NASA Contractor Report 182073
ICASE Report No. 90-49

ICASE

EFFICIENT ALGORITHMS FOR A CLASS OF PARTITIONING PROBLEMS

(NASA-CR-182073) EFFICIENT ALGORITHMS FOR A CLASS OF PARTITIONING PROBLEMS Final Report (ICASE) 27 p CSCL 09B
N90-28357
Unclas
G3/61 0302464

M. Ashraf Iqbal
Shahid H. Bokhari

Contract No. NAS1-18605
July 1990

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665-5225

Efficient Algorithms for a Class of Partitioning Problems

M. Ashraf Iqbal[*]

Department of Electrical Engineering

University of Engineering & Technology, Lahore, Pakistan

Shahid H. Bokhari[†]

ICASE, NASA Langley Research Center

Hampton, Virginia

Abstract

We address the problem of optimally partitioning the modules of chain- or tree-like tasks over chain-structured or host-satellite multiple computer systems. This important class of problems includes many signal processing and industrial control applications. Prior research has resulted in a succession of faster exact and approximate algorithms for these problems.

We describe polynomial exact and approximate algorithms for this class that are better than any of the previously reported algorithms. Our approach is based on a preprocessing step that condenses the given chain or tree structured task into a *monotonic* chain or tree. The partitioning of this monotonic task can then be carried out using fast search techniques.

[*]Research supported by a grant from the Division of Research Extension and Advisory Services, University of Engineering & Technology, Lahore, Pakistan

[†]On leave from the Department of Electrical Engineering University of Engineering & Technology, Lahore, Pakistan. Research supported by the National Aeronautics and Space Administration under NASA contract NAS1-18605 while the author was in residence at the Institute for Computer Applications in Science & Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225.

1 Introduction

The problem of assigning the constituent parts of a large parallel application onto the processors of a multiple computer system is one of the key issues in parallel processing. While the general form of this problem has eluded efficient solution [1, 3] there has been considerable success for problems with constrained structure. The mapping of problems with chain- or tree-like structure on multiple computer systems with chain-like interconnection or on host-satellite systems was shown to have exact polynomial time solutions by Bokhari [2]. Iqbal [6] subsequently developed faster approximate algorithms for this class of problems. These fully polynomial algorithms were faster but provided solutions only to a desired degree of accuracy ϵ. Nicol & O'Hallaron [8] improved Bokhari's exact algorithms and developed new algorithms that were still faster but operated under the assumption of bounded execution and communication costs. In the present paper we describe a new 'condensation' approach that permits exact polynomial time solutions to these problems that are faster than any of the previously reported exact or approximate algorithms. Our approach involves a preprocessing step on the given chain or tree that makes it *monotonic* and permits a very fast exact solution. These new algorithms are straightforward to implement and provide the exactness of [2], the speed of [8], are no more involved than those of [6], and make no assumptions about magnitudes of costs.

Chain-structured computations form an important class that includes many signal processing applications. Such computations are conveniently carried out on chain structured machines in parallel or pipelined mode [4, 5]. Tree-structured computations also arise in signal processing as well as in industrial control applications [2]. In the latter case sensor inputs from the shop floor are processed up the nodes of a tree to a central control node, and control signals travel in the reverse direction. Such tree-structured computations can be partitioned over the processors of a host-satellite system to improve response time.

In Section 2 of this paper we describe the key theoretical results related to our condensation approach. We show how monotonic chains are obtained and discuss their properties. The concept of monotonicity permits us to develop improved algorithms for partitioning chain structured programs on chain connected processors. We describe approximate and exact algorithms that utilize the condensation approach in Section 3. Section 4 addresses the problem of assigning multiple chain-structured computations on a host-satellite system and develops improved approximate and exact algorithms for these. In Section 5 we describe an improved exact algorithm for partitioning a tree structured computation over a host-satellite system. Section 6 summarizes the results of this paper.

2 The Partitioning Problem

In this Section we will define our assignment problem and discuss the properties of chains. We will show how a given chain can be transformed into a *monotonic* chain and how this transformation permits faster solutions to the assignment problem.

2.1 Statement of Problem

We will assume that we are given a chain-structured program of m modules (numbered 1 to m) and that this is to be partitioned over a chain structured processor with $n < m$ nodes (numbered 1 to n). With each module i is associated an execution cost w_i and a communication cost c_i. w_i is the time required to execute that module on any processor (we assume a homogeneous system), while c_i is the time required for module i to communicate with module $i+1$.

We will work under the assumption that each processor has a contiguous subchain of modules assigned to it. Thus the chain is partitioned into subchains such that modules i and $i+1$ reside on the same or on adjacent processors. We call this the *contiguity constraint.* When a subchain is assigned to a processor, the load on that processor is the sum of the execution costs w_i plus the communication costs for the two modules at the ends of the subchain. The time required for the entire system to complete the task is equal to the time taken by the most heavily loaded processor which is equivalent to the weight of the heaviest subchain. The next subsection summarizes these definitions.

The problem of finding the partitioning that minimizes the weight of the heaviest subchain was originally solved by Bokhari [2] in $O(m^3n)$ time. This is an exact algorithm that makes no assumptions about the magnitudes of the execution or communication costs. This algorithm was improved to

$O(m^2n)$ by Nicol & O'Hallaron [8]*. Iqbal [6] developed a fully polynomial approximation algorithm that obtained an assignment optimal to within a factor of ϵ in time $O(mn\log(W/\epsilon))$, where W is the sum of all execution costs. Nicol & O'Hallaron [8] reported a carefully developed algorithm that could solve this problem in $O(mn\log m)$ time under the assumption that the w_is and the c_is are bounded. One of the major results in the present paper is an algorithm that solves this problem in $O(mn\log m)$ time with no assumptions about the magnitudes of costs. We will also describe a faster approximation algorithm.

Since we will be discussing the partitioning of a chain of modules over a chain of homogeneous processors, the problem is equivalent to partitioning chains into subchains. We will consider subchains and processors to be synonymous in the following discussion.

2.2 Definitions

w_i execution time of module i.

c_i communication time between modules i and $i+1$.
We assume that $c_0(c_m)$ is the time required for module $1(m)$ to communicate with the outside world.

W load on a processor if all m modules are assigned to it.
$W = \sum_{i=1}^{m} w_i + c_0 + c_m$

$\Omega_{p,s,t}$ load on processor p if subchain $s \cdots t$ is assigned to it.
$\Omega_{p,s,t} = \sum_{i=s}^{t} w_i + c_t + c_{s-1}$.
This is synonymous with the weight of subchain p.

$\tau(p)$ a vector of length n that specifies the partition.
Processor p has the subchain $\tau(p-1)+1 \cdots \tau(p)$ assigned to it, with $\tau(0) \triangleq 0$.

bottleneck processor/subchain: for a given $\tau(p)$ the processor/subchain with weight $\max_p\{\Omega_{p,\tau(p-1)+1,\tau(p)}\}$

*These two algorithms permit heterogeneous processors. The remaining algorithms assume homogeneous processors.

$\omega(\tau(p))$ *weight of a partition* = the weight of its bottleneck processor. This is denoted by ω when no confusion is likely.

The *optimal* partition is the $\tau(p)$ for which the weight $\omega(\tau(p))$ is minimum.

