AN ANALYSIS OF THE WORKLOAD SCHEDULING ALGORITHM EXTENDED TO SCHEDULE GENERAL TREES

Technical Report 94-08

Kenneth J. Boe, Jr.

Department of Computer Science and Engineering
Auburn University
Auburn University, Alabama 36849-5347

June 20, 1994
An Analysis of the Workload Scheduling Algorithm
Extended to Schedule General Trees

For the Degree
Master of Computer Science and Engineering

Presented to
Dr. Carolyn McCreary
Department of Computer Science and Engineering
Auburn University
1994

Written by
Kenneth J. Boe, Jr.
1 INTRODUCTION

The purpose of this Master of Computer Science and Engineering project is to implement the extension to general trees of the workload scheduling algorithm presented by Yahui Zhu and Dr. Carolyn McCreary in their paper "Optimal and Near Optimal Tree Scheduling for Parallel Systems." [McZhu92] It is the intent of Dr. McCreary and her research team to use the workload scheduling algorithm to improve the performance and results of their own parallel processing research. It is the goal of this author to show if the extension to general trees works, and if not, how to possibly improve upon the extension so as to be of use to Dr. McCreary and her team.

The workload scheduling algorithm for binary trees was implemented and improved by Jeanne C. Stone during her Master of Computer Science and Engineering Project for Dr. McCreary. Ms. Stone found that in Yahui Zhu's original algorithm [Zhu90][Zhu91] local greedy decisions did not result in the optimal time for a graph. Ms. Stone found that through saving many of the workloads generated at a node for use in creating workloads later in the tree better results were generated than through Zhu's Greedy algorithm.

In their paper, Yahui Zhu and Dr. McCreary presented a Greedy algorithm for scheduling a general tree and suggested some methods for determining the optimal solution. These have been implemented in a new Unix based X application named "xwkld". This new implementation will allow much larger graphs to be scheduled than in the previous PC based implementation.

The organization of this paper is as follows. In Section 2, the problem of scheduling a sequential program onto a multiprocessor machine will be discussed. In Section 3, a discussion of previous workload scheduling work will be presented. In Section 4, binary trees then the extension of to general trees will be discussed. In Section 5, results of the
newly adapted algorithms will be presented. Appendix A contains a user's manual for "xwkld", the new X application which runs on a SPARCstation. Appendix B contains a description of "gtree," a general tree generator. Appendix C contains examples of binary tree scheduling and appendix D contains examples of some general tree executions.
1 INTRODUCTION

The purpose of this Master of Computer Science and Engineering project is to implement the extension to general trees of the workload scheduling algorithm presented by Yahui Zhu and Dr. Carolyn McCreary in their paper "Optimal and Near Optimal Tree Scheduling for Parallel Systems." [McZhu92] It is the intent of Dr. McCreary and her research team to use the workload scheduling algorithm to improve the performance and results of their own parallel processing research. It is the goal of this author to show if the extension to general trees works, and if not, how to possibly improve upon the extension so as to be of use to Dr. McCreary and her team.

The workload scheduling algorithm for binary trees was implemented and improved by Jeanne C. Stone during her Master of Computer Science and Engineering Project for Dr. McCreary. Ms. Stone found that in Yahui Zhu's original algorithm [Zhu90][Zhu91] local greedy decisions did not result in the optimal time for a graph. Ms. Stone found that through saving many of the workloads generated at a node for use in creating workloads later in the tree better results were generated than through Zhu's Greedy algorithm.

In their paper, Yahui Zhu and Dr. McCreary presented a Greedy algorithm for scheduling a general tree and suggested some methods for determining the optimal solution. These have been implemented in a new Unix based X application named "xwkld". This new implementation will allow much larger graphs to be scheduled than in the previous PC based implementation.

