1994

Partitioning of image processing tasks on heterogeneous computer systems

M. Ashraf Iqbal

Saeed Iqbal

Muhammad Shaaban

Partitioning of Image Processing Tasks on Heterogeneous Computer Systems

M. Ashraf Iqbal*, Saeed Iqbal
Department of Electrical Engineering
University of Engineering & Technology,
Lahore, Pakistan

Muhammad E. Shaaban
Department of Electrical Engineering-Systems
University of Southern California,
Los Angeles

Abstract

Many computer vision tasks can be decomposed into a set of subtasks which are by their nature heterogeneous. By partitioning such task onto different machines that communicate via high speed links, each level or stage of processing can be executed simultaneously on the machine to which it is best suited. A fundamental problem with heterogeneous computing, however, is the difficulty of optimally partitioning an application program across the machines. In this paper we address the problem of partitioning a chain or a tree structured parallel or pipelined program over a two processor heterogeneous system and show that it is possible to approximately solve this problem. The algorithm, presented in this paper, is based on a *fully polynomial time approximation scheme.*

1 Introduction

Vision processing is classified into a number of categories or levels, all of these levels can utilize a tremendous amount of parallelism, and the levels themselves can operate in parallel [8]. It has been observed that machines in the SIMD class are well suited for early processing of raw images-often termed as low level processing [8, 15]. High level vision tasks such as image understanding, recognition, and symbolic processing exhibit coarse-grain or medium-grain MIMD type characteristics. Thus by exploiting the different features and capabilities of a heterogeneous environment, higher levels of performance can be attained than is possible by using any single type of parallel machine.

Many computer vision tasks, such as image understanding, pattern recognition, dynamic scene analysis, etc., can be expressed as pipelined algorithms [9]. A common requirement in such a system is to apply repeatedly a fixed sequence of operations (or transforms) to an essentially unending series of images. By partitioning the application task into different machines that communicate via high speed links, each step or stage of pipeline processing can be executed simultaneously on the machine to which it is best suited. The maximum rate of processing is now determined by the processor that takes the longest amount of time to perform the application task, known as the *bottleneck* processor [1].

The following problem then emerges. Given a set of m subtasks connected in some fashion, and a heterogeneous computer system consisting of different machines, find the assignment of subtasks to processors that minimizes the load on the most heavily loaded processor. If the number of processors are only two and the program is serial , this problem can be solved efficiently using the network flow approach pioneered by Stone [12]. If the program is serial and the interconnection structure of the modules is tree-like, it is possible to solve it for any number of processors using a shortest tree algorithm [2].

If the modules are executable in parallel, it is very difficult to find efficiently the optimal solution, given a variety of criteria for optimality. This is because the problem is computationally equivalent to one or the other of the notorious NP-complete graph partitioning or multiprocessor scheduling problems [1]. This explains why most of the work in this field focused on heuristic techniques [3, 13]. It has been shown by Bokhari [1] that a chain structured parallel or pipelined program can be partitioned optimally over a chain or ring of processors. Nicol [11] and Iqbal [4] have improved the complexity of earlier algorithms for partitioning chain structured problems with restrictions on the type of mappings and/or on the weights assigned to different modules. Iqbal [6] has also solved a number of partitioning problems in heterogeneous environments. All these researchers worked under the constraint that each processor has a contiguous subchain of program modules assigned to it.

In this paper we address the problem of partitioning a chain or a tree structured parallel or pipelined program over a two processor heterogeneous system and show that it is possible to approximately solve this problem.

*Part of this research was conducted while the author was a Fulbright Scholar at the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, from Sept. 1992 to June 1993.

0-8186-5592-5/94 $3.00

The algorithm, presented in this paper, is based on a *fully polynomial time approximation scheme*, both in the size of the problem and in $\frac{1}{\epsilon}$, where ϵ is the relative error bound for the approximate scheme.

We start the paper by discussing in Section 2 the Heterogeneous Computing (HC) Paradigm. Section 3 addresses the problem of finding the near optimal partition of a chain-structured parallel or pipelined program over a two processor system. Our solution technique involves creating a doubly weighted assignment graph.

We address the problem of partitioning a tree structured parallel or pipelined program in Section 4. We conclude this paper with a discussion in Section 5.