2.3 The Condensation Theorem

Theorem 1. Consider a chain that has a partition of weight ω, and in which there exists an edge c_t such that either $c_t \geq w_{t+1} + c_{t+1}$ or $c_t \geq w_t + c_{t-1}$, or both. Then this chain will continue to have a partition of weight $\leq \omega$ if we merge modules t and $t+1$.

Proof. In the given partition of weight ω, modules t and $t+1$ must belong to different subchains, otherwise the proof is trivial. We assume that modules $s \cdots t$ belong to subchain p and that modules $t+1 \cdots u$ belong to subchain $p+1$ (see Figure 1). The weights of these subchains are

$$\begin{aligned} \Omega_{p,s,t} &= \sum_{i=s}^{t} w_i + c_{s-1} + c_t \\ \Omega_{p+1,t+1,u} &= \sum_{i=t+1}^{u} w_i + c_t + c_u. \end{aligned}$$

Let us merge modules t and $t+1$ into one module. The condensed module can be assigned either to subchain p or to subchain $p+1$. If it is assigned to subchain p, the weights of the two subchains become

$$\begin{aligned} \Omega_{p,s,t+1} &= \textstyle\sum_{i=s}^{t+1} w_i + c_{s-1} + c_{t+1} &&= \Omega_{p,s,t} + w_{t+1} - c_t + c_{t+1} \\ \Omega_{p+1,t+2,u} &= \textstyle\sum_{i=t+2}^{u} w_i + c_{t+1} + c_u &&= \Omega_{p+1,t+1,u} - w_{t+1} - c_t + c_{t+1}. \end{aligned}$$

If $c_t \geq w_{t+1} + c_{t+1}$ we obtain

$$\begin{aligned} \Omega_{p,s,t+1} &\leq \Omega_{p,s,t} \\ \Omega_{p+1,t+2,u} &\leq \Omega_{p+1,t+1,u}. \end{aligned}$$

If the condensed module is assigned to subchain $p+1$, the weights of the two subchains become

$$\begin{aligned} \Omega_{p,s,t-1} &= \textstyle\sum_{i=s}^{t-1} w_i + c_{s-1} + c_{t-1} &&= \Omega_{p,s,t} - w_t - c_t + c_{t-1} \\ \Omega_{p+1,t,u} &= \textstyle\sum_{i=t}^{u} w_i + c_{t-1} + c_u &&= \Omega_{p+1,t+1,u} + w_t - c_t + c_{t-1}. \end{aligned}$$

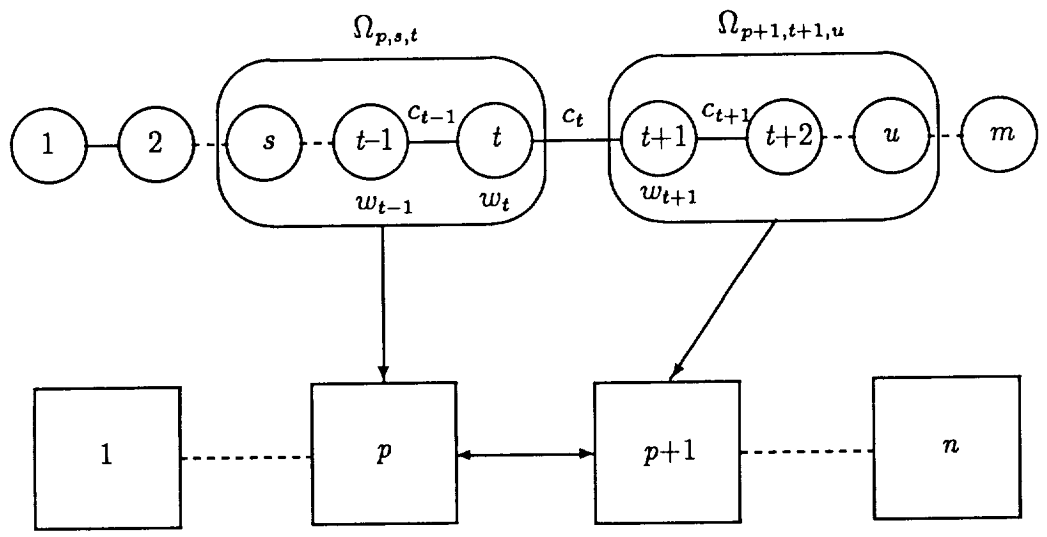

Figure 1: A chain of m modules mapped onto a chain of n processors. The w_is are execution costs; c_is are communication costs. Modules $s \cdots t$ are assigned to processor p; modules $t+1 \cdots u$ are assigned to processor $p+1$.

If $c_t \geq w_t + c_{t-1}$ we obtain

$$\begin{aligned} \Omega_{p,s,t-1} &\leq \Omega_{p,s,t} \\ \Omega_{p+1,t,u} &\leq \Omega_{p+1,t+1,u}. \end{aligned}$$

Our condensation disturbs only subchains p and $p+1$, all other subchains remain undisturbed. The pairs of inequalities obtained above assure us that there will always be one case in which the weights of these condensed subchains is less than the weights of the original uncondensed subchains. Thus the entire condensed chain will have a partition with weight $\leq \omega$.□

2.4 Monotonic Chains

A given chain of m modules can be transformed into a chain of $m' \leq m$ modules by applying the procedure **condense**. This procedure looks at all edges in the chain and merges modules t and $t+1$ if $c_t \geq w_{t+1} + c_{t+1}$ or $c_t \geq w_t + c_{t-1}$, or both. From Theorem 1, we know that if a given chain has

