Stony Brook University

Signature Search and Computing Cost Optimization in Distributed Load Networks

A Dissertation Presented

by **Zhongwen Ying**

to

The Graduate School

in Partial Fulfillment of the requirements

for the Degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Stony Brook University

August 2014

Stony Brook University

The Graduate School

Zhongwen Ying

We, the dissertation committe for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

Thomas G. Robertazzi

Professor in department of Electrical and Computer Engineering

K. Wendy Tang

Associate Professor in department of Electrical and Computer Engineering

Peter Milder

Associate Professor in department of Electrical and Computer Engineering

Esther M. Arkin

Professor in department of Applied Mathematics and Statistics

This dissertation is accepted by the Graduate School

Charles Taber

Dean of the Graduate School

Abstract of the Dissertation

Signature Search and Computing Cost Optimization in Distributed Load Networks

by **Zhongwen Ying**

Doctor of Philosophy

in

Electrical and Computer Engineering

Stony Brook University

2014

A signature is a data pattern of interest in a large data file or set of large data files. Such signatures that need to be found arise in applications such as DNA sequence analysis, network intrusion detection, biometrics, large scientific experiments, speech recognition and sensor networks. Related to this is string matching.

More specifically we envision a problem where long linear data files (i.e flat files) contain multiple signatures that are to be found using a multiplicity of processors (parallel processor).

This paper evaluates the performance of finding signatures in files residing

in the nodes of parallel processors configured as trees, two dimensional meshes and hypercubes. We assume various combinations of sequential and parallel searching. A unique feature of this work is that it is assumed that data is pre-loaded onto processors, as may occur in practice, thus load distribution time need not be accounted for. Elegant expressions are found for average signature searching time and speedup, and graphical results are provided.

Expressions for speedup for divisible load scheduling for a four node cyclic network are found. Load distribution when there are multiple paths for such distribution have received little attention. In this first study speedup is calculated for a wide variety of scheduling assumptions. Comparisons between the different scheduling policies are made.

An integrated optimization problem which involves minimizing the monetary cost of operation and minimizing solution time (makespan) which has computer utility-like applications is investigated. A divisible load model of a bus interconnection network is considered. The trade-offs between monetary cost and solution time is found via a heuristic algorithm. The algorithm is improved compared to an earlier algorithm. Trade-off issues are examined. In this paper we confirm the work of Shaklevich.

Contents

List of Figures

Acknowledgement

It would not be possible to write this doctoral thesis without the help of so many people. Here I can only mention a small part of them.

First and foremost, I would like give my earnest thanks to my advisor, Prof Thomas Robertazzi for supporting me these years, both financially and intellectually. He is the most easygoing and one of the smartest people I know. As an advisor, he is a role model that teaches me how to do research with patience, carefulness and hard work. He not only gave me freedom to manage the time and do research without pressure but also instructive suggestions when I faced difficulties. He expands my understanding and ability to do this research which is a tremendous help for my future career.

I thank my girlfriend Yiyi Xue. Her cherished love, consistent patience, and considerate heart sustained me in getting through this journey of study.

I thank my friends, Kai Wang, Zheming Zhang, Yang Liu, Yunlong Wang, Li Geng, Zhe Shen, who are members in the Cosine Laboratory, (Communication, Signal Processing, and Networking), Stony Brook University. Their invaluable suggestions and encouragement helped me work on this dissertation. I am very grateful to each of them. In addition, I thank Professor Petar M. Djuric for his encouragement and for providing facilities and the needs for the Cosine Laboratory. This invaluable support makes this laboratory much more convenient and pleasant for research work.

I am grateful to my parents: Xiaoxing Ying and Meifen Ge. They give me unconditional support throughout my life and have cherished with me every

achievement I have made. Without their encouragement and support, my study would never have been done in Stony Brook. I thank Zhigen Lin for his friendship and his perennially encouraging emails that have been enriching the colour of my life.

Chapter 1

Introduction

Parallel computing for a multiprocessors system and grid computing or distributed computing for a network system are becoming prevalent because the infrastructure of networked systems is increasingly widespread and parallel data processing technology has become more popular. This dissertation deals with the speedup analysis for a multiple level tree and mesh networks, scalable design for a cyclic network, and cost analysis for a bus interconnection network.

The increasing ubiquity of parallel processors makes parallel computing or grid computing more convenient to implement than before. Its cost effectiveness and speedup performance is more beneficial than that of a single processor machine. In data mining, sensor data analysis, bioinformatics, image processing and signal processing applications, there is a huge volume of data to be processed. Because loads of data can be arbitrarily partitioned into different fractions and transmitted and distributed to other parallel nodes

as to process data, divisible load theory [1] can be employed to implement parallel data processing as a tractable and efficient manner.

Divisible loads are parallel data loads partitioned among links and processors. Such loads arise in massive amounts of data in grid computing, signal processing, image processing and aerospace data processing. Given many processors interconnected by channel speed limited links in some type of network, the optimal fraction of loads to be assigned to processors and links can be defined through linear equations, mathematical programming or, in some cases, by simple algebraic recursions. Divisible load models should consider both network communication and processors computation as factors. Optimal divisible load scheduling has been developed for different type of networks such as linear daisy chains[2], buses[3], trees[4][5][6], hypercubes[7], two and three dimensional meshes[8] and arbitrary graphs. A number of scheduling polices have been developed including multi-installments[9], multi-round scheduling[10][11], different release times, buffer constraints[12], communication start-up costs, time-varying network capacity, simultaneous distribution[13][14] and simultaneous start[15]. Also solution time optimization[16], combinatorial schedule optimization[17], are detailed studied. Moreover, limited memory[18], multiple loads[19], combinatorics relating to divisible load theory[20], a large number of relatively small independent tasks[21][22] are investigated. Introductions to divisible load scheduling theory appear in [23][24][1].

Traditionally, a signature means a handwritten depiction of someones name, nickname, or even a simple X or other mark that a person writes on documents as a proof of identity and intent. Here a signature is a relatively

small data pattern of interest embedded in a very large (in this paper sequential) data file. There are many applications of this problem including DNA sequence searching, network intrusion detection, large scientific experiment, bioinformatics, speech and text recognition and sensor networks. In [25] signature searching is investigated on bus networks experimentally. In this thesis some signature searching is examined in some specific contexts.

Cyclic networks are a type of network derived from structure of tree networks and mesh networks. Traditionally, the processors at a lower layer in a multiple level tree network can receive work load from the same processor at a higher layer. The situation that a processor at the lower layer receives the load from multiple processors at higher level should be considered. One simple type of network that embodies this concept, called a cyclic network, has been investigated in this thesis. This is a new topic within the field of divisible load theory.

The emergence of distributed computing as a divisible technology and the decreasing pricing of computer power leads to the possible emergence of computer "utilities" in the near future. These utilities would charge customers for distributed access to computer resources. To some extent, current computer service leasing companies embody this approach. An important question for the utility then becomes the management of computer resources to provide low cost service. In this spirit, this thesis provides an approach to determine the minimum cost manner in which load should be divided among processors that a customer is being charged for access to.

The thesis is organized as below: chapter 2 gives the definition, applica-

tion and significance of signature, investigate the signature searching time and speedup in multi-level tree network under different cases. Chapter 3 extend other type of network: mesh network and hypercube network to discuss signature searching time, compare four types network when search signature. Chapter 4 develop a new kind of network: cyclic network and discusses the load distribution strategy and compare them. Chapter 5 propose the way to balance solution time and computing in a bus network, plot out the trade-off line with constant master processor and ultimate boundary with multiple master processor. Chapter 6 makes the conclusion.

Chapter 2

Signature Searching in a Networked Collection of Files for multi-level tree network

2.1 Introduction

A signature is a relatively small data pattern of interest embedded in a very large (in this paper sequential) data file. It is assumed signatures are temporally distinct and do not overlap each other. That is, there can be multiple signatures in a file. Because the files we study are much longer than the signatures, it is assumed that signatures have infinitesimally small length. Such signature searching occurs in network security, signal processing, medicine, image processing, and sensor technology and many other fields.

Most previous work on signature searching (known as template matching

and string matching) develop algorithm for the detailed matching process. This paper, like [26][27] addresses an upper level view of signature searching involving system performance evaluation. However we now briefly summarize some string matching work.

String searching, which is similar to our concept of signature searching, is a special case of pattern searching. String searching generically involves finding a pattern of length m in a text of length n over some alphabet. The worst case complexity of exact string matching is O(n) but the proportionality constant of the linear term can be very different depending on the string matching algorithm, ranging from m for the naive algorithm to 2 for the Knuth-Morris-Pratt algorithm [28].

Approximate string matching involves string matching that allows errors. That is, the pattern and/or text suffer some corruption. Applications include noisy channels, speech recognition, hand writing recognition, finding DNA sequences in the presence of mutations and text searching. Approximate string matching algorithms utilize some distance metric to quantify the amount of difference between two strings. For instance, the edit distance is the number of differences between two strings. The computational complexity of approximate string matching can range from linear to NP complete depending on the error mechanism [29].

For on-line algorithms, it is assumed that the text is not known in advance. For off-line algorithms the text can be pre-processed, thus indexing can be used [30]. On-line algorithms generally consist of a phase of entering the string into a data structure and a phase of looking for a match using the

data structure [31].

In this paper we assume the data is stored in flat files (i.e. very long linear sequences of data) stored at nodes of certain interconnection networks. We assume linear (in the file size) computational complexity which applies to exact string matching and some approximate string matching. Naturally more sophisticated database methodologies are possible and flat file are often converted into other structures [32][33] but for initial raw data processing flat files are natural [34].

We envision a scenario where files containing signatures are placed on a multiplicity of (parallel) processors tied together by an interconnection network [26][27]. Unlike the work in much of the divisible load theory literature [27][35][36][37], we do not take the time to distribute the load to the processors and links into account. Rather we assume the files are pre-loaded onto the processors prior to time t = 0. This is relevant in certain applications. Load is often spontaneously distributed to processors without being scheduled, monitored or timed. Our goal is to determine expected search time and speedup under a variety of search protocols that largely differ in a number of aspects. These include:

1) Whether the number of signatures in a file at a node is known or unknown a priori. The latter is more likely in general but the former could also occur. For instance when a search is done for "management" in a company database, the number of signatures matches may be generally known a priori.

2) Whether a data file stored at a node contains one or more than one

signature. For instance, a database of company employee information may contain one match for a given individual, but multiple matches for individual who are "management".

3) If no signatures are stored in a node, whether it implies that there are no signatures stored in the children nodes of the node. For instance a node may store aggregated/summarized national data and children nodes store more detailed state/provincial data. It may be possible in searching the national nodes to find children nodes that should be searched and those that need not be searched.

4) Differing degrees of sequentiality and concurrency in the search strategy. Both have been investigated in the divisible load scheduling literature over the years[35][36][37].

These are important assumptions that may be made in specific applications and we try to study this problem in a complete manner. In this context this work examines tree, mesh and hypercube interconnection networks [8][38][39][40]. We look at trees, meshes and hypercubes mainly because these are fundamental interconnection networks. Trees are often used as spanning trees to distribute load in other types of interconnection networks. Meshes are often used as interconnection networks in parallel processors. Meshes are particularly well suited to networks on chips [41][42][43]. Hypercubes are widely used in parallel processors. One could certainly examine other interconnection networks but space limitations prevent this in this paper.

In earlier work the expected search time to find either one [26] or multiple signatures [27] with a uniform distribution of signatures in a flat file was

found analytically. This earlier work involved trees and daisy chains and incorporated, unlike this work, load distribution time to processors over a network.

The rest of the paper is organized as follows. Section 2.2 give some assumption and definition of tree network. Section 2.3 and 2.4 discusses searching time and speedup in tree networks for different cases. The searching time and speedup in mesh networks (store and forward) for different cases are discussed in chapter 3. A comparison of the speedups for different type of networks is presented in section 3.2. Finally, this paper concludes with some possible extensions in section 3.3.

2.2 Tree Networks

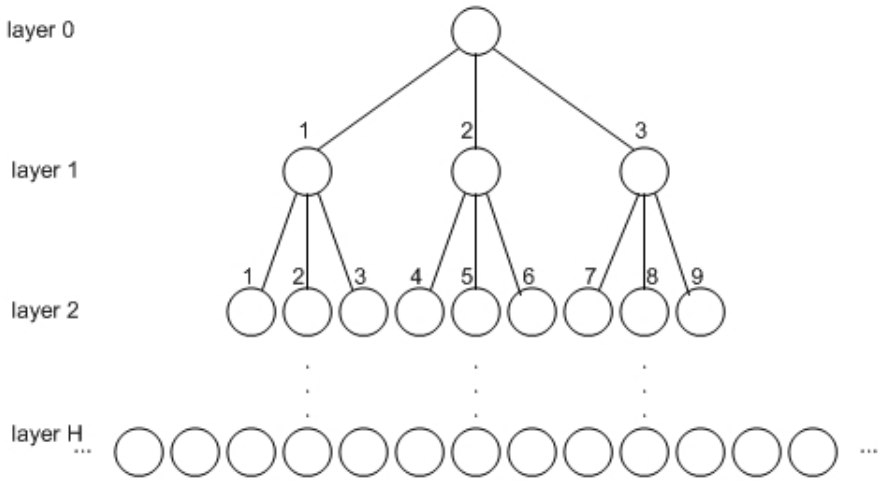

Figure 2.1: Multi-level tree network

A general tree is shown in Figure 2.1. Assume that in a tree a node is a

structure which contains exactly one file. Each node has one, two or more child nodes, which are below it in the tree space. A node that has a child is called the child's parent node.

A node has at most one parent. Nodes that do not have any children are called leaf nodes. They are also referred to as terminal nodes. A node's height is the length of the longest downward path to a leaf from that node. The root's height is the height of the tree. The depth of a node is the length of the path to its root. The nodes in the same depth of the tree are said to be at the same level.

The height of the multiple-level tree in this paper is H. Throughout this paper there is a single file, possibly containing signatures, in each node. In all the cases for tree networks which we examine the number of children nodes n_i per node is a random variable from 1 to N, possibly different for each level. But at the same level the number of children in each subtree is the same. In this section if a parent node has a signature(s), its children nodes may also have signatures. If the parent node does not have any signature of interest, its children nodes also have no signatures and do not need to be searched. Some dependency between signature occurrences is thus being assumed. This may model locality of reference - if a node has a signature(s), there may be related information (signatures) on its children.

The lengths of all the files in the tree are same, L (bits). The inverse searching speed for one processor is w (s/bit). Suppose that the expected time we need to search a file with signature is $\overline{X}$, because the expected position of signature is in the middle of such file (i.e. uniform distribution

in linear file assumption). The time to search the whole file is $2\overline{X}$. while $2\overline{X} = L \cdot w$. The purpose of this paper is to calculate the time we need to find all of the files with signatures. We will discuss several cases in sub-sections 2.1 and 2.2 using the assumptions listed in the introduction.

2.3 Number of signatures is unknown

2.3.1 Each file has at most one signature

Let M_i be the number of signatures in the i_{th} layer and n_i be the number of children nodes in each subtree in the i_{th} layer. If there is at most one signature in one file, $M_i \leq M_{i-1}n_i \leq M_{i-2}n_{i-1}n_i \leq \ldots \leq n_1 n_2 \ldots n_i$. The expected time to search one file is $\overline{X}$, the searching process for a file will stop as soon as we find one signature (as also true in section 3.1.1 and 3.2.1).

Only one node in the tree commands all the nodes

First, assume that only the top most root node in the multi-level tree commands all the nodes to search, for every layer, every file in the layer and each node replies to the top node whether there is a signature in the node. That is, one layer is searched sequentially at a time. The expected time to search the entire tree is

$$T_{u1} = n_1\overline{X} + M_1 n_2\overline{X} + \ldots + M_{H-1}n_H\overline{X} = \sum_{i=1}^{H} M_{i-1}n_i \cdot \overline{X} \qquad (2.1)$$

Here, $M_0 = 1$.

Now we consider a homogeneous case that for all the subtrees in a certain layer the probability of a signature is the same: p_i for the i_{th} layer. Equation (2.1) will be $T_{u1} = n_1\overline{X} + p_1 n_1 n_2 \overline{X} + \ldots + (\prod_{j=1}^{H-1} p_j n_j) n_H \overline{X} = \sum_{i=1}^{H} (\prod_{j=1}^{i-1} p_j n_j) n_i \overline{X}$. A homogeneous case occurs if $p_1 = p_2 = \ldots = p_{H-1} = p$, equation (2.1) will be $T_{u1} = \sum_{i=1}^{H} (\prod_{j=1}^{i} n_j) p^{i-1} \overline{X}$. The fully homogeneous case occurs if the number of files in all of the subtrees is the same: n, and the probability of signature is also the same: p, then the expected search time is $T_{u1} = \sum_{i=1}^{H} n^i p^{i-1} \overline{X} = \frac{p^H n^{H+1} - n}{pn-1} \overline{X}$.

Every root node of each local tree commands the nodes below it

In this case each level is searched sequentially, within each subtree the search is sequential but all subtrees at the same level are searched in parallel. The time to search the i_{th} layer is $n_i\overline{X}$. The total expected time to search the multi-level tree is

$$T_{u2} = n_1\overline{X} + n_2\overline{X} + \ldots + n_H\overline{X} = \sum_{i=1}^{H} n_i\overline{X} \tag{2.2}$$

The speedup compared with last case is

$$\tau_{u1} = \frac{\sum_{i=1}^{H} M_{i-1} n_i \cdot}{\sum_{i=1}^{H} n_i} \tag{2.3}$$

For the fully homogeneous case, $\tau_{u1} = \frac{p^H n^{H+1} - n}{pn-1} \frac{1}{nH} = \frac{(pn)^H - 1}{(pn-1)H}$.

All the files in every level are searched in parallel, levels are

searched sequentially

In this case each level is searched sequentially and within a level the nodes are searched in parallel. For every level the search time is only $\overline{X}$. The total search time is $T_{u3} = H\overline{X}$. The speedup compared with the first case in this section is

$$\tau_{u2} = \frac{\sum_{i=1}^{H} M_{i-1} n_i}{H} \tag{2.4}$$

For the fully homogeneous case, τ_{u2} is $\frac{((pn)^H - 1)n}{(pn-1)H}$.

All the files in the tree are searched in parallel

This case is the fastest search method. All the files in the entire tree are searched in parallel. The total search time is $T_{u4} = \overline{X}$ and the speedup is

$$\tau_{u3} = \sum_{i=1}^{H} M_{i-1} n_i \tag{2.5}$$

For the fully homogeneous case, τ_{u3} is $\frac{((pn)^H - 1)n}{(pn-1)}$.

It is apparent that as the height increases, all the three speedups also increase, and $\tau_{u1} < \tau_{u2} < \tau_{u3}$, τ_{u3} is much larger than τ_{u1} and τ_{u2} when the height is high. From equations (2.3), (2.4), (2.5) we know $\tau_{u3} = H \cdot \tau_{u2}$, $\tau_{u2} = \frac{\sum_{i=1}^{H} n_i}{H} \tau_{u1}$, the expected mean number of $\frac{\sum_{i=1}^{H} n_i}{H}$ is $\frac{\frac{N}{2} H}{H} = \frac{N}{2}$. This confirms that the highest performing strategies, if it can be implemented, is to search all files in parallel.

2.3.2 Multiple signatures can exist in one file

One should search the whole file to find all of the signatures in each file. If there are multiple signatures in one file, the expected search time is $2\overline{X}$ (section 3.1.2 and 3.2.2), M_i can be larger than $M_{i-1} \cdot n_i$, and we assume that those M_i signatures exist in N_i files (each of N_i files with at least one signature).