2 Heterogeneous Computing for Vision

Many scientific and engineering applications have diverse computational requirements. Thus any one single type of machine may perform poorly on such applications. For an application consisting of subtasks of various computational requirements, a suite of heterogeneous machines is likely to provide superior performance. Heterogeneous Computing(HC) has been recently proposed as a novel paradigm to exploit the existing hardware and advances in networking to lead to feasible solutions to complex problems [8].

2.1 The Heterogeneous Computing Paradigm

HC is a computing paradigm in which an application is run in an environment that incorporates several autonomous high performance parallel machines. These machines, providing different types of parallelism, are capable of communicating over a high-speed intelligent interconnection network to cooperate in an application by representing the solution as a set of tasks to be executed.

Partitioning problems for two processor heterogeneous systems have recently become a focus of interest. The Minnesota Supercomputer Center(MSC) [14], and The Distributed High Speed Computing (DHSC) environment at the Pittsburg Supercomputing Center [10], are two examples of research sites working in a two processor heterogeneous environment.

2.2 The Mapping Problem

Algorithms used for symbolic computing tasks in image understanding applications exhibit different computation characteristics and processing requirements. Algorithms developed by the vision community have addressed each step of an integrated vision system separately and hence the conversion overheads involved in switching from one level to another level have not been understood. Data decomposition and conversion overheads involved in communicating among such machines adds further delays. Such a scenario, if not carefully analyzed, can drastically degrade the overall performance of the system. In order to optimally partition and map tasks in an integrated vision system over processors of a heterogeneous computer system we should take care of the following: The heterogeneous nature of each subtask, the type of machines available in the heterogeneous environment, and data decomposition and conversion overheads involving communicating subtasks which reside on different types of machines.

3 Partitioning Chain Structured Problems

We discuss here a simple algorithm for finding an optimal partitioning of a chain structured parallel program, belonging to an integrated vision system, over a dual-processor heterogeneous system. A chain structured program is made up of m modules numbered $t_1 ... t_m$, and has an intercommunication pattern such that module t_i is connected only to modules t_{i+1} and t_{i-1}. The optimal assignment of subchains to the two processors is influenced by the time of execution of module t_i on processor $x(y)$ designated by $w_{xi}(w_{yi})$ time required for communication between module t_i, and t_{i+1} designated by c_i, provided the two modules are assigned to different processors.

The partitioning problem can be expressed in the following manner: Given a set of m modules connected in a chain like fashion, and a two processor heterogeneous system find the assignment of subchains of modules to processors that minimizes the $max(W_x, W_y)$, where $W_x(W_y)$ is the load on processor $x(y)$. Our approach to the solution of this problem is to first draw up a doubly weighted assignment graph. A path in this graph corresponds to the assignment of subsequences of modules to processors.

3.1 The Doubly Weighted Assignment Graph

In this section we discuss the concept of a doubly weighted assignment graph G. There are two weights associated with each edge of this graph: Δ_x weight corresponds to additional (or incremental) load assigned to the x processor, and Δ_y weight corresponds to additional (or incremental) load assigned to the y processor.

There are two kinds of *sum* weights associated with

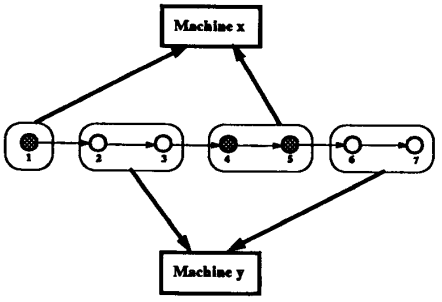

Figure 1: A seven-module chain mapped onto a two processor heterogeneous system.

each path P, one is the familiar sum of all $\Delta_x(e_i)$, the other is the sum of all $\Delta_y(e_i)$. A path for which the maximum of $(\sum \Delta_x(e_i), \sum \Delta_y(e_i))$ is minimal, corresponds to the optimal assignment.

The Structure of the Assignment Graph

There are two layers in the graph G, the x layer corresponds to the x processor, while the y layer corresponds to the y processor. The jth node in the $x(y)$ layer of this graph corresponds to module t_j in the application program and is thus represented by $t_j^x(t_j^y)$. The start node is connected to every node in the x layer as well as in the y layer except $t_1^x(t_1^y)$. Every node $t_j^x(t_j^y)$ in the $x(y)$ layer is connected to each node $t_k^y(t_k^x)$ in the $y(x)$ layer provided $j < k \leq m$. Every node in the $x(y)$ layer is connected to the end node.