The organization of this paper is as follows. In Section 2, the problem of scheduling a sequential program onto a multiprocessor machine will be discussed. In Section 3, a discussion of previous workload scheduling work will be presented. In Section 4, binary trees then the extension of to general trees will be discussed. In Section 5, results of the
1 Introduction ................................................................................................................. 1
2 The Scheduling Problem ............................................................................................ 3
3 Literature Review ..................................................................................................... 4
4 Scheduling Trees ...................................................................................................... 6
   4.1 Scheduling Binary Trees .................................................................................. 6
      4.1.1 Binary Tree Scheduling Terminology ..................................................... 6
      4.1.2 Zhu Greedy Algorithm ........................................................................... 11
      4.1.3 Lighter Workload Pruning ................................................................. 12
   Figure 4.1.3.1 Candidate Workloads .................................................................. 12
      4.1.4 Dominating Workload Pruning ........................................................... 13
   4.2 Scheduling General Trees .............................................................................. 14
      4.2.1 McCreary-Zhu Local Greedy Decisions ............................................. 14
      4.2.2 Determining Workloads in General Tree .......................................... 17
      4.2.3 Zhu Local Greedy Decisions .............................................................. 21
      4.2.4 Lighter Workload Pruning in the General Tree .................................. 21
5 Results ....................................................................................................................... 22
   5.1 General Tree Results ...................................................................................... 22
   5.2 General Tree Verification ............................................................................... 24
6 References ............................................................................................................... 26
2 THE SCHEDULING PROBLEM

The goal of parallel processing is to improve the run time of a sequential program by dividing it into executable segments which can then be run on a multiprocessor machine. A great deal of research effort has gone into trying to find the optimal method of decomposing a sequential program into modules, scheduling the modules onto processors, and maintain the module dependencies. This research will result in the "best communication cost schedule," which will result in shortest execution time.

In a sequential program, the dependencies are handled by the very nature of the sequential process, but when the sequential code is separated into multiple modules, the intermodule dependencies are not accounted for by process execution. The dependencies which exist between executable segments can be represented as a directed acyclic graph (DAG). The nodes of the graph represent the execution segments and the arcs represent the dependencies between code segments. These dependencies must be accounted for when scheduling the sections of the sequential program.

Once the DAG has been created, time can be devoted to scheduling the modules onto processors and enforcing the dependencies between the modules. This is the point where the communication delay versus execution time tradeoff enters consideration. Every module could be scheduled on a different processor, but if communication time between two processors is large it may be more efficient to schedule modules with large communication times on the same processor while scheduling modules with short communication times in parallel.
3 LITERATURE REVIEW

Three types of scheduling algorithms have been suggested. The first entitled "Critical Path Heuristics" attempted to shorten the longest execution path. The second, "List Scheduling Heuristics," uses greedy heuristics, performs module duplication to reduce communication costs, and schedules the modules in a certain order. Carolyn McCreary and Helen Gill published a paper in 1989 entitled "Automatic Determination of Grain Size for Efficient Parallel Processing," which used graph decomposition theory to schedule the modules. This algorithm determines parallelism by only analyzing the graph. Yahui Zhu arrived at his Workload Scheduling paradigm separately, which was first published as a technical paper at North Dakota State University, [Zhu90] and later in the Proceedings of the 1991 International Conference on Parallel Processing, [Zhu91].

Dr. Zhu's workload scheduling process for binary trees involves four steps for each non-leaf node of the graph. First, sum the start time, execution time, and communication time of each ancestor node; the node that generates the earlier start time will be the parallelized candidate workload. Second, the ancestor workloads will be aggregated to create a second candidate workload. Third, the earlier start time generated for the workloads in the first and second steps is selected. Lastly, if the selected workload is the aggregated workload from step, two use the workload for the node being scheduled. Otherwise create a workload of the node not parallelized in step one and the start time from the parallelized workload. The leaf nodes of the graph are initially assumed to have a start time of zero for use in determining their predecessor node's workload.

Jeanne Stone implemented Dr. Zhu's algorithm for workload scheduling for her Master of Computer Science and Engineering project. The original algorithm consisted of Greedy decisions which eliminates all but the best workload at each node in the processing tree. An optimal algorithm keeps the workloads generated at each node and returns the best
starting time obtained at the root of the processing tree. This change resulted in superior schedules as shown in her paper.