Only one node in the tree commands all the nodes The expected search time in layer i is $N_{i-1}n_i \cdot 2\overline{X}$, for N_i files with signatures, $N_i \leq M_i$. The total expected search time is:

$$\acute{T}_{u1} = \sum_{i=1}^{H} N_{i-1}n_i \cdot 2\overline{X} \tag{2.6}$$

Every root node of each local tree commands the nodes below it Again the levels are searched sequentially, within each subtree the search is sequential but all subtrees at the same level are searched in parallel. The expected search time in layer i is $n_i \cdot 2\overline{X}$. The total expected search time is:

$$\acute{T}_{u2} = \sum_{i=1}^{H} n_i \cdot 2\overline{X} \tag{2.7}$$

The speedup is:

$$\acute{\tau}_{u1} = \frac{\sum_{i=1}^{H} N_{i-1}n_i \cdot 2\overline{X}}{\sum_{i=1}^{H} n_i \cdot 2\overline{X}} = \frac{\sum_{i=1}^{H} N_{i-1}n_i}{\sum_{i=1}^{H} n_i} \tag{2.8}$$

All the files in every level are searched in parallel, levels are searched sequentially The same case as before, $\acute{T}_{u3} = H \cdot 2\overline{X}$. The speedup

is $\acute{\tau}_{u2} = \frac{\sum_{i=1}^{H} N_{i-1} n_i}{H}$

All the files in the tree are searched in parallel The same case as before, $\acute{T}_{u4} = 2\overline{X}$. The speedup is $\acute{\tau}_{u3} = \sum_{i=1}^{H} N_{i-1} n_i$

The speedup is plotted in figure 2.2. As a baseline result, it is assumed that the distribution of signatures in every file is a Poisson distribution: $P(m = k) = \frac{\lambda^k}{k!} e^{-\lambda}$($m$ is the number of signatures in every file). We can see from figure 2.2 that when λ goes up from 1 to 20, the speedup $\acute{\tau}_{u1}$ increases quickly first and then saturates. That is because initially the number of subtrees needed to be searched increased and later one winds up searching all, but not more than all, files in a level, then the speedup saturates.

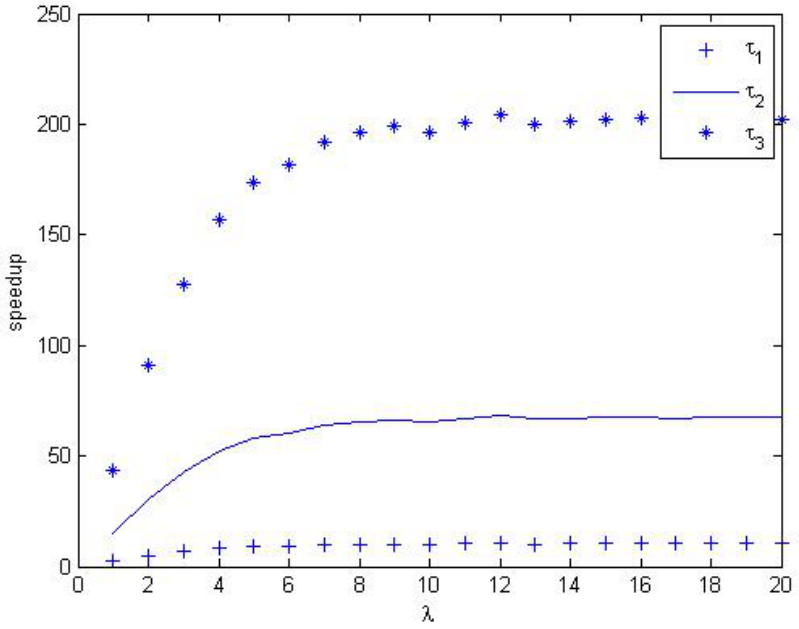

Figure 2.2: Speedup when the number of signatures is unknown and each file can have multiple signatures

2.4 Number of signatures is known

When the number of signatures is known, the searching process will stop as soon as the last signature is found, thus the time should be shorter than that of the case where the number of signatures is unknown. Two cases are discussed.

2.4.1 Each file has at most one signature

In this case assume that there are m signatures in n files in a single level of the tree. Each file can have at most one signature. Each node holds one file. We focus on the last signature because the searching process will not stop until the last signature is found, thus the search time depends on the position of the last signature. Because there are m signatures and every file can have at most one signature, the last signature should be in the $m_{th}, (m+1)_{th}, \ldots, n_{th}$ file. If the last signature is in the $(m-1)_{th}$ or even a file to the left of m_{th} file, there will be multiple signatures in some files, which is a contradiction. The probability that the last signature is in the m_{th} file is (there are $m-1$ signatures in $m-1$ files)

$$P_m = \frac{\binom{m-1}{m-1}}{\binom{n}{m}} \tag{2.9}$$

And the expected search time is

$$t_m = P_m(m\overline{X}) \tag{2.10}$$

The probability that the last signature is in the i_{th} file is (there are $m-1$ signatures in $i-1$ files) for $i \geq m-1$ is

$$P_i = \frac{\binom{i-1}{m-1}}{\binom{n}{m}} \tag{2.11}$$

And the expected search time is

$$t_i = P_i \cdot (m\overline{X} + (i-m)2\overline{X}) = \frac{\binom{i-1}{m-1}}{\binom{n}{m}}(2i-m)\overline{X} \tag{2.12}$$

Thus, the expected search time T_s for this single level is

$$T_{n,m} = \sum_{i=m}^{n} t_i = \sum_{i=m}^{n} \frac{\binom{i-1}{m-1}}{\binom{n}{m}}(2i-m)\overline{X} \tag{2.13}$$

We plot the expected search time as the number of signatures increases. In figure 2.3 the number of files $n = 100$, and we can conclude that when $m = 13$, the expected search time is the largest.

We derive the expression for the maximum value of the function of figure 2.3 as below.

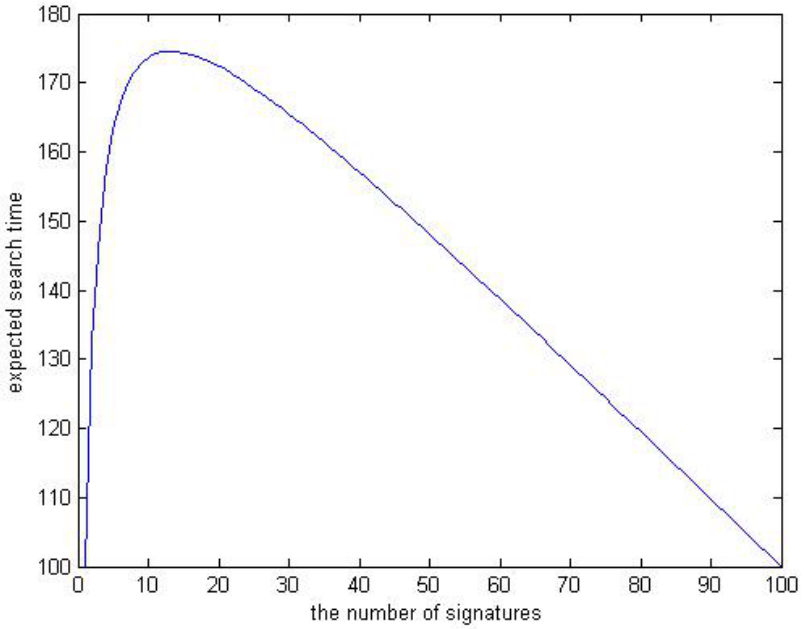

Figure 2.3: Single level tree when the number of signatures is known

$$
\begin{aligned}
T_{n,m} &= \sum_{i=m}^{n} t_i \\
&= \sum_{i=m}^{n} \frac{\binom{i-1}{m-1}}{\binom{n}{m}}(2i-m)\overline{X} \\
&= 2\sum_{i=m}^{n} i\frac{\binom{i-1}{m-1}}{\binom{n}{m}}\overline{X} - m\sum_{i=m}^{n}\frac{\binom{i-1}{m-1}}{\binom{n}{m}}\overline{X} \\
&= 2\sum_{i=m}^{n} m\frac{\binom{i}{m}}{\binom{n}{m}}\overline{X} - m\overline{X} && (2.14) \\
&= 2m\frac{n+1}{m+1}\overline{X} - m\overline{X} && (2.15) \\
&= \frac{2nm - m^2 + m}{m+1}\overline{X} && (2.16) \\
& && (2.17)
\end{aligned}
$$

$$\frac{\mathrm{d}}{\mathrm{d}m}T_{m,n} = \frac{2n - m^2 - 2m + 1}{(m+1)^2} \cdot \overline{X} \tag{2.18}$$

Then get $m = \sqrt{2n+2} - 1$. We demonstrate $\sqrt{2n+2} - 1$ is between 1 and n.

$$\sqrt{2n+2} - 1 < n \tag{2.19}$$

$$2n + 2 < (n+1)^2 \tag{2.20}$$

$$1 < n^2 \tag{2.21}$$

$$\tag{2.22}$$

Then we obtain $m = \sqrt{2n+2} - 1$. If $n = 100$, then equation 2.14 will be $\frac{\mathrm{d}}{\mathrm{d}m}T_{m,n} = \frac{200 - m^2 - 2m + 1}{(m+1)^2}$, let $\frac{\mathrm{d}}{\mathrm{d}m}T_{m,n} = 0$, then we get the maximum expected search time occur when $m = 13.21$ which is close to $m = 13$. For this graph if a file has a signature the expected search time is $\overline{X}$, and if there is no signature, it is $2\overline{X}$. Initially the mean time to find all signatures increases as more signatures are located closer to the last file. Eventually though as the number of signatures increases most files have a signature forcing the average search time per file closer to $\overline{X}$ rather than $2\overline{X}$ and the curve decreases.

Up to this point in this section the expected time for one single level has been solved. Next the expected time to search an H level tree is considered. There are four cases just as in the case where the number of signatures is unknown.

Only one node in the tree commands all the nodes

Assume that there are $m_{i,j}$ signatures in the j_{th} subtree in layer i, and $M_i = \sum_{j=1}^{M_{i-1}} m_{i,j}$, M_i stands for the total number of signatures in the i_{th} layer. Then for the j_{th} subtree in layer i, the expected search time $T_{n_i,m_{i,j}} = \sum_{i=m_{i,j}}^{n_i} \frac{\binom{i-1}{m_{i,j}-1}}{\binom{n_i}{m_{i,j}}}(2i - m_{i,j})\overline{X}$. The total expected search time is thus:

$$T_1 = \sum_{i=1}^{H} \sum_{j=1}^{M_{i-1}} T_{n_i,m_{i,j}} \tag{2.23}$$

Assume $m_{1,1} = m_1 = M_1$ and $M_0 = 1$.

Every root node at local tree to command the nodes below it

Again, the levels are searched sequentially, within each subtree the search is sequential. However all subtrees at the same level are searched in parallel. For a given layer, the search time in this layer is the longest expected search time in the layer's subtrees. Assume that in the i_{th} layer the largest expected search time is $max[T_{n_i,m_{i,j}}]$ (j is from 1 to M_{i-1} while M_i has been defined before), The total time to search the entire tree is:

$$T_2 = \sum_{i=1}^{H} max[T_{n_i,m_{i,j}}] \tag{2.24}$$

The speedup is:

$$\tau_1 = \frac{\sum_{i=1}^{H} \sum_{j=1}^{M_{i-1}} T_{n_i,m_{i,j}}}{\sum_{i=1}^{H} max[T_{n_i,m_{i,j}}]} \tag{2.25}$$

For the fully homogeneous case where $n_1 = n_2 = \ldots = n_H = n$ and $m_1 = m_{2,1} = \ldots = m_{H,1} = \cdots = m$, the number of signatures in every

subtree is the same in every layer in the first case. Then

$$\tau_1 = \frac{\sum_{i=1}^{H} m^{i-1} T_{n,m}}{H \cdot T_{n,m}} = \frac{m^H - 1}{H(m-1)} \tag{2.26}$$

All the files in every level are searched in parallel, levels are searched sequentially

Discussed as before, $T_3 = H\overline{X}$ and the speed up compared with case 2.2.2.1 is

$$\tau_2 = \frac{\sum_{i=1}^{H} \sum_{j=1}^{M_{i-1}} T_{n_i,m_{i,j}}}{H\overline{X}} \tag{2.27}$$

For the fully homogeneous case that $n_1 = n_2 = \ldots = n_H = n$ and $m_1 = m_{2,1} = \ldots = m_{H,1} = \ldots = m$, $\tau_2 = \frac{m^H-1}{H(m-1)} T_{n,m} = \frac{m(m^H-1)(2n-m+1)}{H(m^2-1)}$.

All the files in the tree are searched in parallel

Discussed as before $T_4 = \overline{X}$, and the speedup $\tau_3 = \frac{\sum_{i=1}^{H} \sum_{j=1}^{M_{i-1}} T_{n_i,m_{i,j}}}{\overline{X}}$. For the fully homogeneous case that $n_1 = n_2 = \ldots = n_H = n$ and $m_1 = m_{2,1} = \ldots = m_{H,1} = \ldots = m$, then $\tau_3 = \frac{m^H-1}{H(m-1)} T_{n,m} = \frac{m(m^H-1)(2n-m+1)}{m^2-1}$.

This is apparent that $\tau_3 > \tau_2 > \tau_1$, because for the fully homogeneous case, $\tau_3 = H\tau_2$, $\tau_2 = \frac{m(2n-m+1)}{m+1}\tau_1$, absolutely $\frac{m(2n-m+1)}{m+1} > 1$.

2.4.2 Multiple signatures can exist in one file

First we consider a single level tree which has n files and m signatures. We combine all the files logically in one level into a single file. We also assume that the capacity of one file is L bits, so that the length of the combined file is $n \cdot L$. The position of the signatures is $X_1, X_2, \ldots, X_m$,

$(nL \geq X_1, X_2, \ldots, X_m \geq 0)$, $X \sim U(0, nL)$.

$$P(X_{max} = x) = \binom{m}{1} \cdot \frac{1}{nL} \cdot (\frac{x}{nL})^{m-1} \tag{2.28}$$

The expected length of the last signature position:

$$\overline{X_{max}} = \int_0^{nL} \frac{m}{nL} \cdot (\frac{x}{nL})^{m-1} \cdot x dx = \frac{m}{m+1} \cdot nL \tag{2.29}$$

The search inverse speed is w s/bit. Then the expected time we need to search one single level is $\acute{T}_{n,m} = \overline{X_{max}} \cdot w = \frac{m}{m+1} \cdot n \cdot L \cdot w$, here $L \cdot w = 2\overline{X}$, thus

$$\acute{T}_{n,m} = \frac{2m}{m+1} n\overline{X} \tag{2.30}$$

This is plotted in figure 2.4. Here the number of files is 10 and the maximum number of signatures is 50. From the figure we can see that the expected search time will increase and saturate as the number of signatures increase. That can be seen from equation (2.22), as m increases, $\acute{T}_{n,m}$ will be close to $2n\overline{X}$.

Until now in this section we have solved the expected time in one single level, now we consider the expected time we need to search the H level tree. Assume that the m_i signatures are in $\acute{n}_i$ files. Apparently $\acute{n}_i \leq m_i$. The four cases are discussed as below.

Only one node in the tree commands all the nodes

For the i_{th} layer, there are N_{i-1} files that should be searched, there

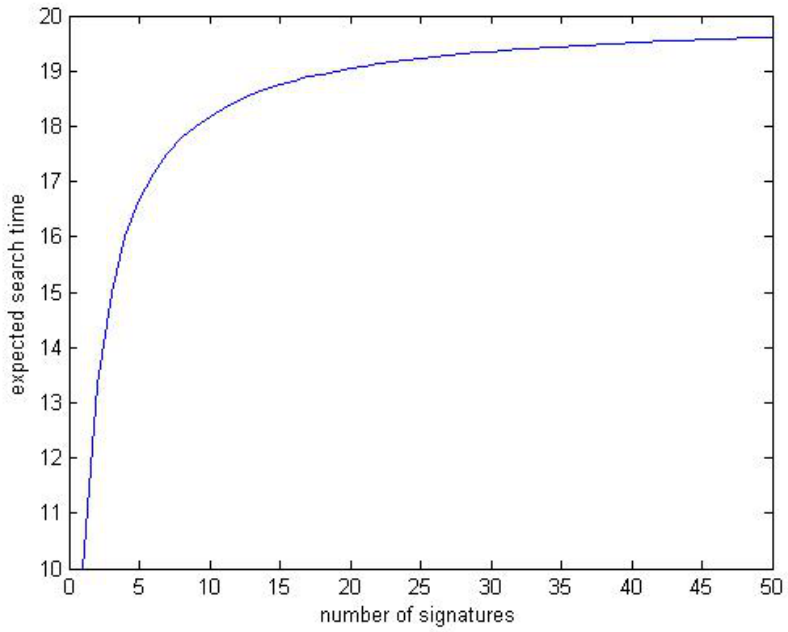

Figure 2.4: The expected search time when the number of signatures is known and multiple signatures can exist in one file for a single level tree

are $m_{i,1}$, $m_{i,2}$, ..., $m_{i,N_{i-1}}$ signatures in $\acute{n}_{i,1}$, $\acute{n}_{i,2}$, ..., $\acute{n}_{i,N_{i-1}}$ files, $N_i = \sum_{j=1}^{N_{i-1}} \acute{n}_{i,j}$. The total time is

$$\acute{T}_1 = \sum_{i=1}^{H} \sum_{j=1}^{N_{i-1}} \acute{T}_{n_i,m_{i,j}} \tag{2.31}$$

Note that $N_0 = 1$ and $\acute{n}_{i,j} \leq m_{i,j}$.

Every root node at local tree to command the nodes below it

In this case the search time in the i_{th} layer is the time to search one of the subtrees which has the largest number of signatures (like the case discussed before). The total search time is:

$$\acute{T}_2 = \sum_{i=1}^{H} max[\acute{T}_{n_i,m_{i,j}}] \tag{2.32}$$

Note that j is from 1 to N_{i-1}. The speed up is:

$$\acute{\tau}_1 = \frac{\sum_{i=1}^{H} \sum_{j=1}^{N_{i-1}} \acute{T}_{n_i,m_{i,j}}}{\sum_{i=1}^{H} max[\acute{T}_{n_i,m_{i,j}}]} \tag{2.33}$$

All the files in every level are searched in parallel, levels are searched sequentially

Discussed as before, $\acute{T}_3 = 2H\overline{X}$. The speedup compared with case that only one node command all the nodes is

$$\acute{\tau}_2 = \frac{\sum_{i=1}^{H} \sum_{j=1}^{N_{i-1}} \acute{T}_{n_i,m_{i,j}}}{2H\overline{X}} \tag{2.34}$$

All the files in the tree are searched in parallel

The total search time in this case is $2\overline{X}$. The speedup here is:

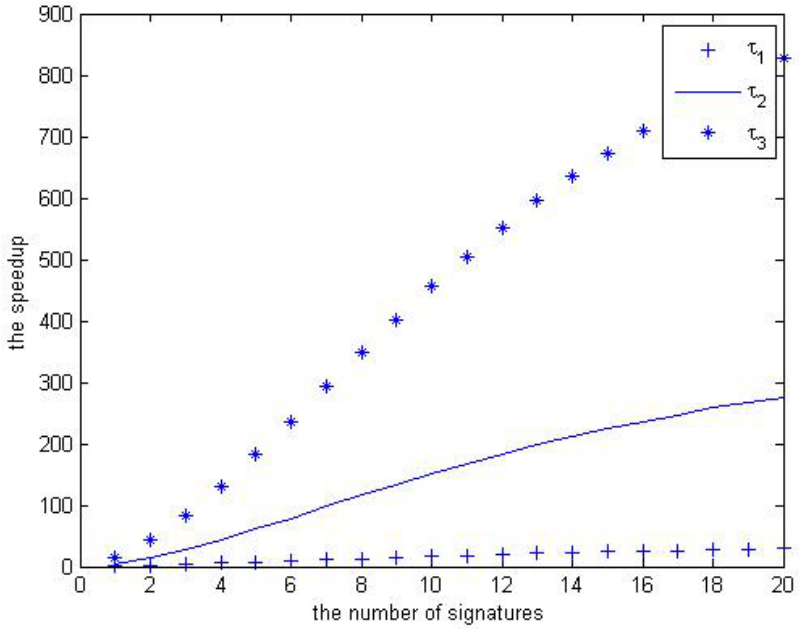

Figure 2.5: Speedup when there can be multi-signatures in one file and the number of signatures in a level is known

$$\acute{\tau_3} = \frac{\sum_{i=1}^{H} \sum_{j=1}^{N_{i-1}} \acute{T}_{n_i,m_{i,j}}}{2\overline{X}} \tag{2.35}$$

The speedup figure for these cases has been plotted in figure 2.5. From the figure we can see that $\acute{\tau_3} > \acute{\tau_2} > \acute{\tau_1}$. The reason is similar to the case when each file has at most one signature.