The Labelling Technique

An $edge(start, t_j^x)$, $1 \leq j \leq m$, corresponds to a partitioning p in which modules $t_1 ... t_{j-1}$ are assigned to processor x, while at least module t_j is assigned to processor y. The Δ_x and Δ_y weights of this edge are given below:

$$\Delta_x(start, t_j^x) = \sum_{i=1}^{j-1} w_{xi} + c_{j-1},\ \Delta_y(start, t_j^x) = c_{j-1}$$

An $edge(t_j^x, t_k^y)$, where $j < k \leq m$, corresponds to a partitioning in which modules $t_j ... t_{k-1}$ are assigned to processor y, while at least module t_k is assigned to processor x. The Δ_x and Δ_y weights associated with this edge are given below:

$$\Delta_x(t_j^x, t_k^y) = c_{k-1},\ \Delta_y(t_j^x, t_k^y) = \sum_{j}^{k-1} w_{yi} + c_{k-1}$$

Example 1

Consider the seven-module chain shown in Fig. 1. Assume that $w_{xi} = 1, w_{yi} = 4$ if $i = 1, 4$, and 5, $w_{xi} = 4, w_{yi} = 1$ if $i = 2, 3, 6$, and 7, and c_i=2, where $1 \leq i \leq 6$. In simple the black modules have an execution cost of 1(4) on processor $x(y)$, while the white modules have execution costs equal to 4(1) on processor $x(y)$, and the communication cost is assumed to be uniform equal to 2. The assignment graph corresponding to the seven-module chain is shown in Fig. 2. The partitioning of the chain structured parallel program, shown in Fig. 1, is represented by a path (shown in bold) between the start node and the end node in the assignment graph of Fig. 2.

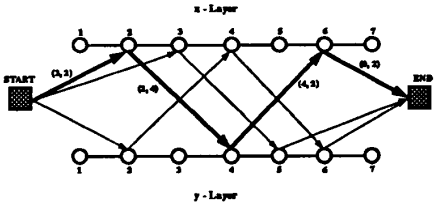

Figure 2: The layered graph and several paths between the start and the end node for a problem with seven modules.

3.2 The Approximate Assignment Scheme

It has been shown in the last section that the incoming degree of a node t_i^x in the assignment graph is 2^{i-2}. Thus the total number of distinct paths between the start node and t_i^x would be precisely equal to 2^{i-2}. Each such path corresponds to an assignment in which the modules $t_1 ... t_{i-1}$ have already been assigned in some fashion while the assignment of the remaining modules, $t_i ... t_m$, is yet to be made. Let $< W_x(i), W_y(i) >$ represent the ordered pair associated with a path terminating at node t_i^x, where $W_x(i)(W_y(i))$ represents the sum of all $\Delta_x(\Delta_y)$ weights of all the edges in the path between the start node and t_i^x. Thus $W_x(i)(W_y(i))$ is in fact the load assigned to processor $x(y)$ due to the assignment of modules $t_1 ... t_{i-1}$ over the two-processor system. It is important to note that each path between start node and t_i^x may have a distinct ordered pair and thus, in general, each node t_i^x can have as many as 2^{i-2} different $< W_x(i), W_y(i) >$ ordered pairs.

An upper bound on the maximum load, which can be assigned to processor x in any assignment, is when all modules are processed sequentially on processor x assuming that $w_{xi} > 0$, where $1 \leq i \leq m$. If this upper bound is represented by W_T then $W_T = \sum_{i=1}^{m} w_{xi}$.

Let us resolve W_T to an accuracy of e In other words $W_x(i)$ is restricted to have only $\frac{W_T}{e}$ distinct values in the range of zero and W_T. This operation of *restricting* the number of possible paths between the start

node and any other node in the x or y layer in the assignment graph is performed as follows. We look at a selected node p, where p can be either t_i^x or t_i^y in the assignment graph. Out of every incoming path from the start node to node p we reject every incoming path P_2 in comparison with an incoming path P_1 provided $W_{x1}(i) \leq W_{x2}(i)$ and $W_{y1}(i) \leq W_{y2}(i)$, Where $< W_{x1}(i), W_{y1}(i) > (< W_{x2}(i), W_{y2}(i) >)$ is the ordered pair associated with the incoming path $P_1(P_2)$ at node p. Out of all the remaining paths between the start node and node p for which the actual value of $W_x(i)$ is in between two successive permissible levels, we select the one with minimal value of $W_y(i)$, and reject all others (see Lemma 1).