With Ms. Stone's implementation in hand Dr. McCreary began work on pruning techniques for use in reducing the number of workloads which must be considered at each node in the processing tree. Kim Smith implemented these pruning techniques for Dr. McCreary. The pruning techniques used were lighter loads, parallel decision pruning, deletion of separate choice, and local Greedy decisions. Lighter workloads compare two workloads starting times and total busy time. If one workload has an earlier start time and its total busy time is less then it is lighter. Parallel decision pruning considers only the workload with the earlier start time instead of both parallel workloads. Deleting the separate choice reduces the number of workloads to be considered. Making local Greedy decisions also reduces the number of workloads to be considered. The last two pruning techniques may result in the best start not being found, but both methods can significantly reduce scheduling overhead.

With the results from Ms. Stone's and Ms. Smith's work Dr. McCreary and Yahui Zhu collaborated on a paper entitled "Optimal and Near Optimal Tree Scheduling for Parallel Systems" [McZhu92] in which they presented these results. It was also in this paper that the extension to general trees was suggested. It is on the extension to general trees that this paper and accompanying X application is based. This idea will be explored in the next section.
4 SCHEDULING TREES

This section describes scheduling binary and general trees. Section 4.1 is a discussion of binary tree scheduling, including terminology and algorithms used. This discussion provides the background information needed for understanding of section 4.2, General Tree Scheduling. The general tree section will explain new concepts needed and the algorithms used.

4.1 SCHEDULING BINARY TREES

A discussion of scheduling binary trees must begin with the defining of terms needed to provide complete explanations of algorithms used to perform the scheduling. Section 4.1.1 will present the items and concepts used in describing the scheduling algorithms. In section 4.1.2 the Greedy algorithm originally presented by Dr. Zhu will be explained and examples discussed. Section 4.1.3 will be concerned with Lighter Workload Pruning and an example, and section 4.1.4 will present Dominating Workload Pruning and an example.

4.1.1 BINARY TREE SCHEDULING TERMINOLOGY

Figure 4.1.1.1 shows an inverted, labelled binary tree, which will be referenced in all subsections of Binary Tree Scheduling (4.1). The tree is drawn inverted to show order of execution. The nodes at the top of the diagram execute first, followed by the next level down, and so forth, until the root of the tree is reached. Note that nodes occupying the same level may execute simultaneously. The labelled circles in the figure represent the nodes which must be scheduled. The pertinent information associated with a node is the task time and the communication time. As the key in the figure shows the numbers labelled with a 't' are the task times. The task time indicates how long the node takes to execute. The numbers labeled with a 'c' are the communication times of the node. The
communication time is the time cost of transmitting data from the processor on which the node executes to another processor, this is also referred to communication cost.

![Figure 4.1.1.1 Example Binary Tree](image)

The relationship of nodes in the tree must also be specified in order to provide a useful means of referring to nodes during scheduling. Since the tree is inverted the leaf nodes are at the top of the diagram, and the root is at the bottom. The root node has a communication time of zero because its execution ends the execution of the tree. The nodes in the first level above the root are called the root’s ancestors, e.g. in figure 4.1.1.1 nodes e and f are the ancestors of node g, the root. More specifically node e is the left ancestor of g, and node f is the right ancestor of g. Left and right are determined simply by position in relation to the nodes' successor. The successor is the node which must follow by one level
in the execution order, e.g. node g is the successor of both e and f because both e and f must execute before g can execute. Another example: e is the successor of a and b, and a is the left ancestor of e, while b is the right ancestor of e. Ancestors which share the same successor are siblings. Lastly, e is the root of the subtree consisting of nodes a, b, and e, see figure 4.1.1.2.