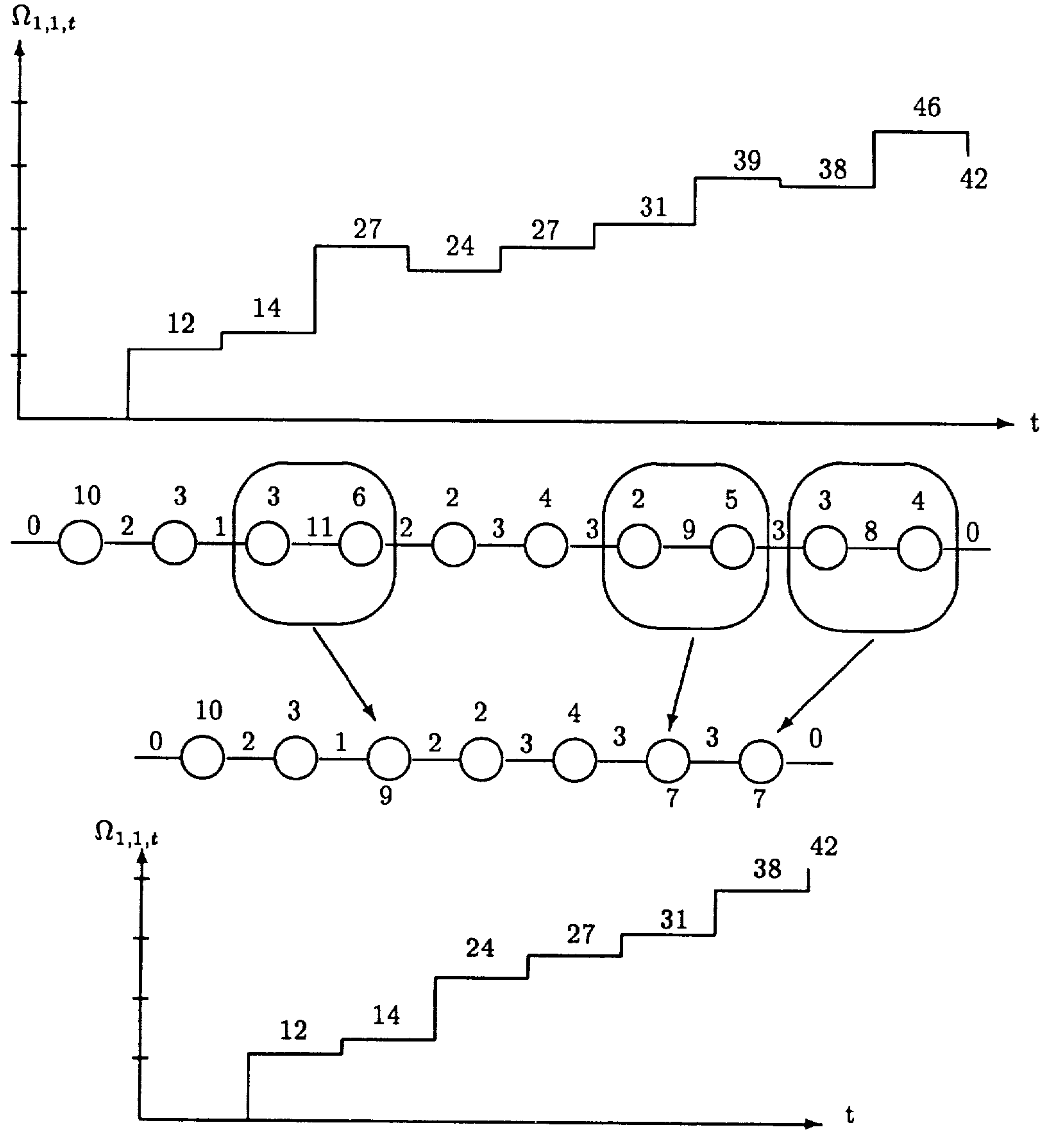

Figure 2: *Top.* A 10 module chain and the plot of its $\Omega_{1,1,t}$ which is not monotone. *Bottom.* The 10 module chain transformed into a 7 module chain by applying procedure **condense**. The plot of the condensed chain's $\Omega_{1,1,t}$ is monotonic.

a partition of weight ω the corresponding condensed chain will also have a partition of weight $\leq \omega$. This procedure obviously takes $O(m)$ time.

Theorem 2. In a chain that has been transformed by applying procedure **condense**, $\Omega_{p,s,t} \leq \Omega_{p,s,t+1}$, for all $1 \leq p \leq n, 1 \leq s, t, \leq m$.

Proof. By contradiction. Suppose $\Omega_{p,s,t} > \Omega_{p,s,t+1}$. Then

$$\begin{aligned} \sum_{i=s}^{t} w_i + c_t + c_{s-1} &> \sum_{i=s}^{t+1} w_i + c_{t+1} + c_{s-1}, \text{ and thus} \\ c_t &> w_{t+1} + c_{t+1}. \qquad (1) \end{aligned}$$

But this is impossible since **condense** removes all edges that satisfy (1). □

An important consequence of Theorem 2 is the fact that all condensed chains are *monotonic*: the weight of a subchain cannot decrease as more nodes are added to it. This property is crucial to the material that follows.

2.5 Probing Function

Once a given chain has been transformed into a monotonic chain, we can use the function **probe**(m, n, ω) on it. This procedure returns **true** if it is possible to partition the given chain of m modules into n subchains each with weight $\leq w$, and **false** otherwise.

```
function probe(processors[$1 \cdots n$], modules[$1 \cdots m$], $\omega$):boolean;
begin
1.  $s := 1; t := 1; p := 1$;
2.  while $p \leq n$ do
        begin
3.          attempt to find a $t \geq s$ such that
                $(\Omega_{p,s,t} \leq \omega)$ and $((\Omega_{p,s,t+1} > \omega)$ or $(t = m))$
4.          if $t = m$ then return(true);
5.          Assign subchain $s \cdots t$ to processor $p$;
6.          s:=t+1; p:=p+1;
        end;
7.  return(false);
end;
```

The search at step 3 can be carried out by simply incrementing t, in which case this procedure takes time proportional to m, the number of modules in the condensed chain. However, the monotonicity of the condensed chain permits us to use a binary search over the remaining modules at step 3. This is because once we have computed $\Omega_{1,1,t}$ for all t, there is no need to compute any other $\Omega_{p,s,t}$ since $\Omega_{p,s,t} = \Omega_{1,1,t} - c_0 - \sum_{i=1}^{s-1} w_i + c_{s-1}$ (this is illustrated in Figure 3). Thus we need to compute $\Omega_{1,1,t}$ once for all t, and compute $\sum_{i=1}^{s-1} w_i$ once for all s. These computations take $O(m)$ time each and subsequently let us execute **probe** in $O(n \log m)$ time. Thus each execution of **probe** takes $O(\min(m, n \log m))$ time, depending on the search strategy.

This is a *greedy* algorithm and the partition that it returns is called a *greedy partition.* In [7] a similar probing function was applied to chains with zero communication costs.

Theorem 3. If it is possible to partition a chain with m modules into n subchains, each with weight ω, the function **probe**(m, n, ω) will always find that or a partition of weight $\leq \omega$.

Proof. Similar to the proof given in [7]. Omitted for brevity.□

3 Partitioning Chains on Chains

We now show how the results of the preceding Section can be used to obtain faster algorithms for partitioning chains on chains. We will discuss first an approximation algorithm that supplies an answer to within any specified degree ϵ of accuracy. We will then go on to develop a fast exact algorithm.

3.1 Approximate Assignment

Suppose we wish to solve the problem of partitioning chains on chains approximately. That is, we wish to partition a chain of m modules into n subchains such that the weight of the heaviest subchain is within ϵ of the optimal partition. We proceed by first applying procedure **condense** on the given chain. An upper bound on the weight of the optimal partition is W, the cost of executing all modules on one processor. A lower bound is 0. We can divide this interval into no more than W/ϵ subintervals and conduct a binary