There would be at the most $\frac{W_T}{\epsilon}$ paths between the start node and any node in the x or y layer after the application of the procedure *Limit*. Thus there will be a maximum of $2m\frac{W_T}{\epsilon}$ distinct paths between the start node and the end node in the assignment graph. A path in which $max(W_x, W_y)$ is minimal can thus easily be found in time proportional to $\frac{2mW_T}{\epsilon}$, but the total time would be limited by the complexity of the procedure LIMIT which is $O(\frac{m^2W_T}{\epsilon})$ as just given above. If the relative error bound for the Approximate Scheme is ϵ then the time complexity of our algorithm is bounded by $O(m^3(\frac{W_T}{\epsilon}))$.

Lemma 1

Assume that P_1 and P_2 are two paths between the start node and the end node in the assignment graph with the following properties:

- Path P_1 as well as P_2 consists of two subpaths, one is from the start node to t_i^x, and the other is from t_i^x to the end node.
- The subpath from the start node to t_i^x in path P_1 is different from the corresponding subpath in path P_2. Let $< W_{x1}(i), W_{y1}(i) >$ represents the ordered pair associated with the subpath in P_1. Similarly the subpath in P_2 between the start node and t_i^x is represented by $< W_{x2}(i), W_{y2}(i) >$.
- Paths P_1 and P_2 share a common subpath from node t_i^x to the end node.

Under the conditions stated above the load on the bottleneck processor corresponding to path(or partitioning) P_1 would always be less than or equal to the corresponding load in partitioning P_2 provided:

$$W_{x1}(i) \leq W_{x2}(i) \text{ and } W_{y1}(i) \leq W_{y2}(i)$$

Example 2

Consider the assignment graphs (Fig. 3) for the seven-module chain of Fig. 1. P_1 and P_2 are two paths between the start and the end nodes in the assignment graph, and are shown in Fig. 3(bottom) and (top) respectively. The corresponding partitionings of the chain are also shown in Fig. 3. It is important to note that the subpath in P_1 from the start node to node 3 in the x layer is different from the corresponding subpath in P_2. The two paths, however, share a common subpath from node 3 in the x layer to the end node. The ordered pair $< W_x(3), W_y(3) >$ associated with each subpath is also indicated in Fig. 3. As $W_{x1}(3) < W_{x2}(3)$, and $W_{y1}(3) < W_{y2}(3)$, it can be deduced that the load assigned to processor x or y in the entire path (or partitioning) P_1 would be less than the corresponding loads in path P_2. Thus when we look at the two ordered pairs associated with the two paths terminating at module 3 in the x layer, we immediately recognize that one of the subpaths, when extended to the end node, would always result in a costly assignment (as compared to the other path), and thus should be rejected immediately.

Suppose there is a third path (P_3) between the start and the end node in the assignment graph of Fig. 3, and it also passes through node 3 in the x layer. Assume that $W_{x3}(3)$=6.999, and $W_{y3}(3)$=4. Thus $W_{x1}(3)$=$W_{x3}(3)$+0.001, and $W_{y1}(3) < W_{y3}(3)$. It is obvious that if we reject path P_3 in comparison with path P_1 then the total load assigned to processor x in P_1 would be larger than the load assigned to x in P_3 by at most 0.007, while the load assigned to processor y in P_1 would be less than the corresponding load in P_3. The above statement will be true provided $W_{x1}(i) \leq W_{x3}(i) + 0.001$, and $W_{y1}(i) < W_{y3}(i)$ for each common node i between path P_1 and P_3 in the assignment graph.

4 Partitioning Tree Structured Problems

Our approach to the solution of this problem is very similar to the one described in the previous section for partitioning chain structured problems. A tree is similar to a chain in the sense that by removing a single edge it can be divided into two parts. There are, however, important differences between the two structures, and these should be kept in mind while designing the new algorithm.