Figure 4.1.1.2 Subtree of Binary Tree

With the elements of the tree and their characteristics defined, attention will now be turned to the concept of the workload. A workload attempts to record the busy and idle times of a processing element. An attempt is made to use the idle time to perform other processing instead of wasting the time. More formally, a workload of node u with k (>0)
time intervals, denoted by $W(u)$, is defined as a list of $k$ time intervals $[a(i), b(i)]$, and a starting time $s(u)$, i.e.,

$$W(u) = ([a(1), b(1)], ..., [(a(k), b(k)], s(u)),$$

where

$$a(1) = 0, a(i) \leq b(i) \text{ and } b(i) < a(i+1) \text{ for all } i, \text{ and } b(k) \leq s(u).$$

For example, the workload $([0, 42], [73, 111], 111)$ represents the workload generated for the root of the binary tree in figure 4.1.1.1 by the Greedy algorithm (the determination of this workload will be discussed in section 4.1.2). Note that the processor on which the root will be scheduled will be busy from time zero until time 42, then be idle until time 73, at that time the processor will be busy until time 111. At time 111 the root node's task can begin execution, which will result in a finish time for the schedule of 147. Thus the finish time is the time at which all execution is complete. Another workload of interest is the workload of all leaf nodes. The workload is $([0, 0], 0)$, meaning that the node can begin execution immediately.

The formation of new workloads for non-leaf nodes can be done in the following ways: parallel left, parallel right, aggregate, and separate. Figure 4.1.1.3 shows the node combinations for each of the workload creation methods, as well as how the nodes are placed on processors. Each node or nodes within a closed loop are scheduled on the same processor.

Creating a workload using the parallel right method results in the left ancestor being scheduled on the same processor as its successor node, and the right ancestor being scheduled on a different processor. The workload for node $e$ is $([0, 40], 44)$. The start time of 44 results from the execution time of node $b$ plus the communication time of node $b$. Thus node $e$ can begin execution after node $a$ executes, and node $b$ executes and transmits its data. Parallel left works in a similar fashion, with the scheduling of the left and right ancestors reversed, and results in a workload for node $e$ of $([0, 14], 65)$. 
The aggregation method of creating workload schedules the left and right ancestors and the successor node on the same processor. The resulting workload ([0, 54], 54), node e can begin its execution immediately after both ancestors complete execution because the needed data resides on the processor with node e.

![Diagram of workload creation methods]

**Figure 4.1.1.2 Workload Creation Methods**

Lastly, the separate method of creating workloads, schedules both ancestors and the successor on different processors. The workload for node e is therefore ([0,0], 65). Note that the start time is the same as for the parallel left case. This is due to the fact that both
ancestors must transmit their data to node e before it can begin execution. Since node e
does not have all needed data until the node a completes transmission node e must wait until
the later start time.

The workload creation methods and also the scheduling algorithms rely on a few
restrictions: 1) an unlimited number of processors, 2) a node will not be interrupted during
execution, 3) communication between processors is error free, 4) data is sent at the end of a
module's execution, and 5) only binary tree directed acyclic graphs are used.

4.1.2 ZHU GREEDY ALGORITHM

This algorithm uses local greedy decisions to schedule the tree. The algorithm first
creates the candidate workloads for a node. These workloads are created using the parallel
left, parallel right, and aggregate methods. The resulting candidate workload with the
earliest start time is selected. Figure 4.1.2.1 shows the candidate workloads for node e.

```
+-----+-----+
|  t:40 |  t:14 |
|  a    |  b    |
+-----+-----+
|  c:25 |  c:31 |
+-----+-----+
|  e    |  t:16 |
```

Parallel Left : ([0,14], 65)
Parallel Right: ([0,40], 44)
Aggregate : ([0,54], 54)

Figure 4.1.2.1 Local Candidate Workloads

Thus the parallel right workload is chosen for node e since its start time is 44 compared to
65 for parallel left, and 54 for aggregate. The separate creation method is not used because
it will always result in a slower start time than either the left or right parallel method.
4.1.3 LIGHTER WORKLOAD PRUNING

An alternative to choosing only the workload with the best start time is to keep all generated workloads. The separate workload now becomes a useful candidate. If this option were not included an optimal solution would not be possible for all trees. Figure 4.1.3.1 shows the four workloads created for node e of the example binary tree. If the four candidate workloads were kept for node e and node f of the example tree, node g would then have sixteen workloads. While keeping all workloads would result in an optimal solution, the number of workloads for the root of the tree would be four raised to the n, where n is the number of levels in the tree. Lighter workload pruning discards candidate workloads based on two criteria: the start time and the total busy time of the workload.