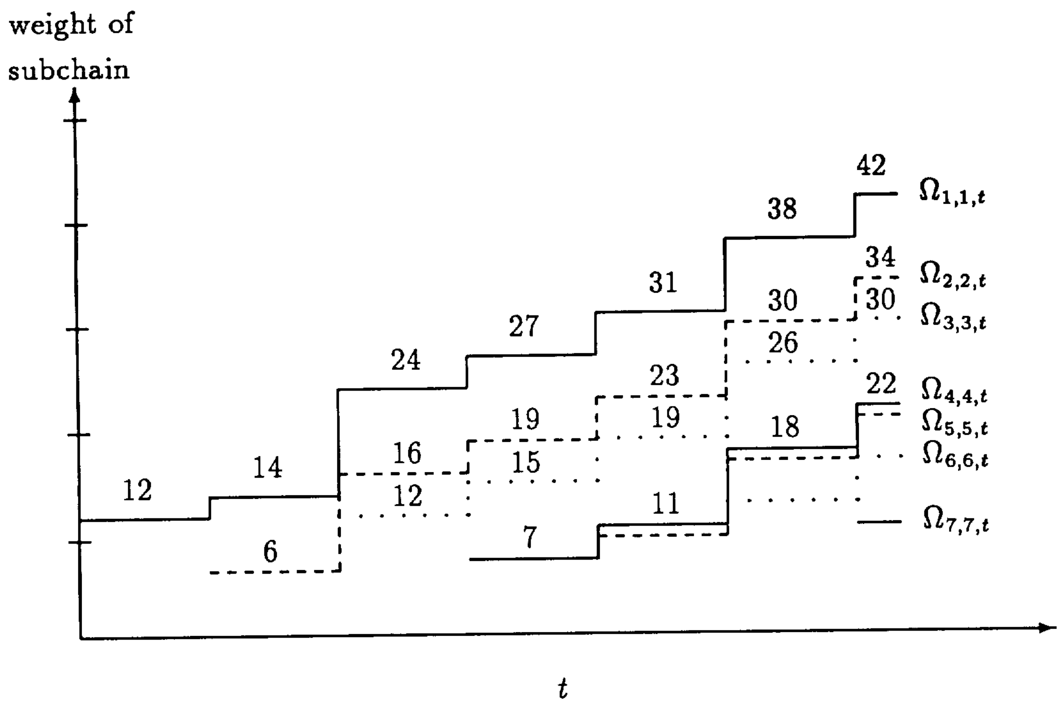

Figure 3: The plots of $\Omega_{1,1,t}$ and $\Omega_{s,s,t}$ are spaced exactly $\sum_{i=1}^{s-1} w_i + c_0 - c_{s-1}$ apart. Thus a binary search on $\Omega_{2,2,t}$ can be carried out on $\Omega_{1,1,t}$ by compensating for the offset $w_1 - c_1$. Some numbers have been omitted to avoid congestion.

search using **probe** over this range. A binary search is permissible since the chain has been condensed into a monotonic chain. Thus the time required is $O(\min(m, n \log m) \log(W/\epsilon))$. This is better than the best previously known approximation algorithm [6] which is $O(mn \log(W/\epsilon))$.

3.2 A Simple Exact Algorithm

Once we have condensed our chain of modules into a monotonic chain, we can compute the $O(m^2)$ values of $\Omega_{1,s,t}, 1 \leq s \leq m, 1 \leq t \leq m$ (we assume that the condensed chain has m modules). We can arrange these values in a master sorted list without having to sort explicitly. This is because each $\Omega_{1,s,t}$ is monotonic for a fixed s. We can thus merge each Ω into the master list in $O(m^2 \log m)$ time. Once this list has been generated, we can binary search over it using **probe** and find the optimal assignment in $O(\min(m, n \log m) \log m)$ time. Assuming $m \geq n$, the total time is masked by the time to create the master list, which is $O(m^2 \log m)$.

3.3 Improved Exact Algorithm

Observe first that, since our chain connected system is homogeneous, $\Omega_{p,s,t} = \Omega_{q,s,t}$ for all p, q. Thus we can always fix module 1 to processor 1 and consider only the $m(m-1)/2$ values of $\Omega_{s,s,t}, 1 \leq s \leq m, 1 \leq t \leq m$.

The number of probes required to find the optimal bottleneck subchain can be reduced by carefully analyzing the relationships between Ωs. These are shown by the lattice of Figure 4 in which each node represents an Ω and a directed edge from node p to node q implies that $p > q$. Monotonicity of the chain ensures that $\Omega_{s,s,t} < \Omega_{s,s,t+1}$. This accounts for the horizontal edges. We can also observe that $\Omega_{s,s,t} - \Omega_{s+1,s+1,t} = c_{s-1} + w_s - c_s$, which is positive for condensed chains. This accounts for the vertical edges.

We can use binary search with **probe** over the median row s' of this lattice to find the smallest t' for which **probe**$(\Omega_{s',s',t'})$ is **true**. Once this has been done and the value of $\Omega_{s',s',t'}$ recorded, we can eliminate from consideration all $\Omega_{s,s,t}$ with $s \geq s'$ and $t < t'$ since **probe**$(\Omega_{s,s,t})$ is guaranteed to be **false** in this range. We can also eliminate all $\Omega_{s,s,t}$ with $s \leq s'$ and $t \geq t'$, since $\Omega_{s',s',t'}$ is the smallest feasible value in this region. Figure 4 illustrates these regions. This process of elimination is continued recursively on the two remaining subregions. This 2-dimensional search technique is due to Nicol & O'Hallaron [8] who show that it takes no more than $4m$ probes to find the optimal value.

Since our **probe** takes $O(\min(m, n \log m))$, we have an overall complexity of $O(mn \log m)$. This is the same as Nicol & O'Hallaron's algorithm [8], which assumes bounded execution and communication costs. Our algorithm makes no assumptions about the magnitudes of costs.

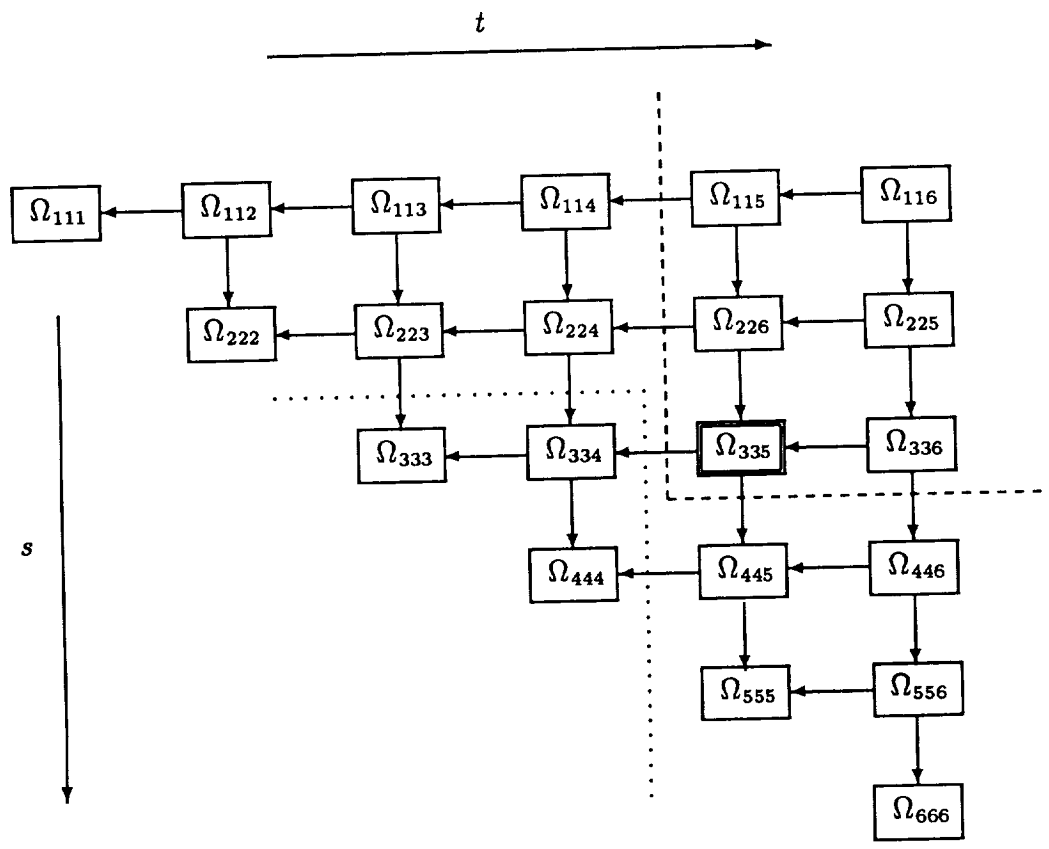

Figure 4: Illustration of 2-dimensional binary search over $\Omega(s, s, t)$. A search over row 3 yields $\Omega_{3,3,5}$ as the smallest for which **probe** returns **true**. We can now eliminate from consideration all Ωs in the dotted region, as **probe** can never be **true** for these. We can also eliminate the dashed region, since $\Omega_{3,3,5}$ is the smallest from among these Ωs.