We first traverse the given tree and place consecutive labels on the nodes of the tree visited according to the procedure *Label* described below. The resulting path

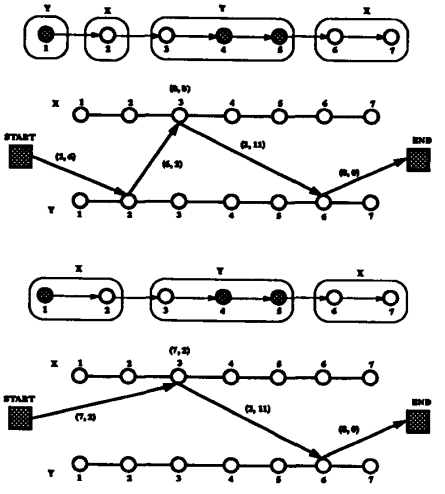

Figure 3: Path P_1 (bottom) and path P_2 (top) consists of two subpaths, one is from the start node to t_3^x, and the other is from t_3^x to the end node.

(of traversal) is treated as a chain of modules which is then partitioned by drawing a doubly weighted assignment graph. In order to partition the tree structured parallel program using our previous techniques of partitioning a chain, we introduce the concept of *critical* nodes in the next section. It has been shown that the time complexity of the modified partitioning algorithm is proportional to $2^{c_{max}}$, c_{max} is the maximum number of critical nodes affecting a node in the assignment graph. By using an intelligent scheme for traversing (and labelling) the nodes of the tree we limit the value of c_{max} to $\log_2 m$.

4.1 The Labelling of the Tree

The modules of a tree structured problem are labelled by the following procedure *Label*. A tree is divided into two parts by cutting an edge(j,k), where nodes j and k are adjacent to each other [5, 7]. One half of the tree, which includes node j, is known as $subtree_{jk}$, while the other half, which includes node k, is called $subtree_{kj}$. The number of nodes in a subtree is denoted by $nodes[subtree]$. The key idea behind this labelling technique is as follows: The next node to be labelled would be a node k adjacent to node j such that $nodes[subtree_{kj}]$ is minimal. The label of node k would be $Last + 1$.

Procedure *Label* (A tree of m nodes, $m > 2$)
begin

1. Start with any leaf node i, and label it with 1, i.e., $Label(i) := 1$, $Last := 1$, $j := i$.
2. Find a node k adjacent to node j, which is not yet labelled; $Label(k) := Last + 1$; $j := k$; $Last := Label(k)$. If all the m nodes are labelled then *return*.
3. Let d denote the degree of node j;
   (a) If $d = 1$ then goto step (5).
   (b) If $d = 2$ then find a node k adjacent to node j, which is not yet labelled.
   (c) If $d > 2$ then find a node k out of all unlabelled nodes adjacent to node j such that $nodes[subtree_{kj}]$ is minimal.

   If you do not find such a node k then goto step (5).
4. $Label(k) := Last + 1$; $j := k$; $Last := Label(k)$; If all the m nodes are labelled then *return* otherwise goto step (3).
5. Backtrack to node b, where b is the last node labelled with degree larger than two; $j := b$; goto step (3).

end.

4.2 The Doubly Weighted Assignment Graph

There are two layers in the graph G, the x layer corresponds to the x processor, while the y layer corresponds to the y processor. The jth node in the $x(y)$ layer of this graph corresponds to module with label j in the application program and is thus represented by $t_j^x(t_j^y)$. The start node is connected to every node in the x layer as well as in the y layer except $t_1^x(t_1^y)$. Every node $t_j^x(t_j^y)$ in the $x(y)$ layer is connected to each node $t_k^y(t_k^x)$ in the $y(x)$ layer provided $j < k \leq m$. Every node in the $x(y)$ layer is connected to the end node. A path in this graph between the start and the end node corresponds to an assignment of subsequences of modules to processors.

An $edge(start, t_j^x)$, $1 < j \leq m$, corresponds to a partitioning p in which modules with labels $1...j-1$, in the application program, are assigned to processor x, while at least module j is assigned to processor y. The x and y weights of this edge are given below:

$x(start, t_j^x) = \sum_{i=1}^{j-1} w_{xi} + c_j$, and $y(start, t_j^x) = c_j$

An $edge(t_j^x, t_k^y)$, where $j < k \leq m$, corresponds to a partitioning in which modules $j+1...k$ are assigned to processor x or y in the following manner:

1. For each i, where $j+1 \leq i \leq k-1$, starting from $i = j+1$, find an adjacent node which has already been assigned to either the x processor or the y

processor. If the adjacent node is assigned to $x(y)$ processor then module i should also be assigned to $x(y)$ processor.