![Diagram of workloads](image)

**Figure 4.1.3.1 Candidate Workloads**

The definition of lighter workloads is:

Given two workloads, \( W1 = ([a(1), b(1)], ..., [a(p), b(p)], s1) \) and \( W2 = ([c(1), d(1)], ..., [c(q), d(q)], s2) \), \( W1 \) is lighter than \( W2 \) if and only if \( \forall i \leq q \sum (d(i) - c(i)) \) and \( s1 \leq s2 \), i.e. if the total busy time of \( W1 \) is less than or equal to
the total busy time of W2 and the start time of W1 is less than or equal to the start time of W2, W1 is lighter than W2.

As the workloads are created the new workload is compared to the workloads created previously. If the new workload is lighter than a workload in the list the old workload is thrown away an the new workload put in its place. If a workload in the list is lighter than the new workload, the new workload is discarded. Lastly, if neither case occurs the new workload is simply added to the workload list. The workloads of node e after the lighter workloads have been pruned are in figure 3.1.3.2.

<table>
<thead>
<tr>
<th>Parallel Right: ([0, 40], 44)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate                    : ([0, 0], 65)</td>
</tr>
</tbody>
</table>

Figure 3.1.3.2 Workloads of Node e after Lighter Workload Pruning

4.1.4 DOMINATING WORKLOAD PRUNING

Another method of reducing the number of workloads for a node is dominating workload pruning. A workload is said to dominate another workload if given two workloads, W1 = ([a(1), b(1)], ..., [a(p), b(p)], s1) and W2 = ([c(1), d(1)], ..., [c(q), d(q)], s2), the intervals of W1 are subintervals of the intervals of W2, and the start time of W1 is less than or equal to the start time of W2. That is for each interval of W1, [a(i), b(i)], there is an interval of W2, [c(j), d(j)], in which c(j) <= a(i) and d(j) >= b(i). Figure 4.1.4.1 shows the workloads for node e of the example tree after dominating workload pruning has been used.
The complete schedules produced by the algorithms discussed in this section are presented in Appendix C.

4.2 SCHEDULING GENERAL TREES

This section describes the new concepts and the algorithms used to schedule general trees, and a discussion of determining the number of possible workloads for a node in the graph. The algorithms which will be described are the local greedy decisions proposed by Dr. McCreary and Dr. Zhu in their joint paper [McZhu92], the local Greedy decisions proposed by Dr. Zhu [Zhu90, Zhu91] extended to general trees, and lighter workload pruning extended to general trees described by Dr. McCreary and Dr. Zhu [McZhu92]. Figure 4.2.1 shows the example general tree which will be used throughout this section.

4.2.1 MCCREARY-ZHU LOCAL GREEDY DECISIONS

This algorithm requires a new workload measure, $z$, which is the sum of the start time, task time, and communication time. Figure 4.2.1.1 shows the components of the $z$ value for the ancestors of node 2 and the calculated $z$ values. Once these values have been calculated the ancestors of node 2 should be sorted in non-increasing order, resulting in an order of Node 10, Node 12 Node 9, and Node 11. Thus the formula for $z$ is:

\[
\text{Parallel Right: } ([0, 40], 44) \\
\text{Aggregate: } ([0, 54], 54) \\
\text{Separate: } ([0, 0], 65)
\]
Figure 4.2.1 Example General Tree
\[ z(i) = s(i) + t(i) + c(i), \]

where \( s \) is the start time of the workload, \( t \) is the task time of the node, and \( c \) is the communication time of the node.