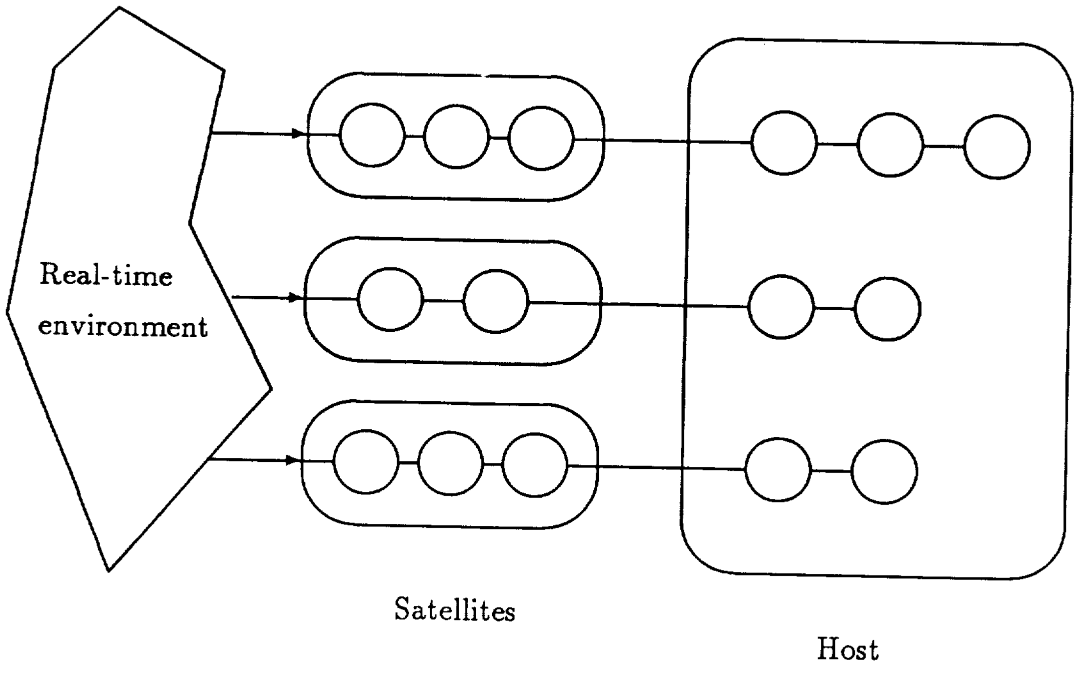

Figure 5: A host-satellite system processing real-time data.

4 Chains on Host-Satellite Systems

We now address the problem of partitioning multiple chains on a host-satellite system. In this case we assume that we have a large, powerful host computer connected to many smaller satellite machines (Figure 5). Each satellite receives a data stream from a real time environment, performs a chain of computations on it, and forwards the results to the central host. It is possible to partition each satellite's chain so that some of its modules reside on the host and take advantage of the host's greater computational power. We are interested in minimizing the time required for all satellites to complete one iteration of their respective tasks. If too much load is assigned to the host, then the time to complete one iteration of all tasks will increase to an intolerable extent. On the other hand if all chains reside on their respective satellites then the power of the host is wasted. The problem is to find a balance between the two extremes, i.e. a partitioning of the several chains that minimizes the maximum of (1) the most heavily loaded satellite and (2) the total load on the host. As before, we assume partitions into contiguous subchains. In the present case, this means that each chain is divided into two contiguous subchains, one of which resides on the host and the other on the satellite.

4.1 Definitions

n number of satellites.

m number of modules per chain. For simplicity, we assume that all chains have the same number of modules.

$e_{i,s}$ execution time of module i of satellite s.

$c_{i,s}$ communication time between modules i and $i+1$ of satellite s. We assume that $c_{0,s}(c_{m,s})$ is the time required for module $1(m)$ of satellite s to communicate with the outside world(the host).

α_s for satellite s, the ratio of compute time for a module on the satellite to its compute time on the host. Thus module i will take $w_{i,s}$ time on satellite s and $w_{i,s}/\alpha_s$ on the host.

$\Omega_{s,t}$ load on satellite s if subchain $1 \cdots t$ is assigned to it. $\Omega_{s,t} = \sum_{i=1}^{t} w_{i,s} + c_{t,s}$.

$\Delta_{s,t}$ load on host caused by modules $t+1 \cdots m$ of chain s. $\Delta_{s,t} = \sum_{i=t+1}^{m} w_i/\alpha_s + c_t$.

We can denote a partition of chains by the vector $\mathcal{T}_1, \mathcal{T}_2, \cdots \mathcal{T}_n$ such that modules $1 \cdots \mathcal{T}_s$ of chain s are assigned to satellite s and the remaining to the host. The time required by this partition is

$$\max(\max_{1 \le s \le n} \Omega_{s,\mathcal{T}_s}, \sum_{s=1}^{n} \Delta_{s,\mathcal{T}_s}). \tag{2}$$