2. Find a node adjacent to node k which has already been assigned to a processor. Node k is assigned to $x(y)$ if the adjacent node is assigned to processor $y(x)$.

The x and y weights associated with this edge can then be evaluated.

An $edge(t_j^x, end)$ i.e. an edge between a node t_j^x and the end node in the assignment graph corresponds to partitioning in which nodes $j+1...m$ are assigned to processor x or y in the following manner: For each node i, where $j+1 \leq i \leq m$, starting from $i = j+1$, find an adjacent node which has already been assigned to either the x processor or the y processor. If the adjacent node is assigned to $x(y)$ processor then node i should also be assigned to $x(y)$ processor i.e. the two nodes should be assigned to the same processor. The corresponding x and y weights associated with this edge can then be found out.

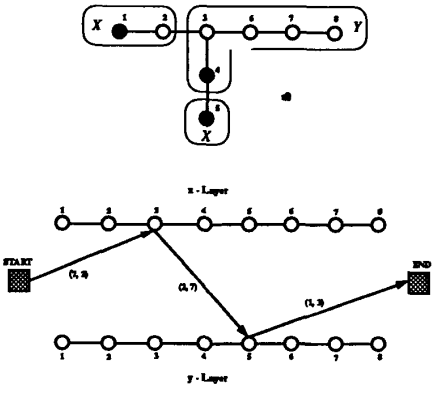

Figure 4: An eight-module tree mapped onto a two processor heterogeneous system.

Example 3

Consider the tree structured parallel program shown in Fig. 4. Assume that the black modules have an execution cost of 1(4) on processor $x(y)$, while the white modules have execution costs equal to 4(1) on processor $x(y)$, and the communication cost is assumed to be uniform equal to 2. The tree is labelled using the labelling technique described in the previous section. Note that once the tree is labelled it can be treated as a chain structure with certain special or critical nodes, defined in the next section. The assignment graph corresponding to the eight-module tree is shown in Fig. 4(bottom).

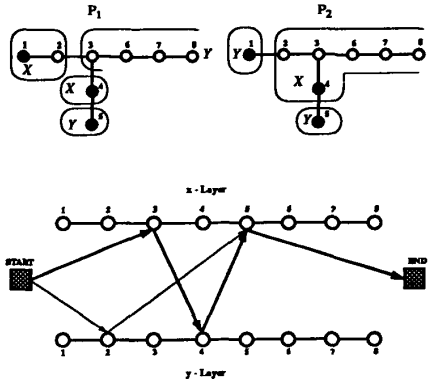

Figure 5: An eight-module tree is partitioned onto a two processor heterogeneous system. Partitionings P_1, P_2, and the corresponding paths in the doubly weighted assignment graph are also shown.

4.3 The Notion of Critical Nodes

In the doubly weighted assignment graph designed for a chain structured parallel program we have noted that the load on the bottleneck processor corresponding to path P_1 would always be less than or equal to the corresponding load in partitioning P_2 provided $W_{x1}(i) \leq W_{x2}(i)$ and $W_{y1}(i) \leq W_{y2}(i)$, where partitionings (or paths) P_1 and P_2 are defined in Lemma 1. This useful property was exploited to restrict the total number of distinct paths between the start node and any node $t_i^x(t_i^y)$, thereby drastically reducing the size of our search space. Thus it became possible for us to efficiently find an approximate partition of the chain structured parallel or pipelined program with the guarantee that the maximum percentage error in the load assigned to the bottleneck processor is within a fixed bound. The above mentioned property for a chain structured program, as described in Lemma 1, does not hold as such in the doubly weighted assignment graph designed for a tree structured parallel program. Consider, for example, the tree structured parallel program shown in Fig. 5. Partitionings P_1, P_2, and the corresponding paths in the doubly weighted assignment graph are also shown in the figure. Note that the subpath in P_1, shown in bold, from the start node to node 5 in the x layer is different from the corresponding subpath in P_2. The two paths, however, share a common subpath from node 5 in the x layer to the end node.

