ..., P_p$. If $S([i+1, m], [t+1, p], W) = 1$ then we know that $W \geq g$. If $S([i+1, m], [t+1, p], W) = 0$ then we know that $W < g$. In the case

$S([i+1,m],[t+1,p],W)=1$ we need find out whether any value smaller than W could be still $\geq g$, therefore we need decrease i. In the case $S([i+1,m],[t+1,p],W)=0$ we need find out whether any value larger than W could be still $< g$, therefore we need increase i. Due to the monotonic nature of the chain task there exists an i such that the minimum cost of mapping $v_1, v_2, ..., v_i$ to $P_1, P_2, ..., P_t$ is W and the minimum cost of mapping $v_1, v_2, ..., v_i, v_{i+1}$ to $P_1, P_2, ..., P_t$ is $W' \geq W$, and $S([i+1,m],[t+1,p],W)=0$ and $S([i+2,m],[t+1,p],W')=1$. Such an i indicates that $W < g \leq W'$. A special case is that $i=0$. In this case we know immediately that $g = w_1$. Otherwise we should consider mapping task modules $v_{i+1}, ..., v_m$ to processors $P_{t+1}, P_{t+2}, ..., P_p$. This obviously can be done by recursion. The recursion will return a value b which is the minimum cost for mapping $v_{i+1}, ..., v_m$ to processors $P_{t+1}, P_{t+2}, ..., P_p$. If $b > W'$ then we should return W' because we know previously that $g \leq W'$. If $b \leq W'$ then we return b. To achieve maximum efficiency we should pick t carefully. We first give the formal description of the algorithm. We show how to pick up a good t when we analyze the time complexity of the algorithm.

Algorithm **Schedule2**(i_1, j_1, i_2, j_2)
/* Map task modules $v_{i_1}, ..., v_{j_1}$ to processors $P_{i_2}, ..., P_{j_2}$. */

if $i_2 = j_2$ **then**
begin
$W := \sum_{j=i_1}^{j_1} w_j$;
return(W);
end

if $S([i_1+1, j_1], [i_2+1, j_2], w_{i_1}) = 1$ **then** return(w_{i_1});
else
begin
Pick a t, $i_2 \leq t < j_2$;
Pick an i, $i_1 \leq i \leq j_1$, using binary search;
W :=Schedule2(i_1, i, i_2, t);
W' :=Schedule2$(i_1, i+1, i_2, t)$;
Let i satisfy that $S([i+1, j_1], [t+1, j_2], W) = 0$
and $S([i+2, j_1], [t+1, j_2], W') = 1$;
b :=Schedule2$(i+1, j_1, t, j_2)$;
if $b \leq W'$ **then** return(b) **else** return(W');
end

From the above analysis we have

Theorem 5: Schedule2$(1, m, 1, p)$ returns the minimum cost for mapping the chain task on chained processors. □

The time complexity of Schedule2 depends on t. If $t = i_2$ then algorithm Schedule2 degenerates to algorithm Schedule1. For each t chosen we analyze the time complexity of Schedule2$(1, m, 1, p)$. In order to find the correct i we need test at most $\log m$ times by using binary search. Each test requires a recursive call to Schedule2$(1, i, 1, t)$ and the evaluation of $S([i+1, m], [t+1, p], W)$ and another recursive call to Schedule2$(1, i+1, 1, t)$ and the evaluation of $S([i+2, m], [t+1, p], W')$. When i is chosen another recursive call to Schedule2$(i+1, m, t+1, p)$ is invoked. These analyses are based on the condition that the binary search trees used for probing and the array $s[i] = \sum_{j=1}^{i} w_j$, $1 \leq i \leq m$, are available. We know that the binary search trees and array $s[i]$ can be constructed in precomputation and they take $O(m)$ time. Now let $T(p)$ be the time complexity of Schedule2$(1, m, 1, p)$, we have

$T(p) \leq 2 \log m(T(t) + cp \log(m/p)) + T(p-t)$,

$T(1) \leq c \log m$, where c is a constant.

Lemma 6: $T(p) = O(p(\log p / \log\log m)^{1/2} c^{\sqrt{\log p \log\log m}} \log^2 m)$, where c is a constant.

Proof: To simplify the estimation we work on the following recurrence equation

$T(p) \leq 2 \log m(T(t) + cp \log m) + T(p-t)$.

If we divide p into segments each of length x and let $t = x$, we have,

$T(p) \leq 2 \log m(T(x) + cp \log m)$
$+2 \log m(T(x) + cp \log m) + 2 \log m(T(x) + cp \log m) + 2 \log m(T(x) + cp \log m) + T(p - 4x)$
$\leq (2cp^2/x) \log^2 m + (2p/x) T(x) \log m$.