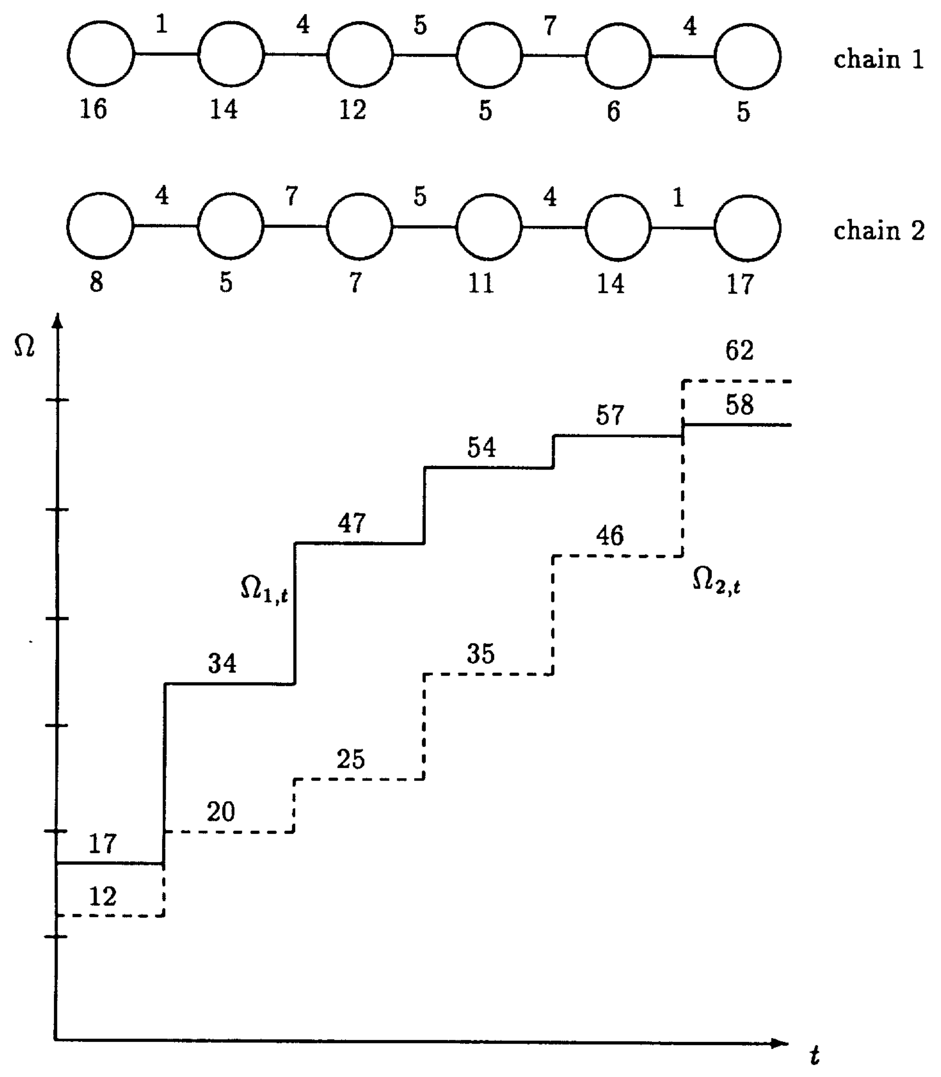

Figure 6: Two monotonic chains to be partitioned over a host satellite system

4.2 Condensing Chains

The chains of our single-host multiple-satellite system can be condensed into monotonic chains. A complicating issue is the fact that each module has two execution costs ($w_{i,s}$ on the satellite and $w_{i,s}/\alpha_s$ on the host). A chain that is monotonic with respect to one execution cost may not necessarily be monotonic with respect to the other. However the probing function that we describe in the following subsection is concerned only with satellite weights and it therefore suffices to condense the chain with respect to these satellite weights.

4.3 Probing Function

We now assume that all our n chains of m modules are condensed, monotonic chains as discussed above. If we view a single host-satellite combination as a two processor system, we can apply a simple modification of function **probe** of Section 2.5 to determine if this chain can be divided into two subchains such that the satellite has load $\Omega_{s,k} \leq \omega$ on it and k is maximum. Since our chains are monotonic with respect to satellite costs, this version of **probe** can use binary search and provide an answer in $O(\log m)$ time. This function will return **true** or **false** and will specify k and $\Omega_{s,k}$ in case the answer is **true**. It is straightforward to compute $\Delta_{s,k}$ in constant time from this information.

Given an ω we can compute if there exists a partition that puts $\Omega_{s,T_s} \leq \omega$ load on each of the satellites and $\sum_{s=1}^{n} \Delta_{s,T_s} \leq \omega$ total load on the host as follows. Apply **probe**(ω) to each of the satellites, computing and adding up all Δ_{s,T_s}s as they are reported. If all processors answered **true** and if $\sum_{s=1}^{n} \Delta_{s,T_s} \leq \omega$, there does indeed exist a partition that puts no more than ω load on each of the satellites and on the host. This entire 'ensemble' probe can be carried out in $O(n \log m)$ time.

4.4 Partitioning Algorithms

In a problem with n chains of m modules each, there are mn possible values of ω. We could carry out mn 'ensemble' probes to obtain the assignment that minimizes (2) in $O(mn^2 \log m)$ time. This is an exact algorithm, but is not an improvement over previously known exact algorithms. If we denote by W the time taken if all modules are assigned to the host and resolve to an accuracy of ϵ, we immediately obtain an approximation algorithm that takes $O(n \log m \log(W/\epsilon))$ time, which is better than Iqbal's $O(mn \log(W/\epsilon))$ approximation algorithm [6].

However it is possible to do much better. Note that our n monotonic chains have m potential ωs each, in ascending order. These n lists can be merged into one sorted list in $O(mn \log n)$ time. We can subsequently use binary search over this sorted list to solve our problem in $O(\log(mn) n \log m) = O(n \log^2 m + n \log m \log n)$ time. This time is masked by the $O(mn \log n)$ time to condense chains and to merge ωs. This time is equal to Nicol & O'Hallaron's $O(mn \log n)$ algorithm, which assumes bounded execution and communication costs. Our algorithm makes no such assumption.

5 Trees on Host-Satellite Systems

We now consider the problem of partitioning a tree structured program over a host-satellite system. Our program is made up of a number of modules that can execute either on the host or on one of the satellites. As in the previous Section, we have a motivation to assign as many modules as possible on the host in order to take advantage of its greater power. However, we do not wish to load the host to the point that the time required for it to complete its portion of the task is greater than the time that would have been required by the satellites.

We will assume that our partitioning is under the following constraints.

1. The root of the tree is always assigned to the host,
2. if a specific node is assigned to a satellite, all its children nodes are also assigned to the same satellite,
3. if two nodes are assigned to a satellite, their lowest common ancestor is also assigned to the same satellite.

In other words each satellite has a single maximal subtree assigned to it. An example of a partition that satisfies these constraints is given in Figure 7. We will assume that we have available as many nodes as there are satellites and that the optimal assignment may choose not to use some of them. This is a good model of many industrial process monitoring and/or control systems.

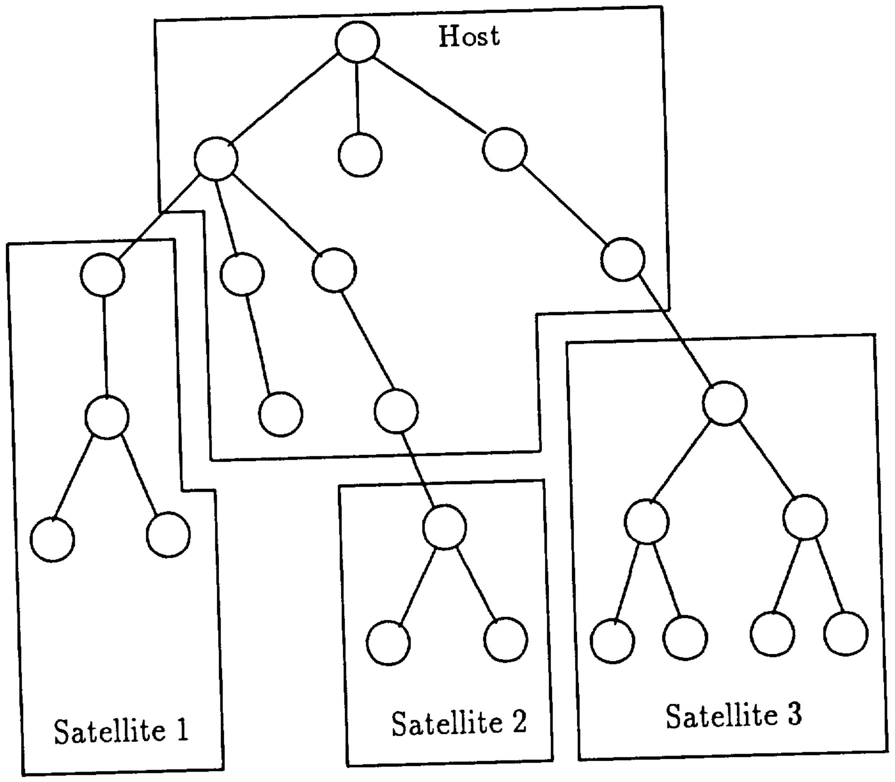

Figure 7: A tree structured program partitioned over a host-satellite system

In such systems, external information from the shop floor is gathered by satellite computers and processed in a hierarchical fashion up the levels of a tree. The root of this tree resides on a large, central host machine. Control signals from the host travel in the opposite direction. Processing may be done in a pipelined or parallel fashion. It is important to partition the tree between the host and the satellites such that the response time of the system is minimized. As in the preceding Section, this response time depends on the larger of (1) the load on the most heavily loaded satellite and (2) the total load on the host.