It is important to appreciate here that the load assigned to processor x or y in the entire path (or par-

titioning) P_1 would not be less than the corresponding loads in path P_2 even if $W_{x1}(5) < W_{x2}(5)$, and $W_{y1}(5) < W_{y2}(5)$. This is because node 3 in the tree structured parallel program is assigned to processor y in partitioning P_1 while it is assigned to processor x in partitioning P_2. This node is critical for node 5 because it decides the future of the nodes with labels 6, 7, and 8, which are yet unassigned. It is important to note that these nodes (with labels 6, 7, and 8) are assigned to processor y in P_1, while they are assigned to processor x in P_2 in spite of the fact that both paths P_1 and P_2 share a common subpath from node 5 in the x layer to the end node.

If, however, node 3 in the tree is assigned to the same processor in both partitionings P_1 and P_2 then the load assigned to the bottleneck processor corresponding to P_1 would always be less than or equal to the corresponding load in P_2 provided $W_{x1}(5) \leq W_{x2}(5)$ and $W_{y1}(5) \leq W_{y2}(5)$. This extra constraint requires us to modify Lemma 1 and we do so by presenting the concept of *critical nodes*. Note that node 3 was a critical node for node 5 of the tree structured parallel program as shown in Fig. 5.

Now consider the tree structured program, same as shown in Fig. 5, with two different partitionings P_3 and P_4 as shown in Fig. 6. The corresponding paths in the doubly weighted assignment graph are also shown in the bottom of the figure. It is important to appreciate here that the load assigned to processor x or y in the entire path (or partitioning) P_3 would be less than the corresponding loads in path P_4 if $W_{x3}(7) < W_{x4}(7)$, and $W_{y3}(7) < W_{y4}(7)$. This is because there is no critical node for node 7 in the doubly weighted assignment graph and consequently Lemma 1 holds for such a node.

Once a tree is labelled using the labelling procedure described before, the critical nodes, and the nodes affected by these critical nodes, are determined using the following simple procedure described in Lemma 2.

Lemma 2

1. For each node (or module), labelled as i, with degree 3 or more in the tree, find a node adjacent to i with a label j such that j is maximal.

2. Node i is a *critical node* affecting only those nodes which are not adjacent to node i and have labels from $i+2$ to $j-1$.

Lemma 3

If a tree is labelled using the labelling procedure, described earlier in this section, then the maximum number of critical nodes affecting any node in the tree would be at the most equal to $\log m$.

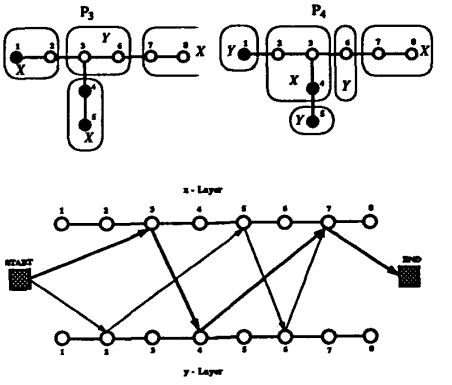

Figure 6: The eight-module tree, as shown in Fig. 5, is partitioned onto a two processor heterogeneous system. Partitionings P_3, P_4, and the corresponding paths in the doubly weighted assignment graph are also shown.

The Approximate Assignment Scheme

An upper bound on the maximum load, which can be assigned to processor x in any assignment, is when all modules are processed sequentially on processor x assuming that $w_{xi} > 0$, where $1 \leq i \leq m$. If this upper bound is represented by W_T then $W_T = \sum_{i=1}^{m} w_{xi}$

Let us resolve W_T to an accuracy of e i.e. two successive permissible levels for the load on processor x are separated by e. In other words $W_x(i)$ is restricted to have only $\frac{W_T}{e}$ distinct values in the range of zero and W_T. The approximate partitioning of the tree structured parallel program can now be found us as follows:

1. Label the tree structured parallel program using the procedure described earlier in this section.

2. For each node i, $1 \leq i \leq m$, find if it is affected by any critical nodes(s).

3. Look at each incoming path from the start node to node p where p can be either t_i^x or t_i^y in the assignment graph. Select those paths or in which all critical nodes, influencing node i, are assigned to processor x or y, and node i of the tree structured program is assigned to processor x or y in a similar manner.

Out of all paths between the start node and node p for which the actual value of $W_x(i)$ is in between two successive permissible levels, select the one with minimal value of $W_y(i)$, and reject all others. This restricts

the number of ordered pairs associated with each node t_i^x (or t_i^y) to $O(\frac{W_T 2^{\log m}}{\epsilon})$.