![Diagram](image)

**Figure 4.2.1.1 Z Values**

The \( z \) value provides an indication of whether a node should be aggregated or scheduled onto a new processor. The higher the value of \( z \) the more it should be aggregated with its successor or the ancestor before it in the list. The algorithm works by

![Diagram](image)

**Figure 4.2.1.2 Schedule and Workload of Node 2**

16
aggregating ancestors until the start time of the workload exceeds the z value of the next ancestor in the sorted list. When the z value is exceeded the next ancestor is placed on another processor, and ancestors are aggregated with the new workload until the z value is exceeded again. The scheduling process continues until all ancestors have been scheduled. Figure 4.2.1.2 shows the aggregations created by the algorithm for the ancestors of node 2 and the workload for the aggregations.

4.2.2 DETERMINING WORKLOADS IN GENERAL TREE

In the binary tree there were four workload types to consider: aggregate, parallel_left, parallel_right, and separate. In the general tree the aggregate type remains the case in which all ancestors and the node which is having its workloads created share the same processor.

Figure 4.2.2.1 Aggregations with the Successor Node - Three Ancestors
The parallel cases are still combinations of aggregation and parallelization; in the binary tree one ancestor was aggregated with its successor and one ancestor was put on another processor. However, the names parallel_left and parallel_right do not remain useful. They fail to describe all the ways in which parallelization can occur when there are more than two ancestors. Figure 4.2.2.1 shows the different aggregations with the successor which can be made with three ancestors.

As can be seen from figure 4.2.2.1 there are more than two parallelization possibilities, and for cases a, b, and c there are two ways to place the non-aggregated ancestors onto processors, see figure 4.2.2.2 for possible schedules of part (a).

![Diagram](image)

Figure 4.2.2.2 Parallelizations of Non-Aggregated Ancestors

Thus for any aggregation which leaves more than one ancestor unaggregated all parallelization combinations must be created. Table 4.2.2.1 shows the parallelizations resulting from a node with four ancestors with only one ancestor aggregated.

<table>
<thead>
<tr>
<th>Agg w/ Succ</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>bc</td>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>bd</td>
<td></td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>cd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bcd</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2.2.1 Parallelizations of Three Non-Aggregated Ancestors
The separate case remains the same in the general terms of there being no node sharing a processor with its successor. The separate workload can be created along with the aggregate and parallel workloads. When the aggregate workload is created also create a workload which has the aggregation on another processor. Similarly, when creating the parallel workloads move whatever nodes are aggregated with the successor to another processor as a group. In both cases, the result is that the successor is on a processor alone. Table 4.2.2.2 shows the changes for anode with three ancestors.

<table>
<thead>
<tr>
<th>Agg</th>
<th>Aggregate and Parallel</th>
<th>Separate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>abc</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>ac</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>ab</td>
<td>c</td>
</tr>
<tr>
<td>ab</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>ac</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>bc</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2.2.2 Separate Workloads Created from Others

This table shows some duplication in the separate workloads. Note that the duplication in the aggregate and parallel section does not duplicate workloads because a different ancestor is aggregated each time, thus removing its communication time from consideration. The duplicate combinations in the separate workloads cause a small amount of overhead from repeating the workload. However, since the separate workloads are created at the same time as the parallel or aggregate workloads the much larger overhead or recreating combinations is avoided. The overhead that occurs is considerably less than would be caused by keeping all different combinations and checking the combination list each time one is generated.
In the binary tree scheduling, the four workload types had the same arrangement of ancestors every time the type was used. For example, parallel_left always meant that the right ancestors was aggregated and the left ancestors was scheduled onto a separate processor. The start time was the larger of the right ancestor's completion time or the left ancestors execution time plus communication time. When there are more than two ancestors the placement of ancestors on other processors is not the same every time the first ancestors is aggregated (figure 4.2.2.2). This makes keeping workloads of the parallelized ancestors necessary in order to know which workloads are to be scheduled on the same processor. The workloads for the parallelized ancestors are called components. The workload for the successor of the parallelized nodes is the main component, and the main and other components together form the complete workload. Examples of complete workloads for node 2 of the example general tree are presented in figure 4.2.2.3.