5.1 Definitions

m number of modules in the tree.

n number of satellites in a given partition.

e_i execution time of module i on a satellite. All satellites are assumed to be similar.

α the ratio of compute time for a module on a satellite to its compute time on the host. Thus module i will take e_i time on a satellite and e_i/α on the host. We assume that $\alpha > 1$ (the host is more powerful than the satellites).

c_i communication time between modules i and father(i) if i is assigned to a satellite and father(i) to the host.

$\mathcal{C}(i)$ the set of children of node i.

$\tau(p)$ the root node of the subtree assigned to satellite p.

$\mathcal{T}(i)$ the set of nodes in the subtree rooted at node i.

$\mathcal{H}_i$ contribution to the load on the host made by the assignment of the subtree rooted at node i to the host.
$\mathcal{H}_i = \sum_{j \in \mathcal{T}(i)} e_j/\alpha$.

W load on the host of all m modules of the program are assigned to it.
$W = \sum_{i=1}^{m} e_i/\alpha$.

$\mathcal{S}_i$ load on a satellite if the subtree rooted at module i is assigned to it.
$\mathcal{S}_i = \sum_{j \in \mathcal{T}(i)} e_j + c_i$.

$\mathcal{H}_T$ total load on the host.
$\mathcal{H}_T = W - \sum_{p=1}^{n} (\mathcal{H}_{\tau(p)} - c_{\tau(p)})$.

Our assignment is specified by the vector $\tau(p), 1 \le p \le n$ that specifies the root node of the subtree resident on each satellite. Given this vector, the weight of an assignment is

$$\max(\max_{1 \le p \le n} \mathcal{S}_p, \mathcal{H}_T). \tag{3}$$

5.2 Condensing Trees

Theorem 4. Consider a tree that has a partition of weight ω, and in which there exists a node f with a child $g \in \mathcal{C}(f)$ such that at least one of the following two inequalities holds.

$$c_g \geq c_f + \sum_{i \in \{\mathcal{T}(f) - \mathcal{T}(g)\}} e_i \tag{4}$$

$$c_g \geq e_g + \sum_{i \in \mathcal{C}(g)} c_i \tag{5}$$

Then this tree will continue to have a partition of weight $\leq \omega$ if we merge nodes f and g.

Proof. The given partition of weight ω must assign f to the host and g to a satellite, otherwise the proof is trivial. When we merge f and g, the condensed node $f + g$ can be assigned either to the host or to a satellite.

If inequality (4) holds assign the condensed node to a satellite (see Figure 8). In this case the load on the satellite before condensation was

$$\mathcal{S}_g = \sum_{i \in \mathcal{T}(g)} e_i + c_g.$$

After condensation it is

$$\mathcal{S}_f = \sum_{i \in \mathcal{T}(f)} e_i + c_f.$$

The decrease is

$$\mathcal{S}_g - \mathcal{S}_f = c_g - \sum_{i \in \{\mathcal{T}(f) - \mathcal{T}(g)\}} e_i - c_f.$$

This quantity is non-negative because of (4).

The load on the host will decrease by at least $c_g + e_f/\alpha - c_f$ which is also non-negative because of (4).

If inequality (5) holds then assign the condensed node to the host. In this case the load on the satellite before condensation is again $\mathcal{S}_g$ (given above). After condensation, part of this load will go to the load and part will be distributed over several additional satellites (so that there is now one

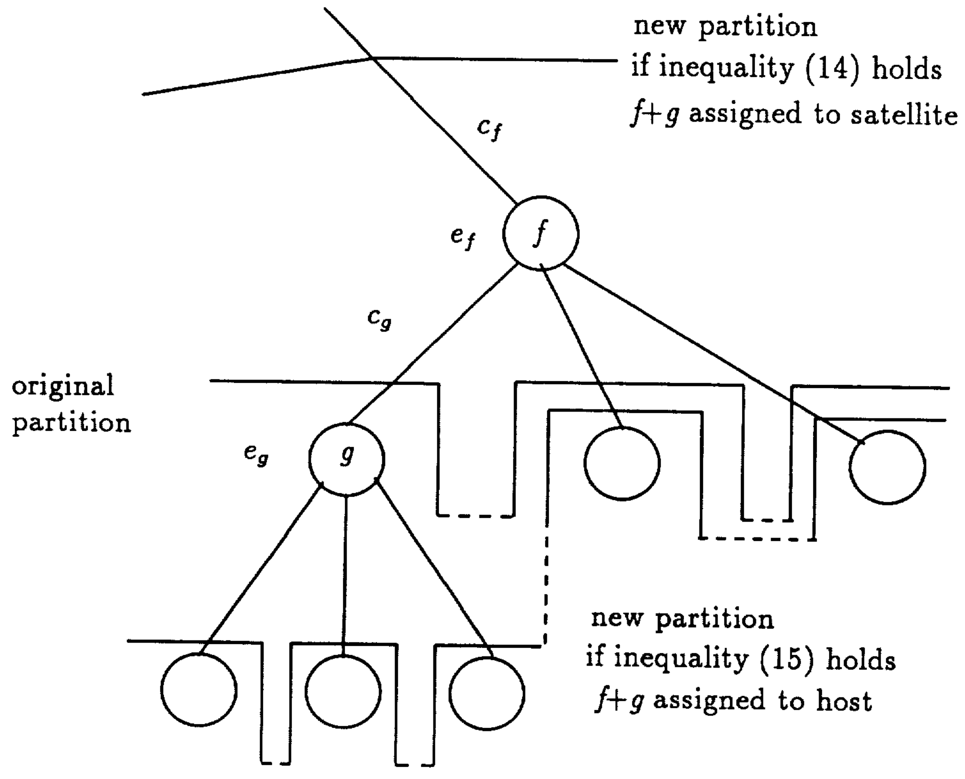

Figure 8: Illustration of proof of Theorem 4

satellite for each child of g). Each of the new satellite loads will be at least $e_g + c_g - \min_{i \in \mathcal{C}(g)} c_i$ less than the original satellite load $\mathcal{S}_g$. This quantity is non-negative because $c_g \geq \sum_{i \in \mathcal{C}(g)} c_i$. The load on the host will increase by $e_g/\alpha + \sum_{i \in \mathcal{C}(g)} c_i$ and decrease by c_g. The quantity $c_g - e_g/\alpha - \sum_{i \in \mathcal{C}(g)} c_i$ is non-negative because of (5) and because $\alpha > 1$.