We shall take segments of length x_i at level i, x_1 is x. We have

$T(p) \leq (2cp^2/x_1) \log^2 m + (2p/x_1) \log m((2cx_1^2/x_2) \log^2 m$
$+(2x_1/x_2) \log m((2cx_2^2/x_3) \log^2 m + (2x_2/x_3) \log m((2cx_3^2/x_4) \log^2 m + (2x_3/x_4) \log m T(x_4))))$
$= (2cp^2/x_1) \log^2 m + (4cpx_1/x_2) \log^3 m + (8cpx_2/x_3) \log^4 m + (16cpx_3/x_4) \log^5 m + \cdots$

To pick suitable x_i's we let $p/x_1 = (2x_1/x_2) \log m$, $x_i/x_{i+1} = (2x_{i+1}/x_{i+2}) \log m$, $i = 1, 2, ...,$. Let x_j be a constant, we will figure out how large j is. Let x_{j-1} and x_j be constants we have $p = (2 \log m)^{O(j^2)}$. Therefore $j = O((\log p / \log\log m)^{1/2})$. $p/x_1 = O((2 \log m)^j)$. We now have $T(p) \leq O(jp(2 \log m)^j \log^2 m) = O(p(\log p / \log\log m)^{1/2} c^{\sqrt{\log p \log\log m}} \log^2 m)$. □

Theorem 7: Schedule2$(1, m, 1, p)$ runs in time $O(m + p(\log p / \log\log p)^{1/2} c^{\sqrt{\log p \log\log p}} \log^2 p)$.

Proof: When $m \geq p^2$ we have $m > p(\log p / \log\log m)^{1/2} c^{\sqrt{\log p \log\log m}} \log^2 m$. Therefore the theorem

holds. □

The analysis we gave here is pessimistic. t can be chosen by experiments to minimize the time complexity. We note that the time complexity we obtained here is not larger than $O(m + p^{1+\epsilon})$. Thus our algorithm is optimal when $m \geq p^{1+\epsilon}$.

5 Generalizations

5.1 Monotonic Chains

The algorithms we presented in the last section are algorithms designed for a chain task with zero communication costs. We note that these algorithms work for arbitrary monotonic chains. We have to modify our probing routines by incorporating communication costs into binary search trees. The calculation of weight W in our algorithms should be modified to include communication costs. After these modifications our algorithms will work for any monotonic chains.

We have mentioned that an arbitrary chain can always be converted into a monotonic chain in time $O(m)$. Thus the algorithms we gave in the last section could be modified to work for arbitrary chains.

5.2 Multiple Chain Tasks

A set of n chain tasks is defined by a set of task graphs $G_c = \{G^1 \cup G^2, ..., \cup G^n\}$. Each task i is represented by the graph $G^i = (V^i, E^i)$ where G^i is a chain task with m modules, such that $V^i = \{v_1^i, v_2^i, ..., v_m^i\}$. and $E^i = \{(v_j^i, v_{j+1}^i) | 1 \leq j \leq m-1\}$. The weight of a module j of task i is denoted as w_j^i, and the weight of edge (v_j^i, v_{j+1}^i), the communication time, is denoted c_j^i. The set of chains is therefore $G_c = (V_c, E_c)$, where $V_c = \cup_{1 \leq i \leq n} V^i$ and $E_c = \cup_{1 \leq i \leq n} E^i$. A mapping π must map modules of all tasks to the p processors and is defined as: $\pi : V_c \mapsto V_p$. The mapping π must satisfy two constraints, the contiguity constraint and the partition constraint. The contiguity constraint stipulates that for any two modules v_i^j and v_{i+1}^j, $\pi(v_i^j) = \pi(v_{i+1}^j)$ or if $\pi(v_i) = P_l$ then $\pi(v_{i+1}) = P_{l+1}$. The partition constraint stipulates that if $\pi(v_k^i) = \pi(v_l^j)$, $1 \leq k, l \leq m$, then $i = j$ for $1 \leq i, j \leq n$. The cost of the mapping $C(\pi)$ is as defined before.

After adding the partition constraint to our algorithms they work for mapping multiple chain tasks to a linear array of processors. Therefore we have,

Theorem 8: It takes $O(mn + p^{1+\epsilon})$ time to compute the optimal mapping for mapping n chains each containing m modules to a chain of p processors. □

5.3 Mapping a Ring Task to a Ring of Processors

We now consider the case where the input chain task is a ring and we are to map it to a ring of processors. And we assume that the communication costs are 0's as it is easy to generalize it to the case where the communication costs are not 0's. Since the processors are homogeneous we could allocate task modules starting at any processor. The problem here is which task module we should start allocating. If we start with an arbitrary task module we may end up finding a mapping which is not optimal. The following example demonstrates this situation.