There will be a maximum of $O(m^2 \frac{W_T}{\epsilon})$ distinct paths between the start node and the end node in the assignment graph. A path in which $max(W_x, W_y)$ is minimal can thus easily be found in time proportional to $O(\frac{m^2 W_T}{\epsilon})$, with a guarantee that the maximum difference between the load on the bottleneck processor in the approximate assignment and the one in the optimal assignment is at the most equal to me. If the relative error bound for the Approximate Scheme is ε then the time complexity of our algorithm is bounded by $O(m^4(\frac{W_T}{\epsilon}))$.

5 Conclusions

In this paper we have described an efficient algorithm which can partition a chain or a tree structured application program consisting of several heterogeneous modules onto a two processor system. Our approach takes into account the heterogeneous nature of each module as well as the conversion overheads involved when modules residing on different types of machines communicate with each other. 5diverse modules onto a dual processor heterogeneous system. Heterogeneous sites consisting of two processor systems can directly benefit from this research, and can use our *crossover* strategy which allows application programs to be optimally partitioned and run simultaneously on more than one supercomputer at a time. It is possible to extend this approach to a three or four processor heterogeneous computer systems, and currently we are working to apply similar techniques to related problems with less restricted structures.

Acknowledgment

This work was partially funded by the University Grants Commission, Pakistan. We acknowledge the motivation and encouragement provided by Viktor Prasanna, Mary Eshaghian, Ikram-ul-Haq Dar, Abdul Hameed, and Syed Nazeer.

References

[1] S. Bokhari. Partitioning problem in parallel, pipelined, and distributed computing. *IEEE Trans. on Computer*, pages 48–57, January 1988.

[2] S.H. Bokhari. Dual processor scheduling with dynamic reassignments. *IEEE Trans. Software Eng.*, SE-5, July 1979.

[3] C. Castro and S. Yalamanchili. Partitioning algorithms for a class of application specific multiprocessor architectures. In *Proc. IPPS Workshop on Heterogeneous Processing*, April 1993.

[4] M. Ashraf Iqbal. Efficient algorithms for partitioning problems. In *ICPP*, 1991.

[5] M. Ashraf Iqbal. *Mapping and Assignment Problems in Multiple Computer Systems.* PhD thesis, Engineering University, Lahore, Pakistan, 1991. Dept. of Electrical Eng.

[6] M. Ashraf Iqbal. Partitioning problems for hetrogeneous computer systems. In *Proc. IPPS Workshop on Heterogeneous Processing*, April 1993.

[7] M.Ashraf Iqbal. Partitioning a tree structured problem on a heterogeneous computer system. Technical report, Dept. of Electrical Eng., Engineering University, Lahore, Pakistan, 1993.

[8] A. Khokhar, V.K. Prasanna, M. Shaaban, and C. Wang. Heterogeneous supercomputing: Problems and issues. In *Proc. Workshop on Heterogeneous Processing*, March 1992.

[9] V.K. Prasanna Kumar, editor. *Parallel Architectures and Algorithms for Image Understanding.* Academic Press, 1991.

[10] J. Mahdavi, G. Huntoon, and M. Mathis. Developement of a HIPPI-based distributed supercomputing environment at the pittsburgh supercomputing center. In *Proc. Workshop on Heterogeneous Processing*, pages 93–96, Mar. 1992.

[11] D. Nicol and D. O'Hallaron. Improved algorithms for mapping pipelined and parallel computations. *IEEE Trans., Computers*, 40(3).

[12] H. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. on Software Eng.*, SE-3:85–93, January 1977.

[13] L. Tao, B. Narahari, and C. Zhao. Heuristics for mapping parallel computaions to heterogeneous parallel architechtures. In *Proc. IPPS Workshop on Heterogeneous Processing*, April 1993.

[14] R. Vetter, D. Du, and A. Klietz. Network supercomputing: Experiment with a CRAY-2 to CM-2 HIPPI connection. In *Proc. Workshop on Heterogeneous Processing*, pages 87–92, Mar. 1992.

[15] C. Weems. Image understanding: A driving application for research in heterogeneous parallel processing. In *Proc. IPPS Workshop on Heterogeneous Processing*, April 1993.