In at least one case the loads on the satellites and on the host decrease or remain unchanged. Thus if there is a partition of weight ω before condensation there will be a partition of weight $\leq \omega$ after condensation.□

5.3 Monotonic Trees

A procedure **condense_tree** can be derived from Theorem 4. This procedure goes through the tree and merges together all nodes f and g, where g is the child of f, which satisfy (4) or (5). A tree to which this procedure has been applied is called a *condensed* tree. Condensed trees are monotonic in a fashion analogous to condensed chains.

Theorem 5. In a tree that has been transformed by applying procedure **condense_tree**, $\mathcal{S}_g \leq \mathcal{S}_f$ for all f, g such that f is the father of g.

Proof. By contradiction. Suppose $\mathcal{S}_g > \mathcal{S}_f$. Then

$$\begin{aligned} \sum_{i \in \mathcal{T}(g)} e_i + c_g &> \sum_{i \in \mathcal{T}(f)} e_i + c_f \\ c_g &> \sum_{i \in \{\mathcal{T}(f) - \mathcal{T}(g)\}} e_i + c_f \end{aligned} \qquad (6)$$

But this is impossible since all f, g that satisfy (6) are eliminated by procedure **condense_tree**.□

This theorem assures us that, once a tree has been condensed, the load caused by a subtree cannot exceed the load caused by a containing subtree.

5.4 Probing Function

A probing function can now be designed to evaluate if there exists a partition of the condensed tree that assigns no more than ω weight to each of the satellites or to the host. This probing function proceeds upwards from the leaves of the tree and stops each time it identifies a maximal subtree that has weight $\leq \omega$.When all such subtrees have been identified, the load on the host can be calculated. If this is less than ω, the function returns **true**. Since the condensed tree is monotonic, i.e. the weight of a subtree is always $\leq$ the weight of a containing subtree, this probing function needs to look at each node only once and will return an answer in $O(m)$ time.

5.5 Partitioning Algorithm

There are m potential subtrees in our condensed tree. Their weights can be evaluated in $O(m)$ time and sorted in $O(m \log m)$ time. Following this, we can can carry out a binary search over this list to find the optimal value of ω. This takes $O(\log m)$ probes each of cost $O(m)$. The overall time for this algorithm is thus $O(m \log m)$. This is better than Bokhari's exact algorithm, which takes $O(m^2 \log m)$ time and Iqbal's approximation algorithm which is $O(m \log(W/\epsilon))$.

6 Conclusions

The general problem of partitioning a program over a multiple computer system has so far eluded an efficient solution. Prior research by Bokhari [2], Iqbal [6] and Nicol & O'Hallaron [8] has reported a succession of efficient algorithms for the restricted class of chain- or tree- structured programs. In the present paper we have described a condensation approach that preprocesses the given chain or tree in linear time. This condensation makes the chain or tree monotonic and permits fast algorithms to be used in the search for the optimal partition.

For the problem of partitioning an m module chain over a chain of n processors, we have improved Iqbal's $O(mn \log(W/\epsilon))$ approximation algorithm to $O(m \log(W/\epsilon))$. Our exact algorithm for this problem is $O(mn \log m)$ which compares with Nicol & O'Hallaron's $O(m^2 n)$ exact algorithm and their $O(mn \log m)$ bounded cost algorithm. Our exact algorithm makes no assumptions about costs.

When faced with the problem of partitioning n chains of m modules each over a host-satellite system, we have developed an $O(n \log m \log(W/\epsilon))$ approximation algorithm that is better than Iqbal's $O(mn \log(W/\epsilon))$ solution. Our exact solution is $O(mn \log n)$, which is equal to Nicol & O'Hallaron's algorithm (which again assumes bounded costs).

Finally, for the problem of partitioning a single tree-structured program over a host-satellite system, we have improved Bokhari's $O(m^2 \log m)$ exact solution to $O(m \log m)$. The following table summarizes this discussion.

Problem		Linear Array	Host Satellite	Tree
Bokhari	exact	m^3n	$m^2n\log m$	$m^2\log m$
N-O'H	exact	m^2n	$mn\log m$	—
Iqbal	approximate	$mn\log(W/\epsilon)$	$mn\log(W/\epsilon)$	$m\log(W/\epsilon)$
N-O'H	bounded costs	$mn\log m$	$mn\log n$	—
Improved Results				
Approximate		$m\log(W/\epsilon)$	$n\log m\log(W/\epsilon)$	—
Exact		$mn\log m$	$mn\log n$	$m\log m$

N-O'H=Nicol & O'Hallaron.

Acknowledgements

We wish to thank R. G. Voigt and K. E. Durrani for their encouragement of this research. We are grateful to D. M. Nicol for many useful discussions and for his comments on an earlier draft of this paper.

References

[1] Shahid H. Bokhari. *Assignment problems in parallel and distributed computing.* Kluwer, Boston, 1987.

[2] Shahid H. Bokhari. Partitioning problems in parallel, pipelined and distributed computing. *IEEE Transactions on Computers*, C-37(1):48–57, January, 1988.

[3] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, C-30:207–214, March 1981.

[4] S. Borkar et al. iWARP: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing 88*, pages 330–339, 1988.

[5] G. Bolch et al. A multiprocessor system for simulating data transmission systems (MUPSI). *Microprocessing and Microprogramming*, 12(5):267–277, December 1983.

[6] M. Ashraf Iqbal. Approximate algorithms for partitioning and assignment problems. Technical Report 86-40, ICASE, June 1986. NASA Contractor Report 178130.

[7] M. Ashraf Iqbal, Joel H. Saltz, and Shahid H. Bokhari. A comparative analysis of static and dynamic load balancing strategies. *Proceedings of the 1986 International Conference on Parallel Processing*, pages 1040–1047, August 1986.

[8] David M. Nicol and David R. O'Hallaron. Improved algorithms for mapping pipelined and parallel computations. *IEEE Transactions on Computers*, to appear, 1990. An earlier version is available as ICASE Report 88-2, NASA Contractor Report No. 181655.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-182073 ICASE Report No. 90-49	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle EFFICIENT ALGORITHMS FOR A CLASS OF PARTITIONING PROBLEMS		5. Report Date July 1990
		6. Performing Organization Code
7. Author(s) M. Ashraf Iqbal Shahid H. Bokhari		8. Performing Organization Report No. 90-49
		10. Work Unit No. 505-90-21-01
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		11. Contract or Grant No. NAS1-18605
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 236665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Submitted to IEEE Transactions on Parallel and Distributed Computers

Final Report

16. Abstract

We address the problem of optimally partitioning the modules of chain-or tree-like tasks over chain-structured or host-satellite multiple computer systems. This important class of problems includes many signal processing and industrial control applications. Prior research has resulted in a succession of faster exact and approximate algorithms for these problems.

We describe polynomial exact and approximate algorithms for this class that are better than any of the previously reported algorithms. Our approach is based on a preprocessing step that condenses the given chain or tree structured task into a monotonic chain or tree. The partitioning of this monotonic take can then be carried out using fast search techniques.

17. Key Words (Suggested by Author(s)) Approximation algorithms, assignments, chains, distributed computing, host-satellite systems, load balancing, partitioning, parallel processing, trees	18. Distribution Statement 61 - Computer Programming and Software 62 - Computer Systems Unclassified - Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 26	22. Price A03