<table>
<thead>
<tr>
<th>Aggregate</th>
<th>Parallel</th>
<th>Separate</th>
</tr>
</thead>
<tbody>
<tr>
<td>([0, 81], 81)</td>
<td>([0, 33], 43)</td>
<td>([0, 0], 55)</td>
</tr>
<tr>
<td>([0, 27], 43)</td>
<td>([0, 33], 55)</td>
<td>([0, 27], 43)</td>
</tr>
<tr>
<td>([0, 21], 33)</td>
<td>([0, 21], 31)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2.2.3 Complete Workloads

When the complete workloads of the general tree are being created the components are created first. The component group start time is the worst parallelized start time of the individual components. In figure 4.2.2.3 the component group start time of part (a) is _. Only the component group with the best start time is kept because the components only affect the main component's start time and are not used again after the main component is generated. The main component is used when making decisions or pruning complete workloads.
4.2.3 ZHU LOCAL GREEDY DECISIONS

This Greedy algorithm is the truest extension to general trees of the Greedy algorithm used in the binary tree application. All candidate workloads, described in section 4.2.2, are created. As the workloads are created the start time of the main component, of the new workload is compared to the start of the current best complete workload. If the new complete workload's start time is earlier than the current best complete workload's start time, the old one is discarded and the new one is kept. Otherwise, the old complete workload is thrown away and the new one is kept. This process is done for each non-leaf node of the tree, ending with the root of the tree.

4.2.4 LIGHTER WORKLOAD PRUNING IN THE GENERAL TREE

The definition of a lighter workload is the same for the general tree as it was for the binary tree. At each non-leaf node of the tree, as all complete workloads are created the main component of the new complete workload is compared to the main component of the other workloads. If the the main component is lighter then a main component in the node's workload list, the list member is discarded and the new complete workload put in its place. If a workload in the list has a main component which is lighter than the new workload's main component, the new workload is thrown away. If neither lightness case occurs, the new workload is appended to the list.
5 RESULTS

This section presents the results generated by the applications written for this project. First, the results from executions of the general tree application will be presented and discussed. The second subsection the general tree output will be validated by comparing the results to those of the binary tree application when the same binary tree is scheduled by both executables.

5.1 GENERAL TREE RESULTS

Figure 5.1.1 shows a general tree which was scheduled by the general tree scheduling application. Figure 4.2.1 shows the tree labelled with communication and task times. The scheduling of this graph resulted in finish times of 257 time units for the McCreary-Zhu Greedy scheduling algorithm, 153 time units for the Zhu Local Greedy Decisions, and 153 time units for the Optimal algorithm with Lighter Workload pruning. See Table 5.1.1 for the schedules created for these finish times.

![Diagram](image)

Figure 5.1.1

An analysis of the execution times required by each algorithm gives a good view of the complexity of each algorithm. The McCreary-Zhu Greedy algorithm executed in 0.016666
seconds. The Zhu Local Greedy Decisions executed in 1.033292 seconds, which is
directly caused by the algorithm checking each possible workload combination at every
node in the graph. The lighter workload method completed its scheduling in 3.099876
seconds. This time increase is due to the algorithm keeping multiple workloads for each
node in the graph, thus exponentially increasing the possible workloads for its predecessor.
The fact that the lighter workload pruning resulted in the same finish time is a factor of the
size of the graph. In a smaller graph the probability is much higher that a greedy algorithm