Chapter 3

Signature Searching in a Networked Collection of Files for mesh and hypercube network

3.1 Mesh Networks: Store and Forward

A regular two dimensional mesh network of processors [36] is shown in Figure 3.1. It is a commonly used interconnection network. In this network structure each processor is located in the corners of four rectangles and has four neighbors. The central processor is called the originator, it can communicate information or transport data to its four neighboring nodes. As in trees, the central node is assumed to be layer 0, and its four neighboring nodes in layer

1, the nodes which are neighbors to the nodes in layer 1 but not in layer 0 are in layer 2, . . . , the nodes which are neighbors to the nodes in layer i but not in layer i-1 are in layer i+1, as shown in the mesh structure. A node can only send messages or data to its neighboring nodes. If the node in the north of layer 2 wants to send a message to the node in layer 0, first it should send message to the node in the north top of layer 1, then the message will be transferred to layer 2.

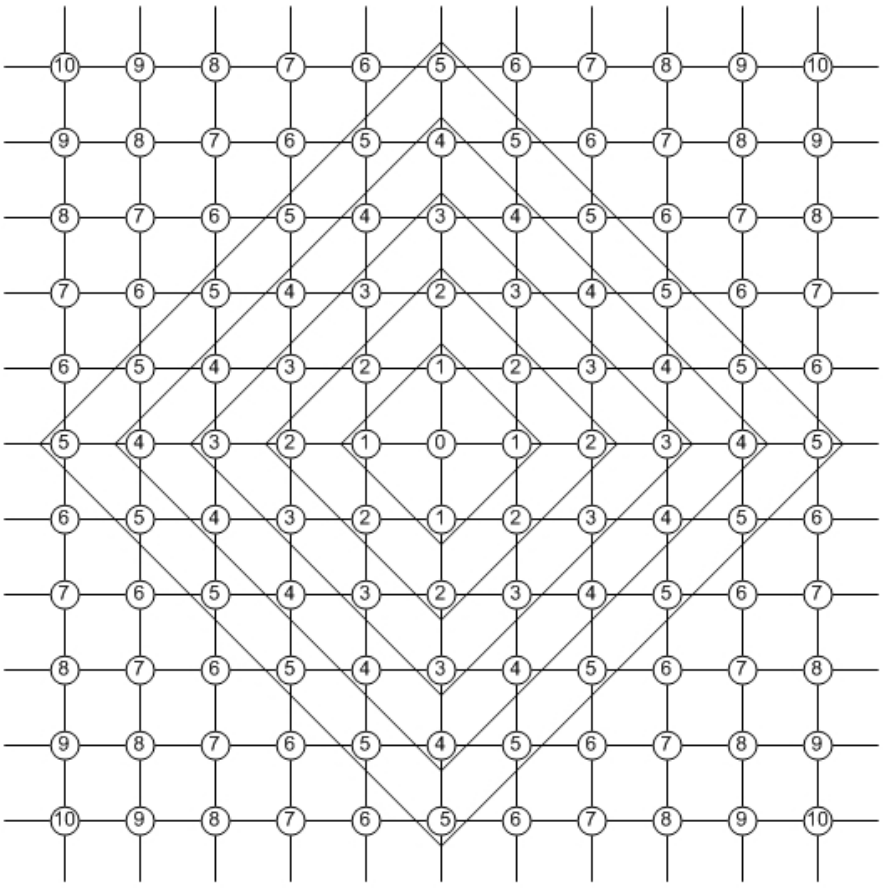

Figure 3.1: Mesh network

Assume that there are N layers (as shown in figure 3.1) and there is at most one signature in every node in a mesh network. There are $4i$ nodes in the i_{th} layer. Different from the tree network, the case that every node can have a signature will be discussed in section 3.2. That is, if a node has no signature, its children nodes can still have signatures. Assume the average time to search every node in the mesh network is the same: $\overline{X}$, and again there is at most one signature in one file.

3.2 Every file can have its own signature regardless of the upper layer node

3.2.1 The number of signatures is unknown

In this case every node should be searched. There are three cases.

Only one node in the network commands all the nodes

The central node which is in layer 0 commands each node to search for signatures. As soon as the central node sends a message to command one node to start to search its file, the central node will wait until it receives a message indicating whether there is any signature in the file. All of the nodes in the mesh network are searched one by one. It is apparent that the searching time is the longest, the searching time in the i_{th} layer is $4i \cdot \overline{X}$. The total expected search time is:

$$T_1 = 4\overline{X} + 8\overline{X} + \ldots + 4N\overline{X} = 2N(N+1)\overline{X} \tag{3.1}$$

Every root node of each local network commands the node below it

The searching time in layer one is $4\overline{X}$. For layer two, there are 8 nodes to be searched. We assign two nodes in layer two to be searched by each node in layer one, the search time should be $2\overline{X}$. For layer three, there are two cases, the first case is for the four nodes in the upmost, or downmost, or leftmost, or rightmost direction. The expected search time for these four nodes is $3\overline{X}$, because there are three neighboring nodes in their lower layer; and for the second case (the remaining nodes), there are only two neighboring nodes in their lower layer. Thus the expected search time is $2\overline{X}$. However, we can eliminate $\overline{X}$ searching time for both cases, because every node in layer three except the four top nodes in four directions have two upper nodes (different from the tree network). If one node for the node in layer three is commanded by one node in layer two, the other node in layer two need not command it to search for a signature. For layer three, the searching time is $2\overline{X}$, the same for layer four, five, ... The total expected search time is $T_2 = 4\overline{X} + 2\overline{X} + 2\overline{X} + \ldots + 2\overline{X} = 4\overline{X} + 2(N-1)\overline{X} = (2N+2)\overline{X}$.

All the files in every layer are searched in parallel, layers are searched sequentially

Like the third case in the tree network, the search time in every layer is $\overline{X}$. The total time is $T_3 = N\overline{X}$.

All the files in the mesh network are searched in parallel

The expected search time in this case is only $T_4 = \overline{X}$.

Then we calculate the speedup in these four cases: $\tau_1 = \frac{T_1}{T_2} = N, \tau_2 =$

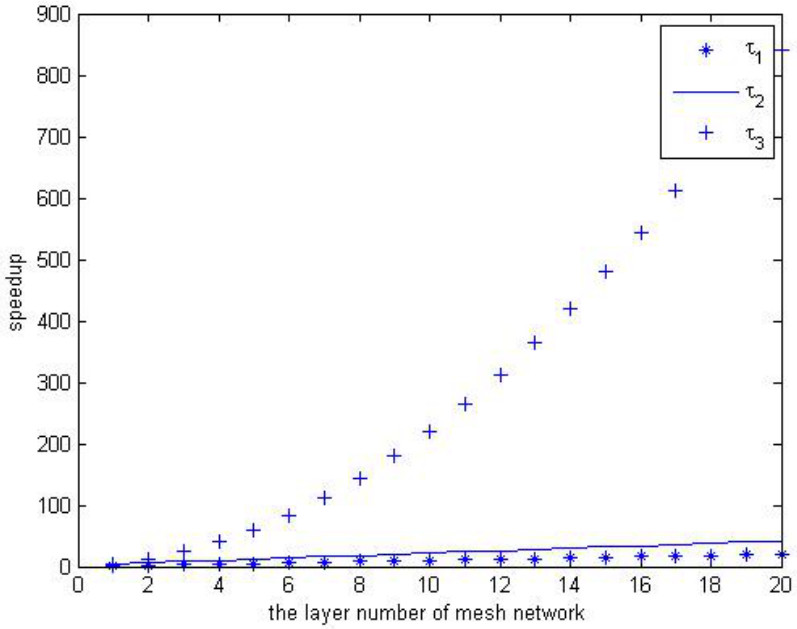

Figure 3.2: Speedup when the number of signatures is unknown in a mesh network and each node can have at most one signature

$\frac{T_1}{T_3} = 2(N+1), \tau_3 = \frac{T_1}{T_4} = 2N(N+1)$, and the speedup figure has been plotted in figure 3.2. From the figure we know that $\tau_3 > \tau_2 > \tau_1$, that is because $\frac{\tau_2}{\tau_1} = \frac{2(N+1)}{N} > 2$ and $\frac{\tau_3}{\tau_2} = N$.

3.2.2 The number of signatures is known

Only one node in the network commands all the nodes

This case is mostly similar to the case in the tree network: assume there are m_i signatures in the i_{th} layer, we focus on the last signature. There are $4i$ files in the i_{th} layer, the probability that the last signature is in the j_{th} file in the i_{th} layer is:

$$P_j = \frac{\binom{j-1}{m_i-1}}{\binom{4i}{m_i}} \tag{3.2}$$

The expected searching time in the i_{th} layer is:

$$\acute{T_i} = \sum_{j=m_i}^{4i} \frac{\binom{j-1}{m_i-1}}{\binom{4i}{m_i}} (2j - m_i)\overline{X} \tag{3.3}$$

The total time to search the mesh network is $\acute{T_1} = \sum_{i=1}^{N} \acute{T_i}$. The time should be less than T_1.

Every root node of each local network commands the nodes below it

For layer one, the expected search time is the same as $T_{2.1} = \sum_{j=m_1}^{4} \frac{\binom{j-1}{m_1-1}}{\binom{4}{m_1}} (2j - m_1)\overline{X}$. For layer two, three and the next layers, there are two types of nodes. The first type of nodes is those nodes which have three children nodes in their lower layer, from figure 3.1 the first type of nodes are in the top most position of east, west, south, north four directions. The other type of nodes are the other nodes, which only have two children nodes in their lower layers. However, the searching time for the two types nodes are not $2\overline{X}$ and $3\overline{X}$, because every node except the four top nodes have two parent nodes. Assume that in the first $\overline{X}$ searching time all the nodes except the four top nodes are searched, and for the parent node only one node commands one node. If all the signatures are found, the searching time in this layer is $\overline{X}$. If there are some signatures in the top four nodes, we need to add $\overline{X}$ more time to find the signature, then the searching time should be $2\overline{X}$. The probability that

there are no signatures in the top four nodes in the i_{th} layer is:

$$P_{2.i} = \frac{\binom{4i-4}{m_i}}{\binom{4i}{m_i}} \tag{3.4}$$

The total time of the mesh network in this case is:

$$\acute{T_2} = \acute{T_1} + \sum_{i=2}^{N}(P_{2,i}\overline{X} + (1 - P_{2,i})2\overline{X}) \tag{3.5}$$

All the files in every layer are searched in parallel, layers are searched sequentially

The same as the last case we have discussed, the total time is $\acute{T_3} = N\overline{X}$.

All the files in the mesh network are searched in parallel

The total time is $\acute{T_4} = \overline{X}$.

Now we calculate the speedup for these four cases. $\acute{\tau_1} = \frac{\acute{T_1}}{\acute{T_2}} = \frac{\sum_{i=1}^{N}\acute{T_i}}{\acute{T_1}+\sum_{i=2}^{N}P_{2,i}\overline{X}+(1-P_{2,i})2\overline{X}}$, while $\acute{T_i}$ and $P_{2,i}$ have been defined before (see eq.3.4 and eq.3.5). $\acute{\tau_2} = \frac{\acute{T_1}}{\acute{T_3}} = \frac{\sum_{i=1}^{N}\acute{T_i}}{N\overline{X}}$, $\acute{\tau_3} = \frac{\acute{T_1}}{\acute{T_4}} = \frac{\sum_{i=1}^{N}\acute{T_i}}{\overline{X}}$. The speedup figure has been plotted in figure 3.3. From the figure we can know that $\acute{\tau_3} > \acute{\tau_2} > \acute{\tau_1}$. That is because $\acute{\tau_3} = N \cdot \acute{\tau_2}$, $\frac{\acute{\tau_2}}{\acute{\tau_1}} = \frac{\acute{T_1}+\sum_{i=2}^{N}P_{2,i}\overline{X}+(1-P_{2,i})2\overline{X}}{N\overline{X}} > 1$.

3.3 Only the nodes whose parent node has a signature can have a signature.

In this case those nodes whose parent node do not have a signature, will also have no signatures. Only those nodes whose parent node has a signature

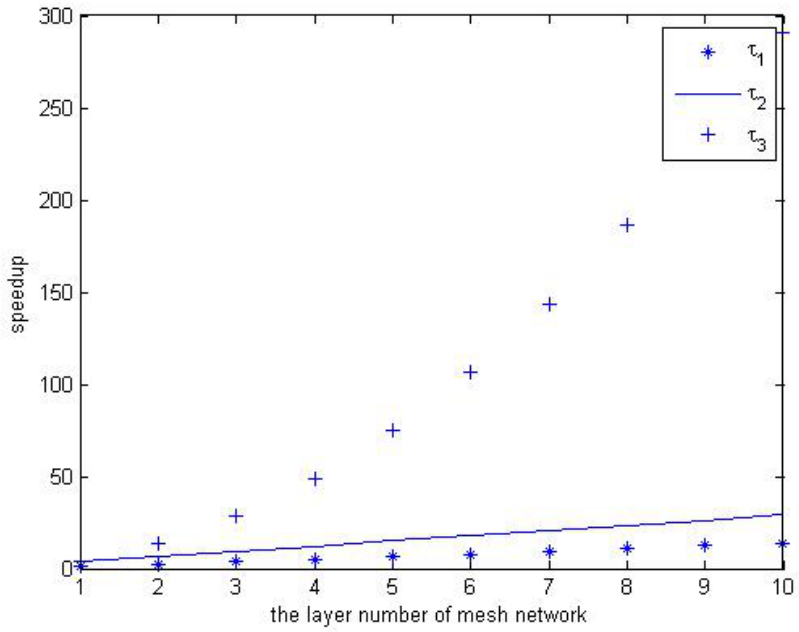

Figure 3.3: Speedup when the number of signatures is known in a mesh network and there is at most one signature in each node

need to be searched. This case is so complex that one can not find a general equation to solve it. We simulated 100,000 mesh networks with different probabilities of finding signatures in the files, and calculated the search times. In this section we only discuss the case that the number of signatures is unknown.

3.3.1 Only one node in the network commands all the nodes

The mesh network structure is divided into four parts: up-left, up-right, down-left and down-right. The nodes in these four parts will search their children nodes in four different directions, which is also up-left, up-right, down-left and down-right. It is probable that those nodes, whose parent node has a signature, will have signatures. The nodes that are not in the top position of every layer have two parent nodes, the case that either of them or both of them have a signature will lead the child node to possibly have a signature. We can get the mesh structure by creating the signature distribution layer by layer. The next step is to calculate the search time in every layer, which depends on the signature distribution in the upper layer. The four top nodes have three children nodes, and the other nodes have two children nodes. Thus the total search time for a certain layer is the number of top nodes, which have a signature to be searched, times $3\overline{X}$, plus the number of the other nodes, which have a signature, times $2\overline{X}$. For this algorithm figure 3.4 shows the search time for every layer as the probability

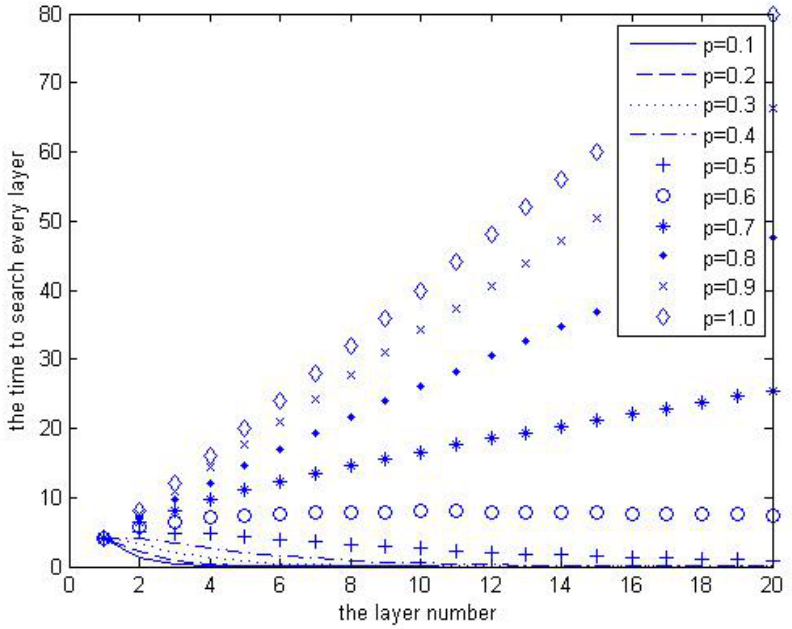

Figure 3.4: Searching time for the mesh network in case 3.3.1 as signature probability is varied

of signatures increases. Smaller values of p result in the signature searching processing stopping earlier as irregular signature free boundaries form around the mesh network that prevent further signature searching.

Figure 3.4 shows that when the probability of finding a signature in every node is higher, the expected time to search every layer will be longer. When the probability of a signature is exactly 1, that means every node will have a signature, the time to search the i_{th} layer is $4i$. When the probability of a signature is equal or less than 0.5, the number of nodes which have signatures will decrease as the layers increase until no node will have signatures in a certain layer. When the probability of signature is equal to 0.6, there are

almost always nodes which have signatures and the number of nodes which have signatures in every level remains almost constant. When the probability of signature is equal or more than 0.7, the number of nodes which have signatures increases.

3.3.2 Every root node of each local network commands the nodes below it

For the four top nodes in the east, west, north, and south directions, if both of the two neighboring nodes in their layer have no signatures (i.e they will not command their lower layer node to search their files), the searching time for the top nodes is $3\overline{X}$. If one of the neighboring nodes or both of them have signatures, we can assign the node whose parent nodes are the top node and its neighboring node to be commanded by the node neighboring the top node. Thus the searching time will be cut to $2\overline{X}$. For the other node, the maximum searching time is $2\overline{X}$, when these nodes have signatures. All the local mesh networks (the node and its children nodes) search their own nodes in parallel. The searching time for a certain layer depends on the signature distribution of the top node of the upper layer, as mentioned before, it will be $3\overline{X}$, $2\overline{X}$, or 0 (there are no signatures in the whole parent nodes). We simulate this case in figure 3.5.