Weight on task modules: 2, 3, 3, 4, 6, 5, 8. These task modules form a ring and they are to be mapped to a ring of three processors. Below we show some possible ways of mapping.

(1) 2 3 3 4 — 6 5 — 8, cost of mapping is 12.

(2) 3 3 4 — 6 5 — 8 2, cost of mapping is 11.

(3) 3 4 6 — 5 8 — 2 3, cost of mapping is 13.

(4) 4 6 — 5 8 — 2 3 3, cost of mapping is 13.

Consider the problem of mapping m task modules with weight $w_1, w_2, ..., w_m$ to p processors, let $W = \sum_{i=1}^{m} w_i$ and $M = \max\{w_i, 1 \leq i \leq m\}$. We have the following lemma.

Lemma 9: If $M \geq 2W/p$ then the optimal mapping has cost M, otherwise the optimal mapping has cost $\leq 2W/p$.

Proof: We start with an arbitrary task module and an arbitrary processor P. We allocate weight W/p to each processor. This allocation will result in allocating one module to several contiguous processors. Start with processor P, we will work on processors one by one. Suppose currently we are working with processor P', we check whether there is only one task module allocated to P'. If there is only one task module allocated to P' and if part of the task module is allocated to some processors following P' we shall allocate the whole task module to P'. If there are more than one task module allocated to P' we check whether the last task module allocated to P' is a complete module or not. If it is not we will allocate this module to the next processor. When we finish with all the processors we obtain a mapping with cost $\leq \max\{M, 2W/p\}$. □

By Lemma 9 we may check whether $M \geq 2W/p$ or not. If $M \geq 2W/p$ then the optimal mapping has cost M. Otherwise we know that the cost of the optimal mapping is $\leq 2W/p$. If $\sum_{i=j}^{k} w_i \geq 2W/p$ we may try to start with each of the m_i, $j \leq i \leq k$, and one of them will give the cost of an optimal mapping. If we check the sum of the weight of every contiguous $\lceil 2m/p \rceil$ task modules, one of the contiguous $\lceil 2m/p \rceil$ task modules will have total weight $\geq 2W/p$, for otherwise the total weight of

all task modules will be $< W$. We therefore have

Theorem 10: An optimal mapping of a ring task to a ring of processors can be computed in $O(mp^{\epsilon})$ time. □

References

[1] S.H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, Vol. C-30, pp. 207-214, March 19981.

[2] S.H. Bokhari. Partitioning problems in parallel, pipelined and distributed computing. *IEEE Trans. Comp.* , Vol. 37, No. 1, Jan. 1988, pp. 48-57.

[3] T.F. Chan and Y. Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Trans. on Comp.*, Vol. C-35, No. 11, 1986.

[4] H.-A. Choi and B. Narahari. Efficient algorithms for mapping and partitioning a class of parallel computations. Tech. Report, GWU-IIST, March 1991.

[5] H.-A. Choi and B. Narahari. Algorithms for mapping and partitioning chain structured parallel computations. *Proc. 1991 Int. Conf. on Parallel Processing*, Vol. I, pp. 625-628.

[6] M.A. Iqbal and S.H. Bokhari. Efficient algorithms for a class of partitioning problems. ICASE Report No. 90-49, July 1990, NASA Contractor Report 182073.

[7] Intel Corp., *Intel iPSC System Overview*, 1986.

[8] R. Krishnamurti and Y.E. Ma. The processor partitioning problem in special-purpose partitionable systems. *Proc. 1988 International Conf. on Parallel Processing*, Vol. 1, pp. 434-443.

[9] D.M. Nicol. Rectilinear partitioning of irregular data parallel computations. Personal communication.

[10] D.M. Nicol and D.R. O'Hallaron. Improved algorithms for mapping pipelined and parallel computations. *IEEE Trans. Computers*, 40(3), 295-306, March 1991.

[11] D.A. Reed, L.M. Adams and M.L. Patrick. Stencils and problems partitionings: their influences on the performance of multiple processor systems. *IEEE Trans. Comp.*, C-36, No. 7, July 1987, pp. 845-858.

[12] H.J. Siegel, L.J. Siegel, F.C. Kemmerer, P.T. Mueller, Jr., H.E. Smalley, and S.D. Smith. PASM: A Partitionable SIMA/MIMD system for image processing and pattern recognition. *IEEE Trans. on Comp.*, Vol. C-30, No. 12, December 1981.