<table>
<thead>
<tr>
<th>McCreary-Zhu</th>
<th>Zhu Local Greedy</th>
<th>Lighter Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc 0</td>
<td>Node 8</td>
<td>Proc 0</td>
</tr>
<tr>
<td>Proc 1</td>
<td>Node 9 Node 11</td>
<td>Node 7 Node 0</td>
</tr>
<tr>
<td>Proc 2</td>
<td>Node 13</td>
<td>Node 1 Node 6</td>
</tr>
<tr>
<td>Proc 3</td>
<td>Node 18 Node 19</td>
<td>Node 2 Node 8</td>
</tr>
<tr>
<td>Proc 4</td>
<td>Node 15 Node 4 Node 14</td>
<td>Node 21 Node 5</td>
</tr>
<tr>
<td>Proc 5</td>
<td>Node 10 Node 12 Node 2</td>
<td>Node 16 Node 3</td>
</tr>
<tr>
<td>Proc 6</td>
<td>Node 21</td>
<td>Node 8 Node 16</td>
</tr>
<tr>
<td>Node 20</td>
<td>Proc 10 Node 18</td>
<td>Proc 11 Node 19</td>
</tr>
<tr>
<td>Node 5</td>
<td>Proc 12 Node 10</td>
<td>Node 1 Node 19</td>
</tr>
<tr>
<td>Node 6</td>
<td>Proc 14 Node 9</td>
<td>Proc 15 Node 11</td>
</tr>
<tr>
<td>Node 1</td>
<td>Proc 15 Node 11</td>
<td>Proc 16 Node 11</td>
</tr>
</tbody>
</table>

Table 5.1.1

will result in the same solution as the lighter workload pruning method. The finish times,
execution times, and schedules produced are precisely the type of results expected.
5.2 GENERAL TREE VERIFICATION

In order to show that the general tree scheduling application was producing useful graph schedules, it was tested against the binary tree scheduling application. Each application scheduled the same four level binary tree using the multiple run facility provided by the programs. The communication cost to execution cost ratio for the first run was the default ratio of 1.0. The second run used a ratio of 5.0, and the third a ratio of 10.0. Appendix A, Xwkld User’s Manual, describes scheduling runs and their setup.

<table>
<thead>
<tr>
<th></th>
<th>BINARY</th>
<th>GENERAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>Execution</td>
<td>0.016666</td>
<td>0.033332</td>
</tr>
<tr>
<td>Run 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>342</td>
<td>342</td>
</tr>
<tr>
<td>Execution</td>
<td>7.066384</td>
<td>0.083330</td>
</tr>
<tr>
<td>Run 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>438</td>
<td>438</td>
</tr>
<tr>
<td>Execution</td>
<td>18.049278</td>
<td>0.266656</td>
</tr>
<tr>
<td>Lighter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>Execution</td>
<td>0.016666</td>
<td>0.066664</td>
</tr>
<tr>
<td>Run 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>272</td>
<td>272</td>
</tr>
<tr>
<td>Execution</td>
<td>7.083050</td>
<td>0.166660</td>
</tr>
<tr>
<td>Run 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>326</td>
<td>326</td>
</tr>
<tr>
<td>Execution</td>
<td>18.082610</td>
<td>0.266656</td>
</tr>
</tbody>
</table>

Table 5.2.1

Table 5.2.1 shows the results of the comparable algorithms from the three scheduling runs.

The applications have two algorithms which can be compared for purpose of showing appropriate results. The Greedy and Lighter Workload Pruning algorithms of the binary tree version relate directly to the Zhu Greedy and Lighter Workload Pruning algorithms of the general tree application. The McCreary-Zhu Greedy algorithm used in scheduling
general trees does not examine all workloads for the node to be scheduled, and the Dominating Workload Pruning of the binary application does not have an equivalent in the general tree version.

As this table shows, the general tree scheduling application finds the same finish times as the binary scheduling program. The surprising discovery is that the general tree takes much less time to schedule a graph when more aggregation of nodes occurs. The processor schedules also matched as did the finish times for the Greedy algorithms, but the processor schedules for the lighter workload pruning allocated the processor differently even though they resulted in the same finish times, see Appendix D for the schedules.
6 REFERENCES