From figure 3.5, when the probability of a signature is equal or less than 0.6, the searching time is shorter and shorter until we need not search the signature in the lower layer as the layer is larger. That's because smaller

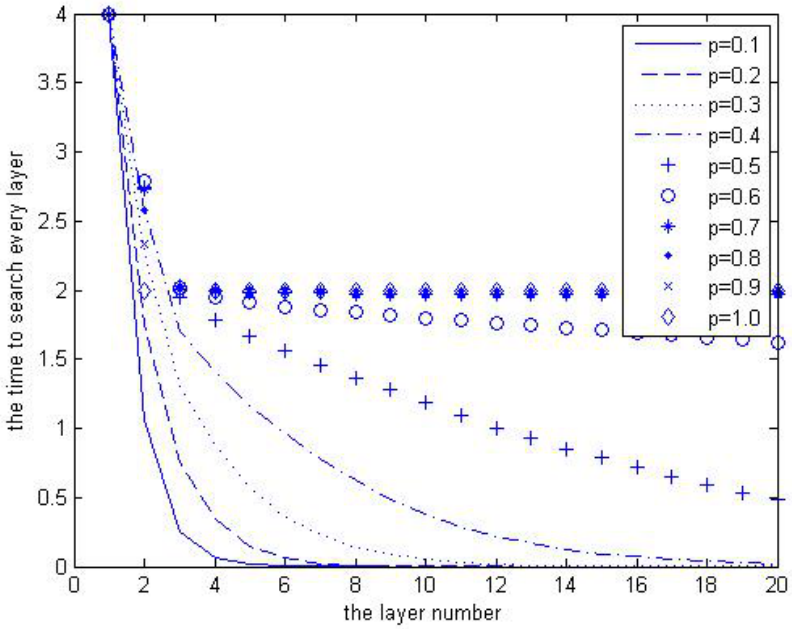

Figure 3.5: Searching time for the mesh network in case 3.3.2 as signature probability is varied

values of p result in the signature searching processing stopping earlier as irregular signature free boundaries form around the mesh network that prevent further signature searching. When the probability of signature is equal or higher than 0.7, for those higher layer the searching time is exactly $2\overline{X}$. That is because the probability that the top node has a signature and meanwhile the two neighboring nodes have no signatures is almost impossible. Thus the searching time for those high layers is exactly $2\overline{X}$.

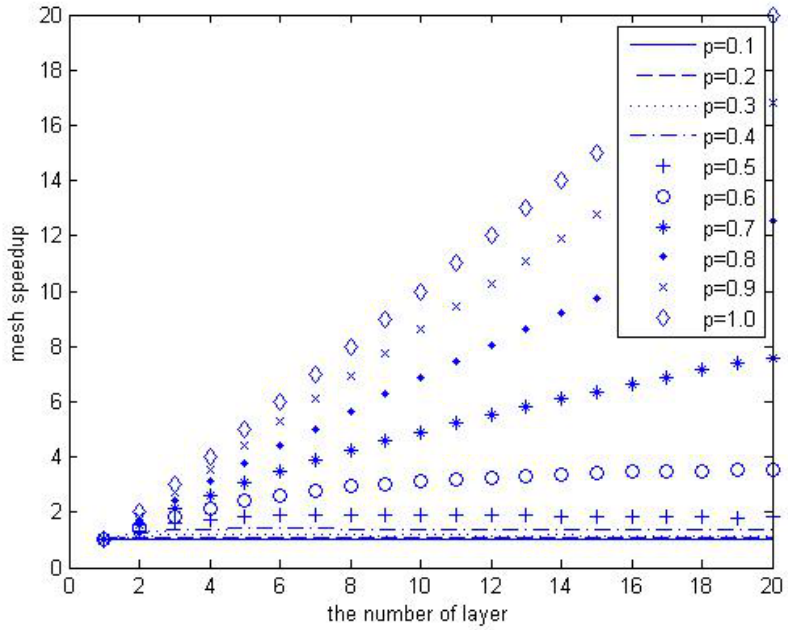

Figure 3.6: Speedup 1 when the number of signatures is unknown and only the nodes whose parent node has a signature can have a signature in the mesh network

The speedup for case 3.3.1 and case 3.3.2 (speedup 1) has been depicted in figure 3.6 . From it we can conclude that when the probability of a signature

is below 0.6, the speedups are saturated at less than 4. We can not reduce much searching time by using the method in case 3.3.2 when the probability of a signature is below 0.6 as the number of layers increases. That is because when the probability of a signature is below 0.6, the searching time for every layer in case 3.3.1 will decrease as the number of layers increases (see figure 3.4). While the probability of a signature is more than 0.6, the speedup rises much faster. That is because in figure 3.4 we know the the searching time will go up quickly, while the searching time will be $2\overline{X}$ in figure 3.5.

3.3.3 All the files in every layer are searched in parallel, layers are searched sequentially.

The searching time for every layer is $\overline{X}$. The total search time is thus $N\overline{X}$. The speedup for 3.3.1 and case 3.3.3 (speedup 2) has been depicted in figure 3.7. It can be seen from figure 3.7 the speedup is larger than the speedup in figure 3.6 when the probability of signature is larger than 0.6, for the searching time for every layer is the same $\overline{X}$. When the probability of signature is lower than 0.6, the same as figure 3.6, the speedup does not increase when the number of layers increases.

3.3.4 All the files in the mesh network are searched in parallel

The expected search time in this case is only $\overline{X}$. The speedup (speedup 3) for case 3.3.4 and case 3.3.1 has been plotted in figure 3.8. It can be seen

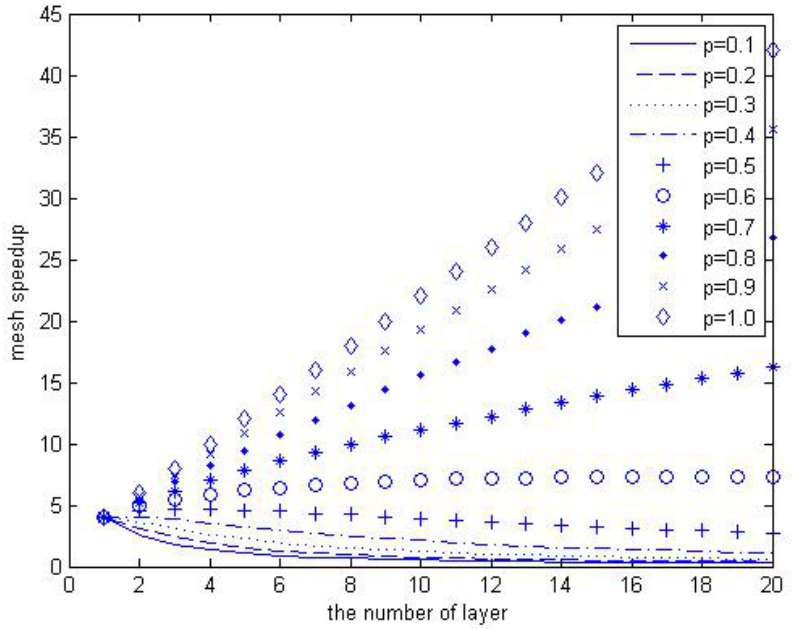

Figure 3.7: Speedup 2 when the number of signatures is unknown and only the nodes whose parent node has a signature can have a signature in the mesh network

that the speedup increases much faster as the number of layers increase, than the speedup in figure 3.7 when the probability of a signature is larger than 0.6, and when the probability of a signature is lower than 0.6, the speedup does not increase as the number of layers increase.

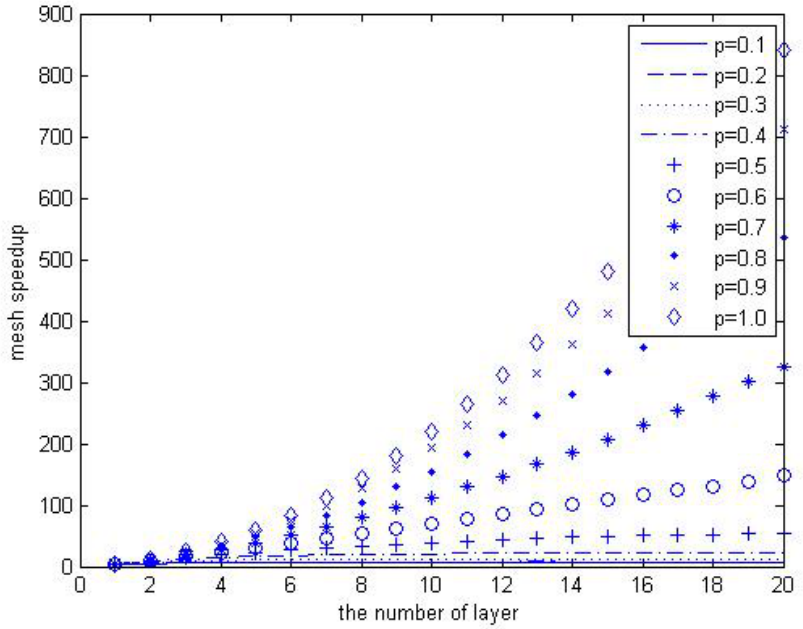

Figure 3.8: Speedup 3 when the number of signature is unknown and only the nodes whose parent node has a signature can have a signature in mesh network

3.4 Mesh Networks: Circuit Switched and Wormhole Routing

We discuss a type of mesh network as shown in figure 3.8. It is proposed for circuit-switched and wormhole routing. [36] In circuit switching and wormhole routing the communication delay does not depend on the distance between originator node and lower layer nodes that are searched by the originator node. As a result it is possible to search the signature of those nodes very far from the originator. Each scattering consists of two phases: chess

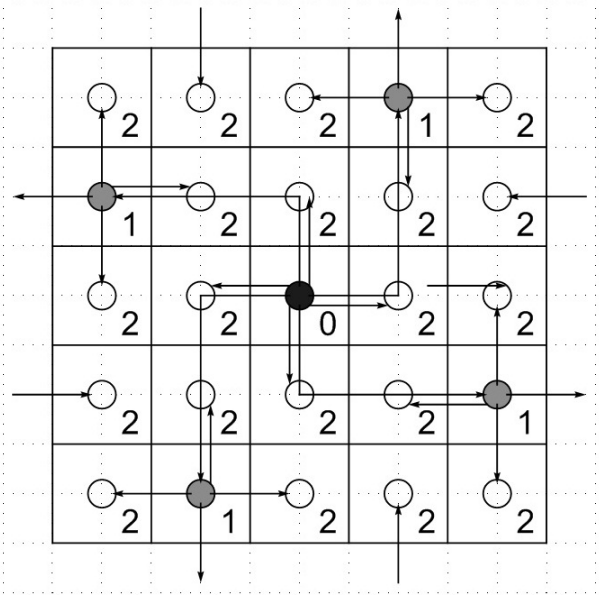

Figure 3.9: Circuit-switched and wormhole routing mesh network

queen moves and then cross moves using the torus wrap-around connections. The searching process pattern recursively repeats itself in sub-meshes of five times smaller side size. Let p denote the number of processors used in each phase of the searching process. Here the communication path patterns are repeated in sub-meshes with $p+1$ times shorter side size, and all p ports of the active processors are busy in each searching phase. In the first searching phase the originator searches p processors. In the next phase each searched processor search their p new processors. Thus, $p(p+1)^{i-1}$ processors are being searched in phase i. At the end of phase i the number of processors that have been searched is $(p+1)^i$. The searching process can be viewed as a tree which can be called a $(p+1)$ - nominal tree. Let us call a layer the set of processors activated in the same phase, and hence on the same

level of the $(p+1)$ - nominal tree. However, it is different from the searching signature process in the tree network. For the tree network, the nodes which have searched their children nodes will not search any other nodes, but in this mesh structure, each node will be searched once, but later each node will search other nodes during the search process. Assume that there are m processors(nodes) should be searched and the expected time to search one node is $\overline{X}$, so the total search time is $T = \log_{p+1} m \cdot \overline{X}$.

3.5 Hypercubes

In this section we assume that the system is homogeneous [7]. A hypercube is an n-dimensional generalization of a square (n = 2) and a cube (n = 3). [wikipedia] The search process in a hypercube is commanded by activating processor along consecutive hypercube dimensions. For example, P_0 search its file and then commands P_1 to search its own file along dimension 0, then P_0, P_1 command P_2, P_3, respectively, along dimension 1, etc. This scattering method uses binomial trees embedded in the log m - dimensional mesh of side size 2. Note that this is a special case of $(p + 1)$ - nominal tree. Let $d = \log m$, which m is the total number of processors here. Each processor of the hypercube is labeled with d - bit binary number such that neighboring nodes differ in exactly one bit. Thus, there are $\log m$ ports in each node. Suppose the originator node P_0 has label 0. Then, its direct neighbors have exactly one bit equal to 1 in their labels. If a processors is in a distance of two hops from the originator, it will have two bits equal to 1 in their labels. So,

processors with i number of 1_s in their labels are i number of hops away from the originator. Let us call a layer the set of processors in equal hop from the originator. The number of processors in layer i is $\binom{d}{i}$. A processor in layer i may command layer $i+1$ through $d-i$ ports. So in the hypercubes network, any node can be the original node to command other nodes to search their own files, assume the time to search one node is $\overline{X}$, so the total searching time should be $\log m \cdot \overline{X}$.

3.6 Searching time comparison for different types of network

The four types of networks considered are tree networks, mesh networks: store and forward, mesh networks: circuit switched and wormhole routing (discussed in the supplementary file) and hypercubes (discussed in supplementary file). Each of these has advantages and disadvantages. Now we want to compare their searching time if the total number of nodes is fixed: N. For the tree network, assume that the number of nodes in every subtree, n, is the same, which is a homogeneous case. The height of the tree should be: $1+n+n^2+\ldots+n^{H-1} = N$, then $H = \log_n((n-1)N+1)$. The total searching time for case 3.1.1.2 is $T_1 = nH\overline{X} = n\log_n((n-1)N+1)\overline{X}$. The searching time is plotted in figure 3.9 (here $N = 100$) for these four types of network. We can see that when the number of nodes in every subtree increases, the searching time also increases. For the store-and-forward mesh network, the

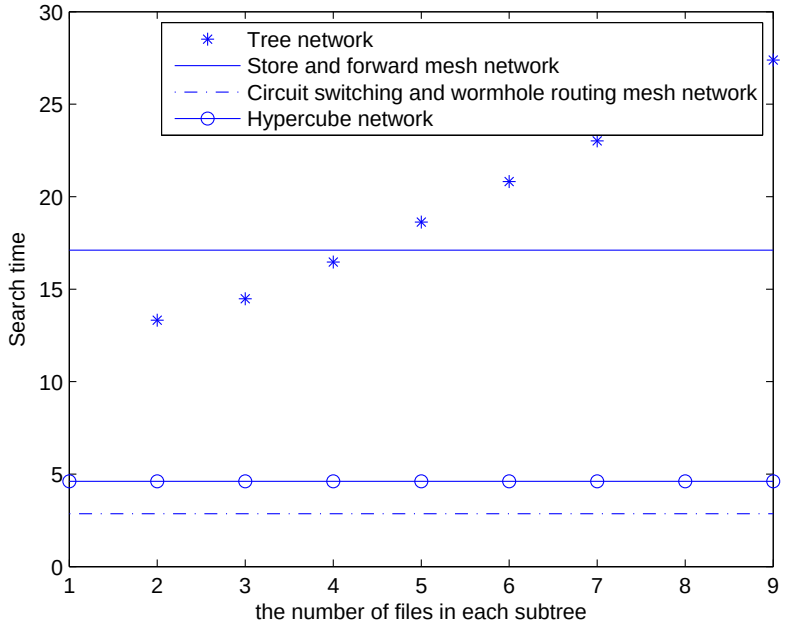

Figure 3.10: Four types of networks comparison

number of layers should be: $1+4+\ldots+4(H-1)=N$, then $H=\frac{1+\sqrt{2N-1}}{2}$. The searching time should be $T_2=(2H+2)\overline{X}=(3+\sqrt{2N-1})\overline{X}$. For the circuit switching and wormhole routing mesh network, the searching time is $T_3=\log_{p+1}N\overline{X}$. For the hypercube network, the searching time is $T_4=log_2N\overline{X}$. In figure 3.10, we can see that the searching time for circuit switching and the wormhole routing mesh network is short, but this type network is hard to realize. The searching time for hypercubes is also very short, and this type can be easily realized in low dimensionality. For the tree network and store-and-forward network, which are easy to realize, if the number of nodes in every subtree is more than 4, the searching time for the store and forward mesh network is shorter, for the tree network when the

number of nodes in every subtree is less than or equal to 4, the searching time for the tree network is shorter.

3.7 Conclusion

It has been demonstrated that the expected search time for signatures in a wide variety of search scenarios for tree, mesh and hypercube networks, where load distribution time is not considered, can be calculated either analytically or through simulation. This should also be possible for other types of interconnection networks. Future research should consider other types of file structures or statistical assumptions. This work is of interest in a wide variety of applied areas involving signature searching.

Chapter 4

Speedup Evaluation for a Cyclic Network with Multiple Paths

4.1 Introduction

Divisible loads are computing and communication loads that are perfectly partitionable among processors and links, respectively. Divisible load scheduling seeks to assign loads to processors and links in a scheduled manner so as to minimize solution time (i.e. makespan) given the scheduling policy, the interconnection network, the processor and link speeds and the computing and communication intensities. There have been over 130 journal papers [http://www.ece.sunysb.edu/ tom/] on divisible load scheduling since the first papers in 1988 by Agrwal and Jagadish [44] and Cheng and Robertazzi

[2].

Most work to date has involved load distribution over trees or over spanning trees embedded in other interconnection networks. For instance, previously linear networks have been discussed in [2][45][46][47][48]. Bus network has been discussed in [3][49][50][23][51]. Hypercube network were examined in [7][14]. Two dimensional network has been discussed in [52] and tree network has been discussed in [53][54][55][9][5].

Put another way, most work to date does not allow cycles in the load distribution (spanning) graph interconnects. In this paper we will discuss a new type of network with two paths for load distribution as shown in Figure 4.1. It is a cyclic network of two four nodes, for a first study. At first the total load is on the original processor 0, it will transfer load to the other processors to do the computation work together. The task here is to find the best distribution load strategy to process the load in minimum time. From Figure 4.1 we know that processor 0 should distribute the load to processor 1 and processor 2, the load for processor 3 is transferred from processor 1 and processor 2.

In this chapter network speedup is found for a wide variety of scheduling policy assumptions.

This chapter is organized as follow. Section two will discuss this special network when load distribution involves processors with front-end processors for communication off-loading (i.e. computation and communication can occur for a processor at the same time). In specific cases we will examine processors with homogeneous link speeds and with different link speeds.

Section three will examine networks with processors without front-ends.

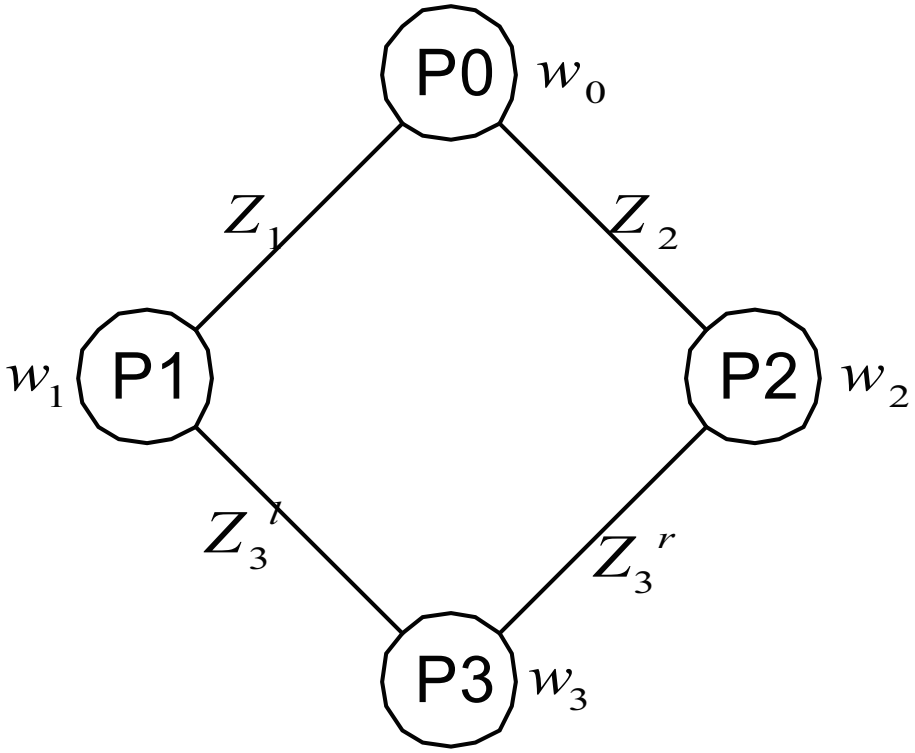

Figure 4.1: Cyclic network

4.2 Load distribution with Front-end

Consider the simple tree network as shown in Figure 4.1. The first case we discuss is the tree network with front-end. That means the processors can compute and communicate at the same time. Let us first introduce the following notation.

α_i: The fraction of measurement data that is assigned to processor i by

the originating processor.

w: A constant that is inversely proportional to the computation speed of any processor in the network. Any processor can process the entire load in time wT_{cp}.

z: A constant that is inversely proportional to the speed of each link. The entire load can be transmitted over a link in time zT_{cm}.

T_{cp}: The time that it takes the i_{th} processor to process the entire load when $w = 1$. The time for arbitrary w is wT_{cp}.

T_{cm}: The time that it takes processor to transmit all the measurement data by the network link when $z = 1$. The time for arbitrary z is zT_{cm}.

In Figure 4.1 we have four processors: $P0, P1, P2, P3$, and they have their own inverse computation speed: w_0, w_1, w_2, w_3, and the inverse link speed between $P0$ and $P1$ is z_1, between $P0$ and $P2$ is z_2, between $P1$ and $P3$ is ${z_3}^l$ and between $P2$ and $P3$ is ${z_3}^r$. For simplicity, we assume $z_1 = z_2 = z$ and $w_0 = w_1 = w_2 = w$. We only discuss different link speed or different computation speed for processor 3 here.

4.2.1 Homogeneous link speeds and the computation speed of Processor 3 is different

In this subsection, we assume that ${z_3}^l = {z_3}^r = z = z_1 = z_2$, but w_3 is not equal to $w = w_1 = w_2$. We also assume sequential distribution from the host. We define that for $P3$ the load transmitted from $P1$ is ${\alpha_3}^l$ and the load transmitted from $P2$ is ${\alpha_3}^r$. Suppose $P3$ can receive the load from $P1$ and $P2$ at the same time. Based on [53] we plot out the timing diagram as shown in Figure 4.2.

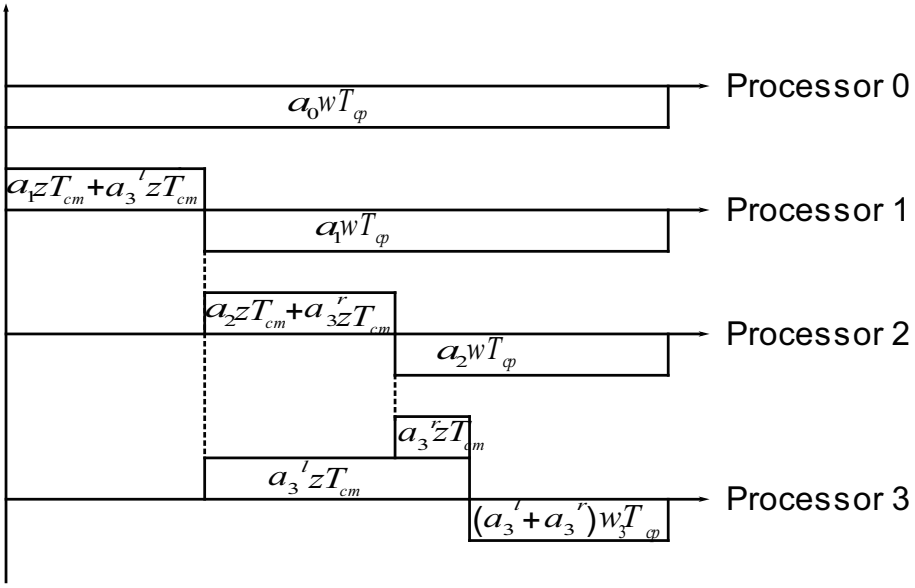

Figure 4.2: Timing diagram 1 in case 2.1

To obtain a minimum finish time, we should let all the processors finish their computation work at the same time, and also $P3$ should finish receiving the load from $P1$ and $P2$ at the same time. If this is not true, load can be reallocated so $P3$ finishes receiving load from $P1$ and $P2$ at the same time.

So we obtain the timing equations:

$$\alpha_0 w T_{cp} = \alpha_1 z T_{cm} + {\alpha_3}^l z T_{cm} + \alpha_1 w T_{cp} \tag{4.1}$$

$$\alpha_1 w T_{cp} = \alpha_2 z T_{cm} + {\alpha_3}^r z T_{cm} + \alpha_2 w T_{cp} \tag{4.2}$$

$$\alpha_2 w T_{cp} = {\alpha_3}^r z T_{cm} + ({\alpha_3}^l + {\alpha_3}^r) w_3 T_{cp} \tag{4.3}$$

$${\alpha_3}^l z T_{cm} = \alpha_2 z T_{cm} + {\alpha_3}^r z T_{cm} + {\alpha_3}^r z T_{cm} \tag{4.4}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + {\alpha_3}^l + {\alpha_3}^r = 1 \tag{4.5}$$

There are five equations and five unknowns. We assume that $\frac{zT_{cm}}{wT_{cp}} = \lambda$, usually the communication speed is much faster than computation speed, so $\lambda << 1$. Also we assume $\frac{w_3}{w} = T$. So from equation (4.1)(4.2)(4.3)(4.4) we calculate that:

$$\frac{{\alpha_3}^l}{{\alpha_3}^r} = \frac{\lambda + T + 2}{1 - T} \tag{4.6}$$

From equation (4.6) we know that T should less than 1, otherwise it makes no sense. This means the computation speed for processor 3 should be faster than the other processors. We define $M_1 = \frac{{\alpha_3}^l}{{\alpha_3}^r}$ and allied with equation (4.5) we calculate out ${\alpha_3}^r$ and α_0 here so that:

$${\alpha_3}^r = \frac{1}{M_1(3T + \lambda^2 + 3\lambda + T\lambda + 1) + 3T + T\lambda + \lambda + 1} \tag{4.7}$$

$$\alpha_0 = {\alpha_3}^r (M_1 T + M_1 \lambda^2 + 2M_1 \lambda + M_1 T \lambda + T\lambda + T) \tag{4.8}$$

So the speedup compared with the finish time using just one processor, whose computation speed is w, to complete the total load is

$$speedup^1 = \frac{wT_{cp}}{\alpha_0 wT_{cp}} = \frac{1}{\alpha_0} \tag{4.9}$$

Refering to [48], we can get a more efficient load distribution strategy in our network. The timing diagram is plotted as in figure 4.3.

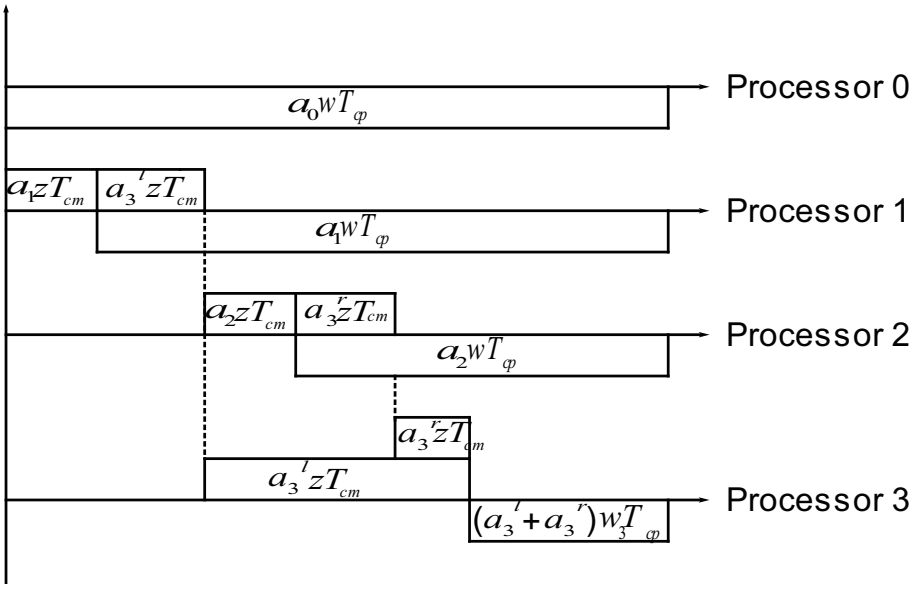

Figure 4.3: Timing diagram 2 in case 2.1

The difference here is we begin processor one's computation work as soon as we finish transferring α_1's part of the load. Here we first transfer α_1's part of the load, then transfer $\alpha_3{}^l$'s part of the load from processor 0 to processor 1. For processor 2 it is the same. So we obtain these timing equations.

$$\alpha_0 wT_{cp} = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} \tag{4.10}$$

$$\alpha_1 w T_{cp} = \alpha_2 z T_{cm} + {\alpha_3}^l z T_{cm} + \alpha_2 w T_{cp} \tag{4.11}$$

$$\alpha_2 w T_{cp} = 2{\alpha_3}^r z T_{cm} + ({\alpha_3}^l + {\alpha_3}^r) w_3 T_{cp} \tag{4.12}$$

$${\alpha_3}^l z T_{cm} = \alpha_2 z T_{cm} + {\alpha_3}^r z T_{cm} + {\alpha_3}^r z T_{cm} \tag{4.13}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + {\alpha_3}^l + {\alpha_3}^r = 1 \tag{4.14}$$

Also like the case we discussed above we assume that $\frac{zT_{cm}}{wT_{cp}} = \lambda$ and $\frac{w_3}{w} = T$, then we obtain:

$$\frac{{\alpha_3}^l}{{\alpha_3}^r} = \frac{2\lambda + T + 2}{1 - T} \tag{4.15}$$

We define $M_2 = \frac{{\alpha_3}^l}{{\alpha_3}^r}$, then

$${\alpha_3}^r = \frac{1}{M_2(3T + 2\lambda^2 + 4\lambda + T\lambda + 1) + 3T + T\lambda + 2\lambda + 1} \tag{4.16}$$

$$\alpha_0 = {\alpha_3}^r (M_2 T + 2M_2\lambda^2 + 2M_2\lambda + M_2 T\lambda + T\lambda + T) \tag{4.17}$$

The *speedup*[2] here is equivalent to equation (4.9).

Now we consider a third load distribution strategy, the timing diagram shown in figure 4.4.

Here processor 3 begins to work as soon as ${\alpha_3}^l$'s part load is transferred from processor 1 completed and finishes this part job and then begin ${\alpha_3}^r$'s part load. There should be no idle time for Processor 3 as soon as it begins

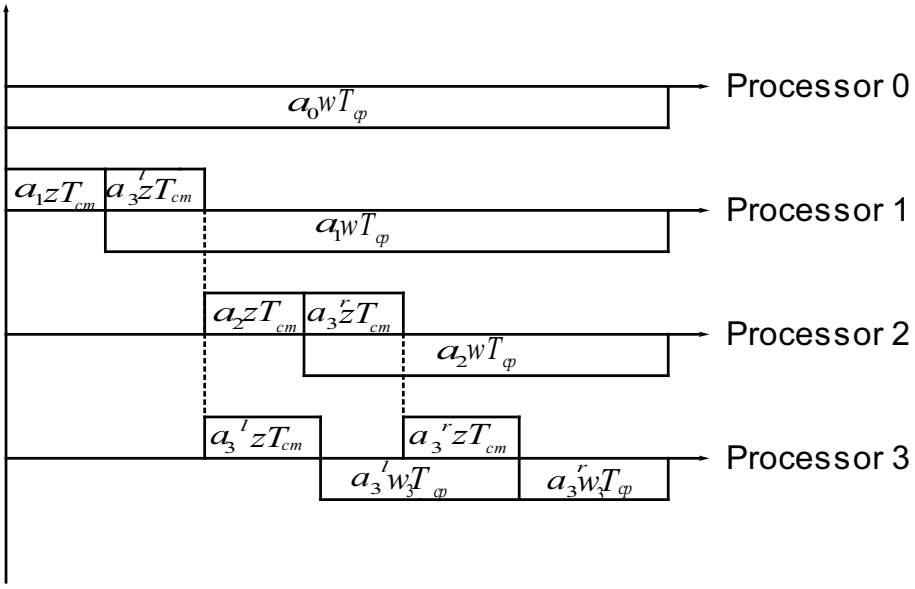

Figure 4.4: Timing diagram 3 in case 2.1

to compute $\alpha_3{}^l$'s part of the load, if not, we can distribute more load from $\alpha_3{}^r$'s part of the load to $\alpha_3{}^l$'s part of the load. So we obtain these timing equations.

$$\alpha_0 w T_{cp} = \alpha_1 z T_{cm} + \alpha_1 w T_{cp} \tag{4.18}$$

$$\alpha_1 w T_{cp} = \alpha_2 z T_{cm} + \alpha_3{}^l z T_{cm} + \alpha_2 w T_{cp} \tag{4.19}$$

$$\alpha_2 w T_{cp} = 2\alpha_3{}^r z T_{cm} + \alpha_3{}^r w_3 T_{cp} \tag{4.20}$$

$$\alpha_3{}^l z T_{cm} + \alpha_3{}^l w_3 T_{cp} = \alpha_2 z T_{cm} + \alpha_3{}^r z T_{cm} + \alpha_3{}^r z T_{cm} \tag{4.21}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.22}$$

Also like case we discussed before we assume that $\frac{zT_{cm}}{wT_{cp}} = \lambda$ and $\frac{w_3}{w} = T$, then we obtain:

$$\frac{{\alpha_3}^l}{{\alpha_3}^r} = \frac{2\lambda^2 + \lambda T + 2\lambda}{\lambda + T} \tag{4.23}$$

We notate $M_3 = \frac{{\alpha_3}^l}{{\alpha_3}^r}$, then

$${\alpha_3}^r = \frac{1}{M_3(2T + 2\lambda^2 + 4\lambda + T\lambda + 1) + 3T + T\lambda + 2\lambda + 1} \tag{4.24}$$

$$\alpha_0 = {\alpha_3}^r(M_3T + 2M_3\lambda^2 + 2M_3\lambda + M_3T\lambda + T\lambda + T) \tag{4.25}$$

The *speedup*[3] here is equivalent to equation (4.9).
There are two other distribution load strategy almost the same as the third timing diagram, just changing some orders. We plot timing diagram 4 and timing diagram 5 in figure 4.5 and figure 4.6.

We do not list the equation for these two cases. The speedup corresponding with timing diagram 4 and 5 is *speedup*[4] and *speedup*[5]. Lastly we set a network for reference which is plotted in figure 4.7.

In figure 4.7 there is no link between Processor 2 and Processor 3. So the timing diagram is showed in figure 4.8 and the equations are shown as below:

$$\alpha_0 wT_{cp} = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} \tag{4.26}$$

$$\alpha_1 wT_{cp} = \alpha_2 zT_{cm} + {\alpha_3}^l zT_{cm} + \alpha_2 wT_{cp} \tag{4.27}$$

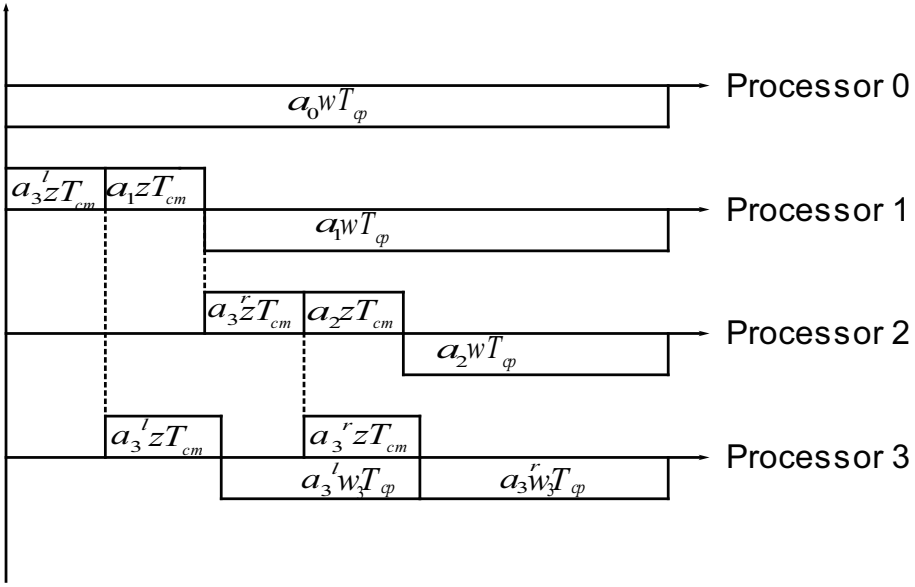

Figure 4.5: Timing diagram 4 in case 2.1

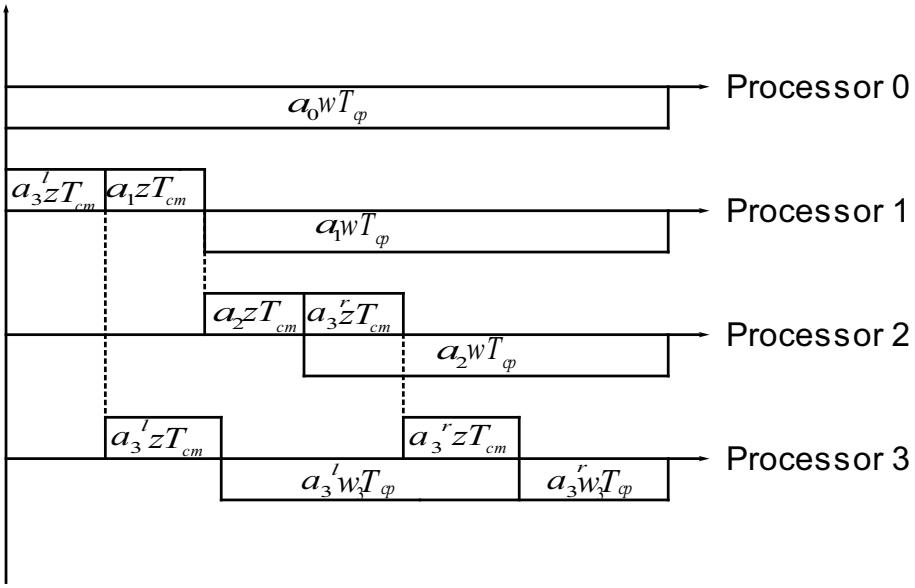

Figure 4.6: Timing diagram 5 in case 2.1

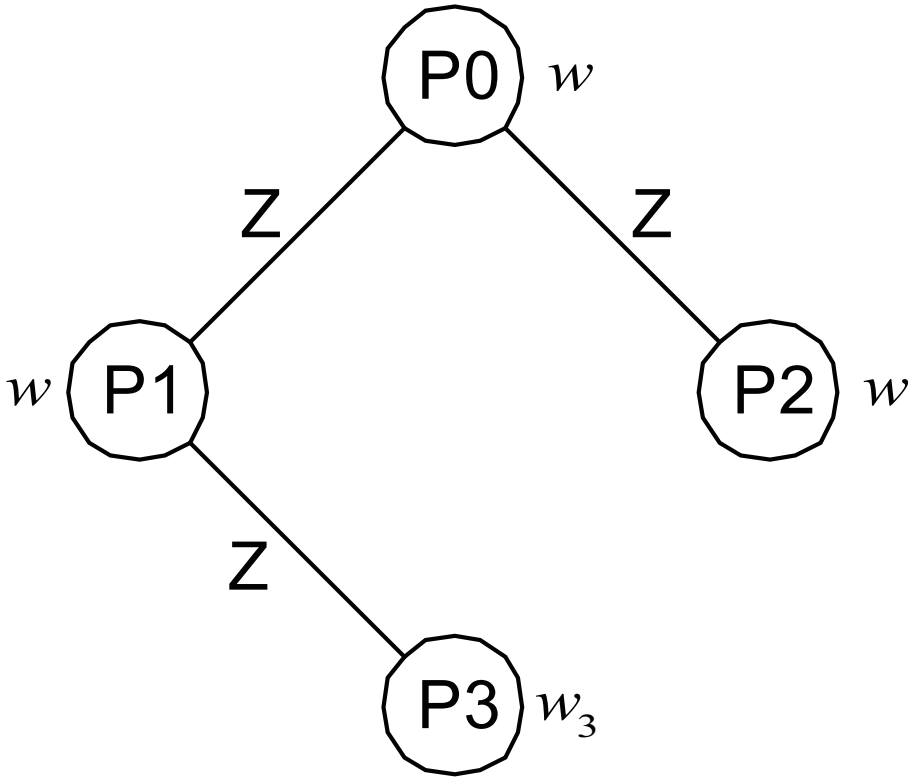

Figure 4.7: Tree network for reference in case 2.1

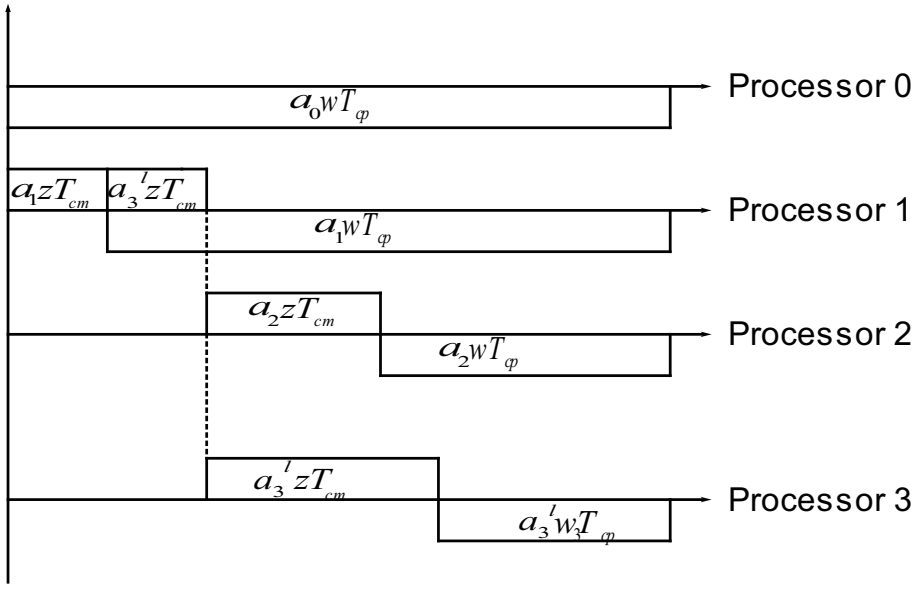

Figure 4.8: Timing diagram for the reference tree network in case 2.1

$$\alpha_1 w T_{cp} = 2\alpha_3{}^l z T_{cm} + \alpha_3{}^l w_3 T_{cp} \tag{4.28}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3{}^l = 1 \tag{4.29}$$

From equation (4.26)(4.27)(4.28)(4.29) we get:

$$\alpha_3{}^l = \frac{1}{2\lambda^2 + 4\lambda + 2T + \lambda T + 1 + \frac{\lambda+T}{\lambda+1}} \tag{4.30}$$

$$\alpha_0 = \alpha_3{}^l(2\lambda^2 + 2\lambda + \lambda T + T) \tag{4.31}$$

The speedup here defined as $speedup^0$ is equivalent to equation (4.9).

We compare $speedup^0$, $speedup^1$, $speedup^2$, $speedup^3$, $speedup^4$, $speedup^5$, $speedup^6$ in figure 4.9. In this figure $T_{cp} = T_{cm} = 1$, $w = 1, z = 0.05$, and

w_3 is from 0.1 to 1. We see from the figure that $speedup^2$ and $speedup^3$ are the best distribution strategy. When w_3 is less than a certain number (about 0.3 in figure 4.9), $speedup^3 > speedup^2$. That is because when w_3 is smaller, it means that the computation speed of processor 3 is faster, and we should let the processor begin to work as early as possible. When the computation speed of processor 3 is slower, processor 3 can not complete ${\alpha_3}^l$'s part load when processor 2 transfers ${\alpha_3}^r$'s to processor 3 completely. In that situation, we distribute more load to ${\alpha_3}^r$'s part load and let the left and right link for processor 3 complete the load communication process at the same time. Also we can find from figure 4.9 that $speedup^3$ is always larger than $speedup^4$ and $speedup^5$, that is to say, we should let processor 1 begin to compute as soon as possible, like the linear network, we should transfer the load to the neighboring processor first regardless of the computation speed of other processor. This graph demonstrates that simply adding a link does not yield the best performance. The scheduling policy used is also important.

4.2.2 Homogeneous computation speeds and the link speeds for processor 3 are different

Here one can set the computation speeds of all the processors to be the same w, the communication speeds from processor 0 to be the same z, but the communication speeds for links to processor 3 to be different. The communication speed of the left link from processor 1 to processor 3 is z^l and the

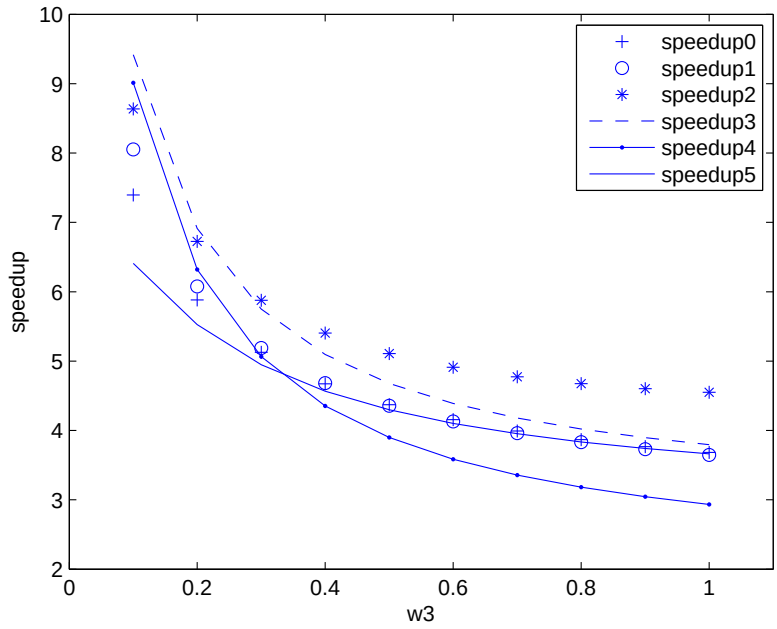

Figure 4.9: Speedups comparison with load distribution with frond-end and different computation speed for processor 3

communication speed of the right link from processor 2 to processor 3 is z^r. We directly analyze the second strategy in the section discussed before. The timing diagram is plotted in figure 4.10.

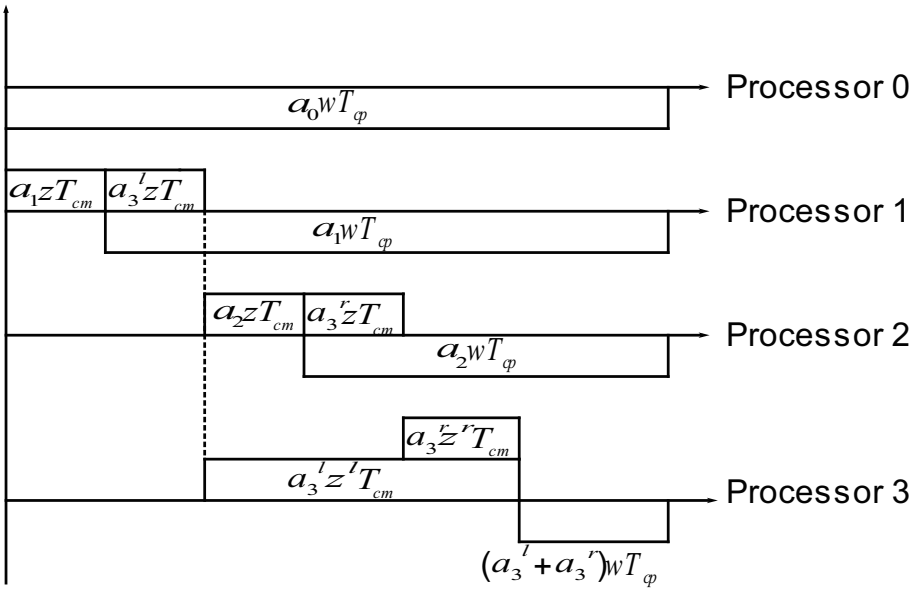

Figure 4.10: Timing diagram 1 in case 2.2

The related timing equations are:

$$\alpha_0 wT_{cp} = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} \tag{4.32}$$

$$\alpha_1 wT_{cp} = \alpha_2 zT_{cm} + \alpha_3{}^l zT_{cm} + \alpha_2 wT_{cp} \tag{4.33}$$

$$\alpha_2 wT_{cp} = \alpha_3{}^r zT_{cm} + \alpha_3{}^r z^r T_{cm} + (\alpha_3{}^l + \alpha_3{}^r) wT_{cp} \tag{4.34}$$

$$\alpha_3{}^l z^l T_{cm} = \alpha_2 zT_{cm} + \alpha_3{}^r zT_{cm} + \alpha_3{}^r z^r T_{cm} \tag{4.35}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.36}$$

From equation (4.32)(4.33)(4.34)(4.35)(4.36) we get:

$$\frac{{\alpha_3}^l}{{\alpha_3}^r} = \frac{\lambda + \lambda T^r + T^r + 2}{T^l - 1} \tag{4.37}$$

Here $\lambda = \frac{zT_{cm}}{wT_{cp}}$, $T^l = \frac{z^l}{z}$, $T^r = \frac{z^r}{z}$. From equation (4.37) we find T^l should be larger than 1, otherwise $\frac{{\alpha_3}^l}{{\alpha_3}^r}$ will be less than 0, which is impossible, that means $z^l > z$, which means the communication speed for the left link of processor 3 is slower than the communication speed for the links from processor 0 (We select the left link for processor 3 to be used to transfer the load first. If we choose the right link to be used to transfer the load, here T^l will be replaced by T^r). We define $M_1 = \frac{{\alpha_3}^l}{{\alpha_3}^r}$, then from equation (4.33)(4.34) we obtain $\alpha_1 = {\alpha_3}^r N_1$, here $N_1 = M_1\lambda + M_1\lambda T^l + M_1 + 1$. So

$${\alpha_3}^r = \frac{1}{N_1\lambda + 2N_1 + \lambda + \lambda T^r + 2M_1 + 2} \tag{4.38}$$

The speedup here defined as *speedup*[1] is identical to equation (4.9). Then we analyze the second strategy whose timing diagram is shown in figure 4.11.

From figure 4.11 we obtain

$$\alpha_0 wT_{cp} = \alpha_1 zT_{cm} + \alpha_1 wT_{cp} \tag{4.39}$$

$$\alpha_1 wT_{cp} = \alpha_2 zT_{cm} + {\alpha_3}^l zT_{cm} + \alpha_2 wT_{cp} \tag{4.40}$$

$$\alpha_2 wT_{cp} = {\alpha_3}^r zT_{cm} + {\alpha_3}^r z^r T_{cm} + {\alpha_3}^r wT_{cp} \tag{4.41}$$

$${\alpha_3}^l z^l T_{cm} + {\alpha_3}^l wT_{cp} = \alpha_2 zT_{cm} + {\alpha_3}^r zT_{cm} + {\alpha_3}^r z^r T_{cm} \tag{4.42}$$

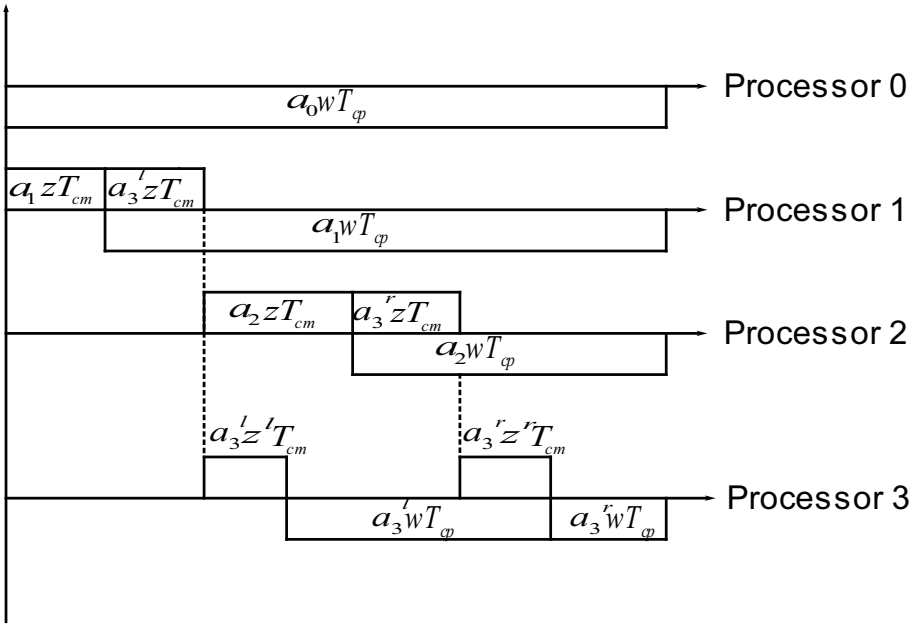

Figure 4.11: Timing diagram 2 in case 2.2

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.43}$$

From equation (4.39)(4.40)(4.41)(4.42)(4.43) we get

$$\frac{\alpha_3{}^l}{\alpha_3{}^r} = \frac{\lambda^2 + \lambda^2 T^r + 2\lambda + \lambda T^r}{1 + \lambda T^l} \tag{4.44}$$

Here we define $M_2 = \frac{\alpha_3{}^l}{\alpha_3{}^r}$, and $N_2 = M_2\lambda + M_2\lambda T^l + M_2 + 1$. So we obtain

$$\alpha_3{}^r = \frac{1}{N_2\lambda + 2N_2 + \lambda + \lambda T^r + M_2 + 2} \tag{4.45}$$

The speedup defined as *speedup*^{2} here is identical to equation (4.9).

We compare these two policies to a network with no right link to node 3 in figure 4.12.

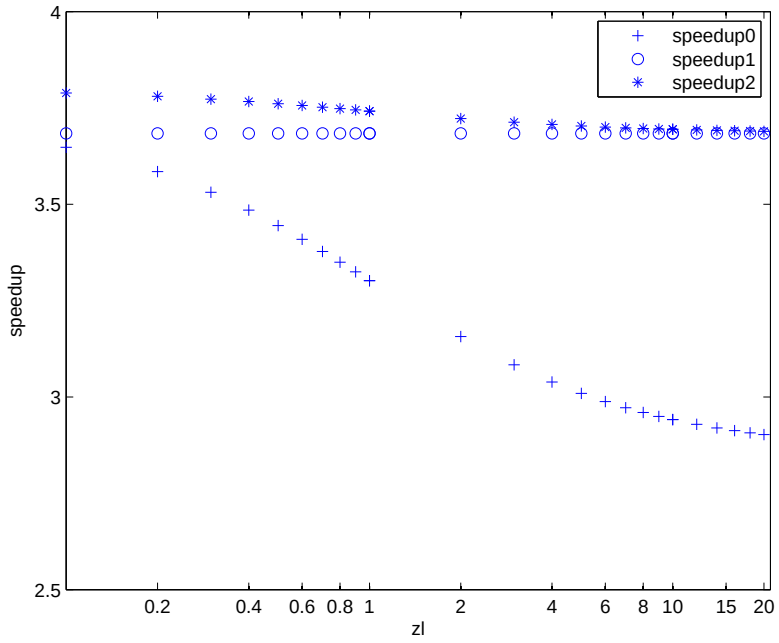

Figure 4.12: Speedups comparison with load distribution with front-end and different link speed for processor 3

In figure 4.12 $speedup^0$ stand for the reference, $T_{cp} = T_{cm} = 1$, $w = 1$, $z = z_l = 0.05$, z_r is from 0.1 to 1.0. we can see from figure 4.12 that $speedup^2$ is always larger than $speedup^1$. We now find an expression to demonstrate it.

$$speedup^2 > speedup^1 \tag{4.46}$$

$$\frac{N_2\lambda + 2N_2 + \lambda + \lambda T^r + M_2 + 2}{N_2\lambda + N_2} \tag{4.47}$$

$$> \frac{N_1\lambda + 2N_1 + \lambda + \lambda T^r + 2 * M_1 + 2}{N_1\lambda + N_1} \tag{4.48}$$

$$N_1(\lambda + \lambda T^r + 2) + N_1 M_2 \tag{4.49}$$

$$> N_2(\lambda + \lambda T^r + 2) + 2N_2 M_1 \tag{4.50}$$

$$(M_1 - M_2)(\lambda + \lambda T^l + 1)(\lambda + \lambda T^r + 2) > \tag{4.51}$$

$$M_1 M_2(\lambda + \lambda T^l + 1) + 2M_1 - M_2 \tag{4.52}$$

$$(\frac{1}{\lambda} + 1)(\lambda + \lambda T^l + 1)(\lambda + \lambda T^r + 2) > \tag{4.53}$$

$$(\lambda + \lambda T^r + T^r + 2)(\lambda + \lambda T^l + 1) + \tag{4.54}$$

$$\frac{1}{\lambda}(\lambda T^l + \lambda + 2) \tag{4.55}$$

$$1 + \frac{1}{\lambda} > \frac{1}{\lambda T^l + \lambda + 1} \tag{4.56}$$

Thus it is assured if this condition is met. So we can conclude that we should choose the second strategy when we have different link speed for processor 3 and the computation speed of processor 3 is the same as the other

processors.

4.2.3 Processor 0 does no processing

In this section the original processor processor 0 does not do computation work and only transfers load to the other processor. However, it can transfer its load to processor 1 and processor 2 at the same time (i.e. simultaneous distribution). First we consider the case the computation speed of processor 3 is different with respect to the other processors and all the link speeds are the same. The first strategy is showed in figure 4.13.

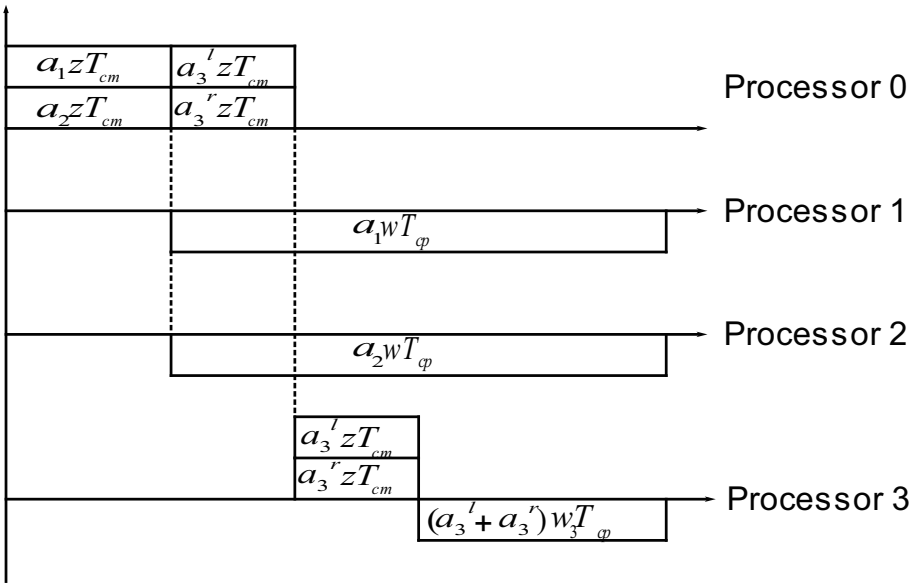

Figure 4.13: Timing diagram 1 with same link speed but different computation speed for processor 3 in case 4.2.3

From figure 4.13 we know that the communication speeds for processor 1 and processor 2 from processor 0 are the same and the computation speeds

for processor 1 and processor 2 are the same, so processor 0 will complete transferring the load to processor 1 and processor 2 at the same time, as well as for the load part to processor 3 through the left and right link. We have the equations shown below:

$$\alpha_1 w T_{cp} = \alpha_2 w T_{cp} \tag{4.57}$$

$$\alpha_3{}^l z T_{cm} = \alpha_3{}^r z T_{cm} \tag{4.58}$$

$$\alpha_2 w T_{cp} = 2\alpha_3{}^r z T_{cm} + (\alpha_3{}^l + \alpha_3{}^r) w_3 T_{cp} \tag{4.59}$$

$$\alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.60}$$

There are four equations and four unknowns. From equation(4.57)(4.58)(4.59)(4.60) we obtain $\alpha_3{}^l = \alpha_3{}^r = \frac{1}{4\lambda+4T+2}$, and $\alpha_1 = \alpha_2 = \alpha_3{}^r(2\lambda+2T)$, while $\lambda = \frac{ZT_{cm}}{wT_{cp}}$ and $T = \frac{w_3}{w}$. The speedup for this section is:

$$speedup = \frac{wT_{cp}}{\alpha_2 z T_{cm} + \alpha_2 w T_{cp}} = \frac{1}{\alpha_2(1+\lambda)} \tag{4.61}$$

Here the speedup defined as $speedup^1 = \frac{2\lambda+2T+1}{(\lambda+1)(\lambda+T)}$. Now we consider the second strategy as shown in figure 4.14.

Here we see that processor 0 first transfer $\alpha_3{}^l$'s load part to make processor 3 start to work as soon as possible. So we obtain:

$$\alpha_3{}^l z T_{cm} + \alpha_1 z T_{cm} + \alpha_1 w T_{cp} = \alpha_2 w T_{cp} + \alpha_2 z T_{cm} \tag{4.62}$$

$$\alpha_2 w T_{cp} = 2\alpha_3{}^r z T_{cm} + \alpha_3{}^r w_3 T_{cp} \tag{4.63}$$

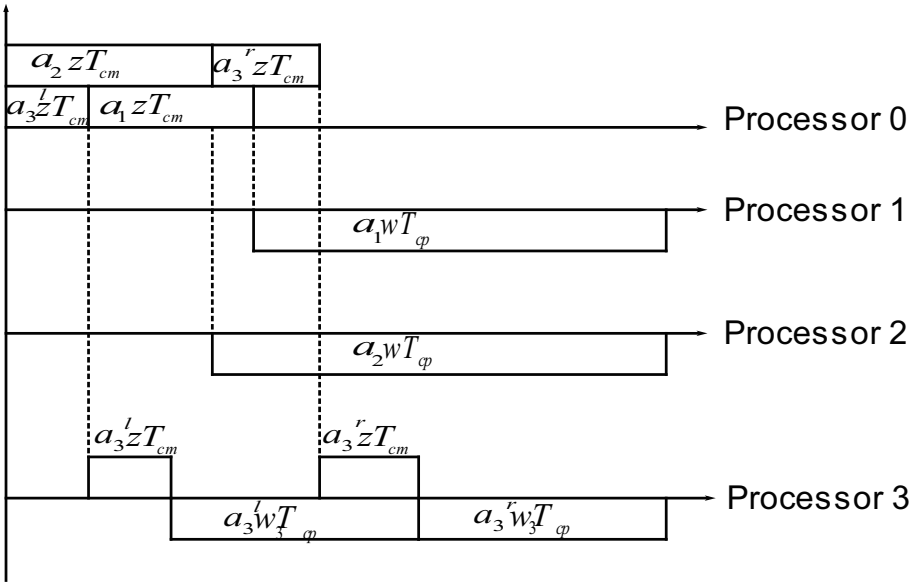

Figure 4.14: Timing diagram 2 with same link speed but different computation speed for processor 3 in case 2.3

$$2\alpha_3{}^l zT_{cm} + \alpha_3{}^l w_3 T_{cp} = \alpha_2 zT_{cm} + \alpha_3{}^r zT_{cm} + \alpha_3{}^r zT_{cm} \tag{4.64}$$

$$\alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.65}$$

From equation (4.62)(4.63)(4.64)(4.65) we obtain that:

$$\frac{\alpha_3{}^l}{\alpha_3{}^r} = \frac{2\lambda^2 + \lambda T + 2\lambda}{2\lambda + T} \tag{4.66}$$

Here we define $M_2 = \frac{\alpha_3{}^l}{\alpha_3{}^r}$, and $N_2 = \frac{2\lambda+T+2\lambda^2+\lambda T-M_2\lambda}{1+T}$, then $\alpha_1 = N_2\alpha_3{}^r$, refer to equation (4.61), we obtain the speedup here defined as $speedup^2$ is:

$$speedup^2 = \frac{N_2 + 2\lambda + T + M_2 + 1}{(2\lambda + T)(\lambda + 1)} \tag{4.67}$$

Compared with the reference network as speedup0 in which there is no

right link for processor 3 we plot out the figure shown as Figure 4.15.

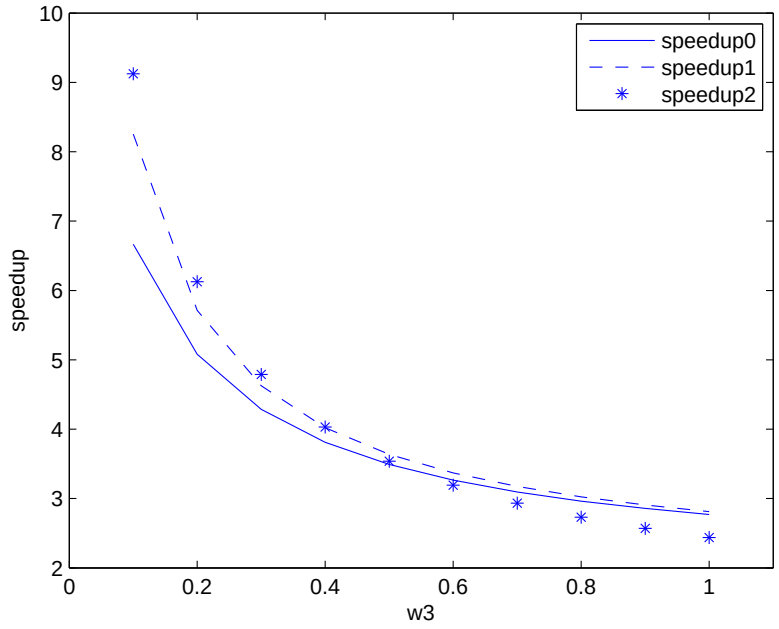

Figure 4.15: Speedups comparison when same link speed but different computation speed for processor 3 in case 4.2.3

In figure 4.15 $T_{cp} = T_{cm} = 1$, $w = 1$, $z = 0.05$, w_3 is from 0.1 to 1.0, and $speedup^0$ is the reference network speedup. From this figure we know that when $w_3 < 0.4$, the second strategy is the best for these parameters, that is because the computation speed of processor 3 is very fast so we choose the second strategy to let processor 3 begin to work as soon as possible. When $w_3 < 0.4$, the first strategy is the best for these parameters.

Now we consider the case that the left and right inverse link speed z^l and z^r for Processor 3 are different from the other link speed z. We assume the left inverse link speed is faster than right link speed, that is $z^l < z^r$. We

have the first strategy whose timing diagram is showed in figure 4.16 and the equations are shown below.

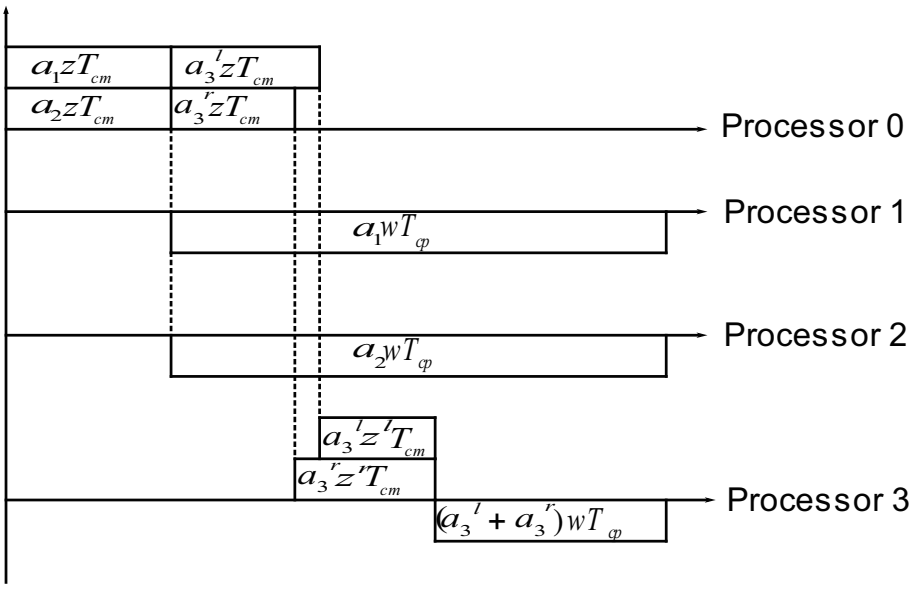

Figure 4.16: Timing diagram 1 with different link speed but same computation speed for all processors as processor 3 in case 4.2.3

$$\alpha_1 wT_{cp} = \alpha_2 wT_{cp} \tag{4.68}$$

$$\alpha_3{}^l zT_{cm} + \alpha_3{}^l z^l T_{cm} = \alpha_3{}^r zT_{cm} + \alpha^r z^r T_{cm} \tag{4.69}$$

$$\alpha_2 wT_{cp} = \alpha_3{}^r zT_{cm} + \alpha_3{}^r z^r T_{cm} + (\alpha_3{}^l + \alpha_3{}^r) wT_{cp} \tag{4.70}$$

$$\alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.71}$$

From equation (4.68)(4.69)(4.70)(4.71) we obtain that $\frac{\alpha_3{}^l}{\alpha_3{}^r} = \frac{1+T^r}{1+T^l}$, while $T^l = \frac{z^l}{z}$ and $T^r = \frac{z^r}{z}$. We define $M_1 = \frac{\alpha_3{}^l}{\alpha_3{}^r}$, then we obtain $\alpha_2 = \frac{\lambda+\lambda T^r+M_1+1}{2\lambda+2\lambda T^r+3M_1+3}$, and the speedup here defined as $speedup^1$ is the same as in equation (4.61).

The second strategy whose timing diagram is shown in figure 4.17 and the equations are shown below.

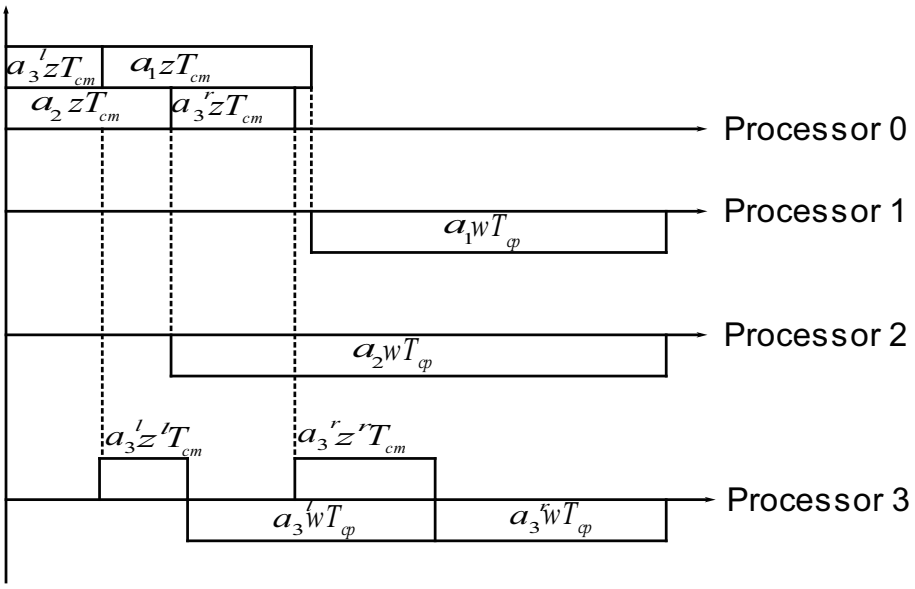

Figure 4.17: Timing diagram 2 with different link speed but same computation speed for all processors as processor 3 in case 4.2.3

$$\alpha_1 zT_{cm} + \alpha_1 wT_{cp} = {\alpha_3}^l z^l T_{cm} + ({\alpha_3}^l + {\alpha_3}^r) wT_{cp} \tag{4.72}$$

$$\alpha_2 wT_{cp} = {\alpha_3}^r zT_{cm} + {\alpha_3}^r z^r T_{cm} + {\alpha_3}^r wT_{cp} \tag{4.73}$$

$${\alpha_3}^l (zT_{cm} + z^l T_{cm} + wT_{cp}) = \alpha_2 zT_{cm} + {\alpha_3}^r (zT_{cm} + z^r T_{cm}) \tag{4.74}$$

$$\alpha_1 + \alpha_2 + {\alpha_3}^l + {\alpha_3}^r = 1 \tag{4.75}$$

From equation (4.72)(4.73)(4.74)(4.75) we obtain:

$$\frac{{\alpha_3}^l}{{\alpha_3}^r} = \frac{\lambda^2 + \lambda^2 T^r + 2\lambda + \lambda T^r}{\lambda + \lambda T^r + 1} \tag{4.76}$$

We define $M_2 = \frac{{\alpha_3}^l}{{\alpha_3}^r}$, and $N_2 = \frac{M_2\lambda T^l + M_2 + 1}{1+\lambda}$, then $\alpha_1 = N_2{\lambda_3}^r$. So we obtain ${\alpha_3}^r = \frac{1}{N_2+\lambda+\lambda T^r+2+M_2}$ and $\alpha_2 = \frac{\lambda+\lambda T^r+1}{N_2+\lambda+\lambda T^r+M_2+2}$. The speedup here notated as $speedup^2$ is the same as in equation (4.61).

Compared with the reference network in which processor 3 has no right link we plot out all the speedups for the three strategies in figure 4.18. In this figure, $T_{cm} = T_{cp} = 1$, $w = 1, z = 0.05$, and w_3 is from 0.1 to 1.

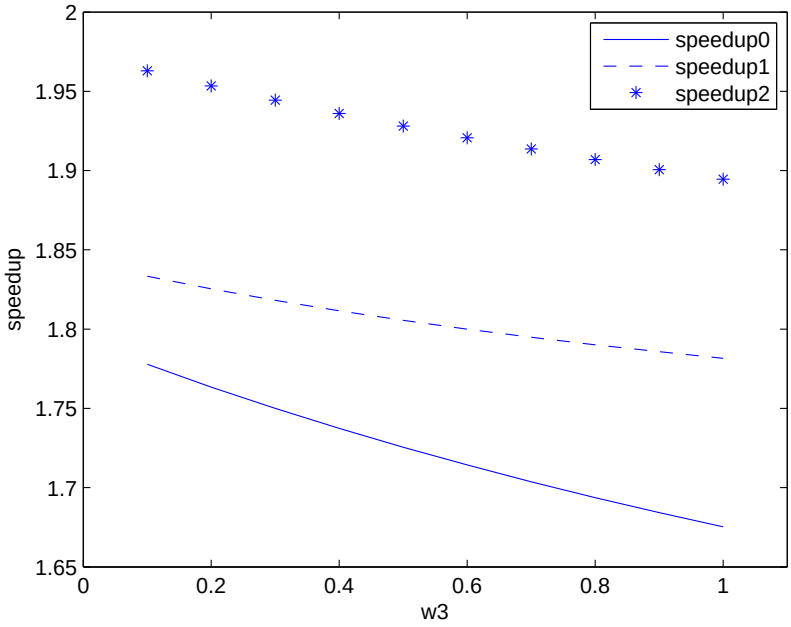

Figure 4.18: Speedups comparison with different link speed but same computation speed for all processors as processor 3 in case 4.2.3

From figure 4.18 we can see that the second strategy is the best.

4.3 Load distribution without front-end

Here we discuss the load distribution using the same network as in Figure 4.1 without front-end, i.e the processors in the network can not communicate and compute at the same time.

4.3.1 The link speeds are all the same and computation speed of Processor 3 is different

We plot the timing diagram in figure 4.19 and the related equations are shown below.

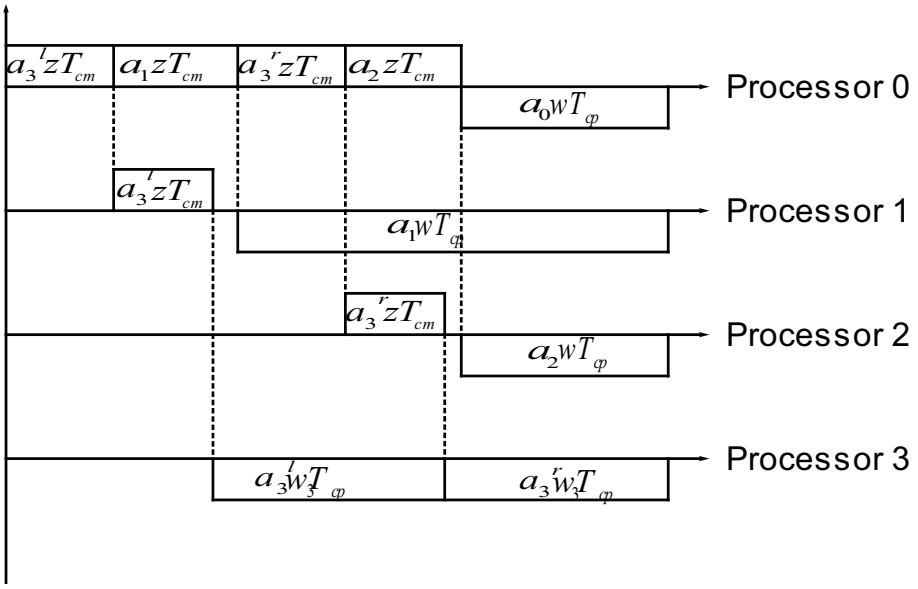

Figure 4.19: Timing diagram 1 in case 4.3.1

$$\alpha_1 wT_{cp} = \alpha_3{}^r zT_{cm} + \alpha_2 zT_{cm} + \alpha_0 wT_{cp} \tag{4.77}$$

$$\alpha_2 z T_{cm} + \alpha_0 w T_{cp} = \alpha_3{}^r z T_{cm} + \alpha_3{}^r w_3 T_{cp} \tag{4.78}$$

$$\alpha_0 w T_{cp} = \alpha_2 w T_{cp} \tag{4.79}$$

$$\alpha_3{}^l z T_{cm} + \alpha_3{}^l w_3 T_{cp} + \alpha_3{}^r w_3 T_{cp} = \alpha_1 z T_{cm} + \alpha_1 w T_{cp} \tag{4.80}$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3{}^l + \alpha_3{}^r = 1 \tag{4.81}$$

From figure 4.19 we realize that $\alpha_3{}^r < \alpha_2$ and from equation (4.78) we obtain $w_3 > w$. Next we discuss the case that $w_3 < w$, in which case the timing diagrams are showed in figure 4.20 and figure 4.21.

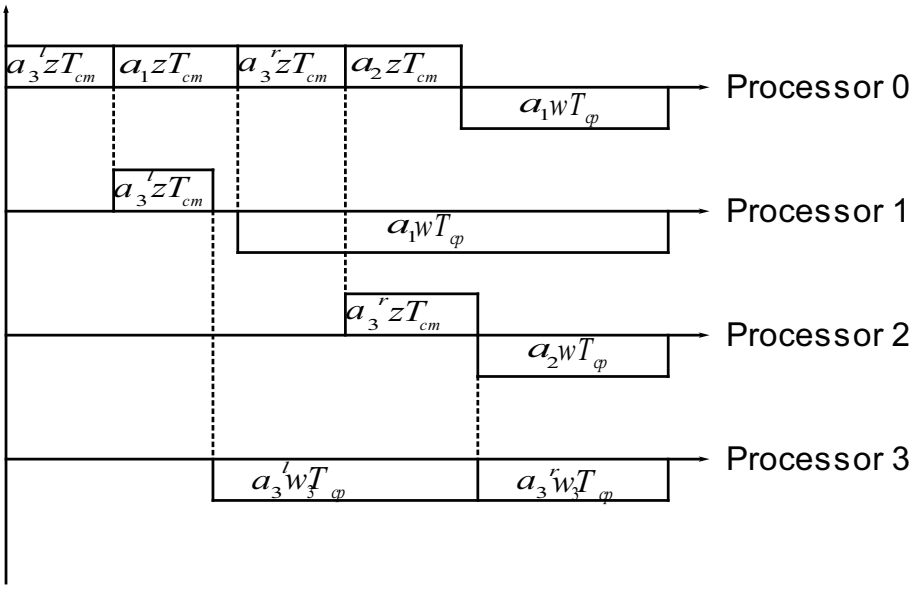

Figure 4.20: Timing diagram 2 in case 4.3.1

We find the difference between these two timing diagrams is whether the finish time of the communication time from processor 1 to processor 3 is earlier than the finish time of the communication time from processor 0 to

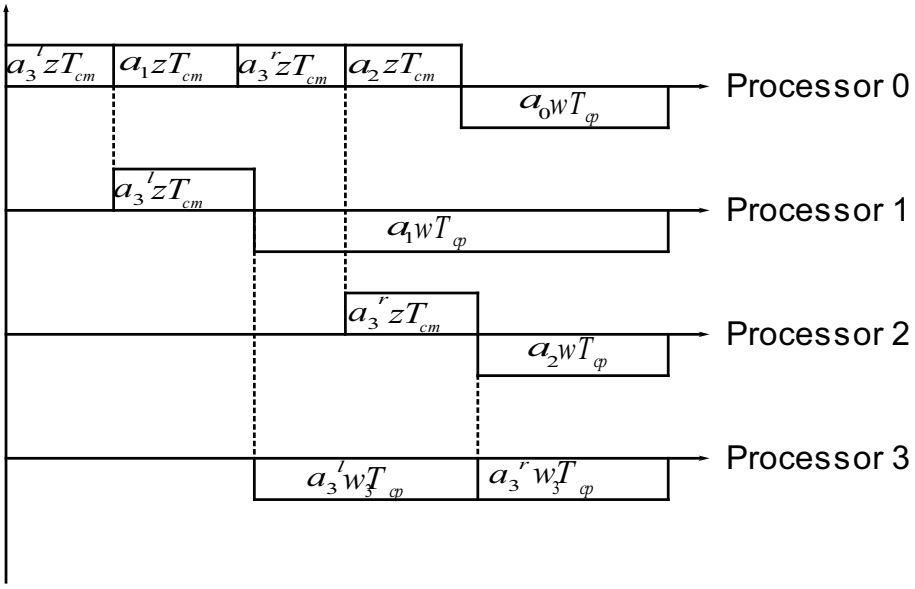

Figure 4.21: Timing diagram 3 in case 4.3.1

processor 1. We want to find the exact value of w_3 for these two different cases. We know the transition value for w_3 is when the two communication processes complete at the same time, that is

$$\alpha_1 z T_{cm} = {\alpha_3}^l z T_{cm} \tag{4.82}$$

$$2{\alpha_3}^r z T_{cm} = {\alpha_3}^l w_3 T_{cp} \tag{4.83}$$

$$\alpha_1 w T_{cp} = ({\alpha_3}^l + {\alpha_3}^r) w_3 T_{cp} \tag{4.84}$$

From equation (4.82)(4.83)(4.84) we obtain ${w_3}^2 + 2\lambda w w_3 - 2\lambda w^2 = 0$, so $w_3 = (\sqrt{\lambda^2 + 2\lambda} - \lambda)w$. We define this transition value as w_{3X}. When $w_{3X} < w_3 < w$, we choose the strategy whose timing diagram shown in figure 4.20, and when $w_3 < w_{3X}$, we choose the strategy whose timing diagram

shown in figure 4.21.

Compared with the reference network in which there is no right link for Processor 3 we plot out the speedup in figure 4.22.

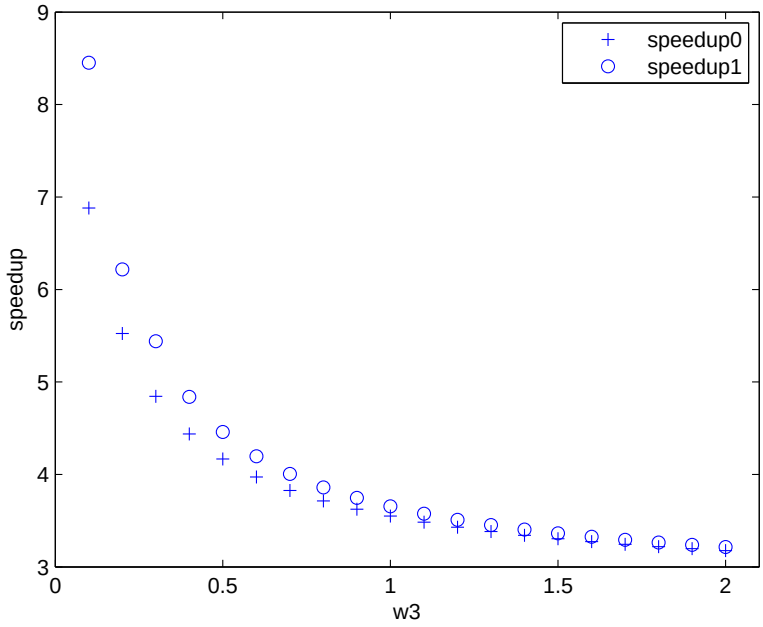

Figure 4.22: speedup when load distribution without frond-end

4.4 Conclusion

In this paper optimal schedules for distributing and processing divisible load for a novel type of cyclic topology have been presented. A wide variety of schedules have been examined.

Future work includes determining to the extent possible the best scheduling strategies over a broad range of network parameters. Naturally it would

also be a logical step to examine larger networks. To make at least initial progress in this regard assumptions of parameter homogeneity and regular network structure could be made. Beyond this one should move onto heterogeneous parameters for irregular topologies.

In the divisible load scheduling literature almost all interconnection networks to date with cycles in their graphs have used spanning trees embedded in the interconnection networks to distribute load. Have multiple distribution paths in such networks offers the possibility of improved throughput, smaller make spans and more robust distribution of load. Thus this may very well be an important research problem in the future.

Chapter 5

Optimal trade-off between monetary cost and solution time

5.1 introduction

Over the past several decades, the emergence of distributed computing as a viable technology and the decreasing pricing of computer power leads to the emergence of computer "utilities". These utilities charge customers for distributed access to computer resources. Such utilities have long been forseen by researchers. To some extent, current cloud computing embodies this approach. An important question for the utility then becomes the management of computer resources to provide low cost service. In this spirit, this paper provides an approach to determine the minimum cost manner in which load

should be divided among processors that a customer is being charged for access to.

A divisible load is a load (usually data) that is arbitrarily partitionable among a network of processors. The divisible load model usually assumes that the processing time of divisible load on a single processor and the transmission time of divisible load from one processor to another are both proportional to the divisible load size [45][6][23]. In general the processing time of a unit divisible load on a standard processor and the transmission time of a unit divisible load through a link with standard bandwidth are denoted as T_{cp} and T_{cm}, respectively. The aim of divisible load scheduling is to minimize the processing time by distributing divisible load among multiple processors which are interconnected by a specific network topology. It is particularly suited to the processing of very long linear data files, such as occur in signal and image processing, data bases, bioinformatics, experimental data processing, Kalman filtering and big data.

Analytically based optimal load sharing for divisible loads was first considered for linear daisy chain networks in [2]. Related results later appeared for tree networks [53], bus networks [51], [3], hypercubes [7], and two-dimensional meshes [40]. There was also work on asymptotic results [56], [45], [47], closed form solutions [54], time-varying models [57], multiple job submission [58], load distribution sequences [46], [55], [9], [48], [5], the modeling of fixed communication delay [59], the real time systems [60], [61], [62]. A book-length treatment of divisible load analysis appears in [63].

The work in [64] suggests a dynamic load balancing strategy to optimize

the trade-off between cost and solution time. Moreover, it compares the trade-off boundary with different processors as the master cloud distributing processor and plots out the optimum computing cost boundary under different solution time ranges. This paper demonstrates its correctness mathematically and furthermore presents the final equation of the trade-off boundary.

There is a fairly large literature on economic models for computer and telecommunication networks particularly from the 1990s. A representative sample appears in [65], [66], [67], [68], [69], [70], [71], [72]. Now [73] [74][75][76][77] examined the divisible load based minimal total computing cost scheme and discuss the case for further reducing the total computing cost with some degradation in processing finish time. However, it is generally assumed that the load should be distributed so that all of the processors complete their portions simultaneously, which involve some break-points for the trade-off in the Cost-Time result figure in [64]. Moreover, the order for Pareto-optimal solutions with relatively large deadlines is correct only for a specified master processor. Optimal schedules for tight deadlines have a different order of processors with a specified master processor. This paper will first examine the minimal total dynamic computing cost scheme for a specific master processor, then discuss the case for different master processors.

The paper is organized as follows: The definition of the load sharing problem for the determination of the optimal load allocation found in earlier work, and existing load sharing theory for the minimal processing finish time as a function of the speed of the load origination processor are presented in Section 5.2. The minimal computing cost strategies under a certain so-

lution time threshold by heuristic algorithm is described in section 5.3 and furthermore the trade-off is discussed for different master processors in Section 5.3.1. Section 5.3.2 compares the trade-off transition with neighboring master processors and gives the ultimate cost and finish time trade-off with minimum time comparison. Finally the scheduling improvement is presented in section 5.4 and the conclusion appears in section 5.5.

5.2 Model Description

The network model to be considered here consists of N processors interconnected through a bus type communication medium, as Fig 5.1. Any one of N processors can receive a new arriving load and distribute this workload to the other processors in order to obtain the benefits of parallel processing. Without loss of generality, it will be assumed that the load is delivered to one of the processors and this processor becomes the load origination processor (master processor). Each processor is interfaced with the network via a front-end communication processor for communications off-loading. That is, the processors can communicate and compute at the same time [23]. The following notation will be used throughout this paper:

α_n: The fraction of the entire processing load that is assigned to the n_{th} processor P_n.

w_n: The inverse of the computing speed of P_n.

z: The inverse of the channel speed of the bus.

c_n: Computing cost per seconds.

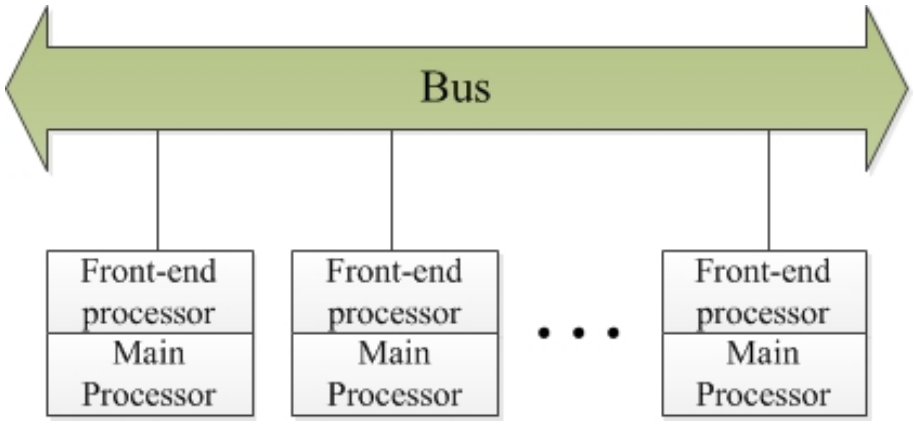

Figure 5.1: Distributed computing system consisting of N processors equipped with front-end processors connected through a bus

$c_n w_n$: Computing cost per load.

T_{cp}: The size of the normalized computational load in time, i.e., the time that it takes for P_n to process (compute) the entire load when $w_n = 1$.

T_{cm}: The size of the normalized communication load in time, i.e., the time that it takes to transmit the entire set of data over the bus when $z = 1$.

T_n: The time for P_n to complete receiving the corresponding fraction α_n of load from the load origination processor P_1.

T_f: The finish time of the entire processing load, assuming that the load is completely delivered to the origination processor at time zero.

The timing diagram for this distributed system is depicted in Fig 5.2. In this timing diagram, communication time appears above the axis and computation time appears below the axis. At time $T_1 = 0$, the load origination processor P_1 keeps the first fraction of the workload α_1 for its own computation, which will take a time of T_f to finish, and simultaneously transmits

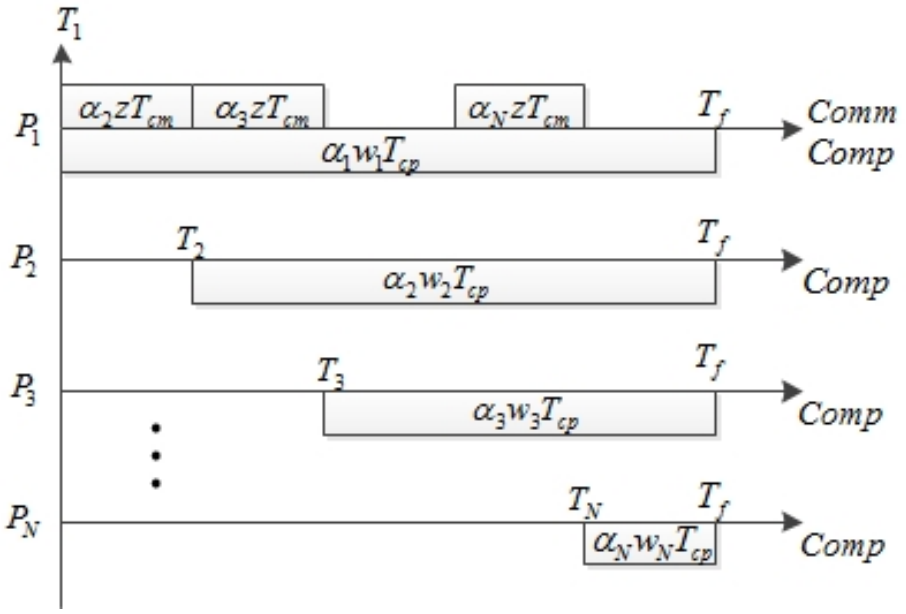

Figure 5.2: Timing diagram of N bus interconnected processors with load origination at P1

the second fraction of the workload to P_2 in time $T_2 - T_1$. Note that, as P_1 has a front-end processor for communications off-loading, it may both compute and communicate at the same time. When the transmission of the second fraction of the workload is finished at time T_2, P_2 starts computing the received workload and P_1 begins transmission of the third fraction of the workload of P_3 in time $T_3 - T_2$. This procedure continues until the last processor. For (finish time) optimality, all the processors must finish computing at the same time. Intuitively, this is because, otherwise, the processing finish time could be reduced by transferring load from busy processors to idle ones. [63]

Based on the above description, one can construct the following $N - 1$

equations by equating the computation time of P_n with the transmission time plus the computation time of P_{n+1}.

$$T_f - T_n = (T_{n+1} - T_n) + (T_f - T_{n+1}) \tag{5.1}$$

Here, $n = 1, 2, \ldots, N-1, n \neq m$. The computation time of P_n and the transmission time of p_{n+1} are:

$$T_f - T_n = \alpha_n w_n T_{cp} \tag{5.2}$$

$$T_{n+1} - T_n = \alpha_{n+1} z T_{cm} \tag{5.3}$$

Then, (5.1) can be rewritten using (5.2) and (5.3) as:

$$\alpha_n w_n T_{cp} = \alpha_{n+1} z T_{cm} + \alpha_{n+1} w_{n+1} T_{cp} \tag{5.4}$$

These equations can be solved

$$\alpha_{n+1} = k_n \alpha_n = \left(\prod_{i=1}^{n} \right) \alpha_1 \tag{5.5}$$

where

$$k_n = \frac{\alpha_{n+1}}{\alpha_n} = \frac{w_n T_{cp}}{z T_{cm} + w_{n+1} T_{cp}} \tag{5.6}$$

There are $N-1$ equations and N unknowns α_n, $n = 1, 2, \ldots, N$. An additional equation is called a normalization equation, which states that the sum

of all the allocation fractions should sum to one.

$$\sum_{n=1}^{N} \alpha_n = 1 \tag{5.7}$$

By combining (5.5) and (5.7), one can find the optimal fraction of the workload that minimizes the total processing finish time. The closed-form expressions are:

$$k_n = \frac{w_n T_{cp}}{z T_{cm} + w_{n+1} T_{cp}} \tag{5.8}$$

$$\alpha_1 = \left[\sum_{n=1}^{N} \prod_{j=1}^{n-1} k_j \right]^{-1} \tag{5.9}$$

$$\alpha_n = k_{n-1} \alpha_{n-1} \tag{5.10}$$

Finally, the processing finish time T_f is:

$$\begin{aligned} T_f &= \alpha_1 w_1 T_{cp} \\ &= \frac{w_1 T_{cp}}{1 + k_1 + k_1 k_2 + \ldots + (k_1 k_2 \ldots k_{N-1})} \\ &= \frac{w_1 T_{cp} \prod_{n=2}^{N} (z T_{cm} + w_n T_{cp})}{\sum_{n=1}^{N} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \prod_{i=n+1}^{N} (z T_{cm} + w_i T_{cp}) \right]} \end{aligned} \tag{5.11}$$

The denominator of T_f is independent of switching any w_i with any other $w_j (i, j = 1, 2, \ldots, N, i \neq j)$. Only the numerator is dependent on switching w_1 with any other w_k $(k = 2, 3, \ldots, N)$ and is minimized when w_1 is chosen to be the smallest w_i. That is, no matter whether the load origination processor transmits the workload to the next fastest processor first or to the slowest

processor first, the processing finish time remains the same. The processing finish time depends only on the speed of the load origination processor and is minimized when P_1 is chosen to be the fastest processor among all the processors in the distributed computing system. Note that, in this paper, we do not consider delivering the load in installments to each processor as in [9].

5.3 Minimizing the total computing cost

We assume that the cost of processors $c_n w_n$ can vary in any way in relation to w_n. If the computing "cost" for the processors are not identically valued, the total computing cost for the entire workload varies and depends on how much of the workload is processed in each processor. Intuitively, if the cheaper processors are more utilized than the more expensive processors, then the total computing cost will be reduced. If the total workload is distributed on the cheapest processor, the total computing cost will be cheapest, but on the other hand the finish time may be the longest. In order to minimize the finish time, therefore, the workload should be distributed to more processors, which makes up the distributed computer system. However, this will involve transferring some of the workload from the cheap processors to expensive processors, the result is: the total computing cost will increase. Thus, this leads to a special issue regarding the number of processors and the selection of processors to trade off computing cost against finish time. The arrangement for the sequence of the processors should be considered.

Let us denote the set $\Theta_{(1,2,\ldots,N)}$ as an ordered set of N processors. The

set $\Theta_{(1,2,...,N)}$ determines the sequence of load distribution. For instance, for the set $\Theta_{(2,1,3)}$, P_2 is the origination processor and P_1 is the processor which receives the workload from P_2 first, and P_3 receives the workload from P_2 second.

The notation c_n will be used for the computing cost of P_n whose unit is "cost per second". The unit of the inverse computing speed of the n_{th} processor, w_n, is "second per load" since w_n is defined as the inverse of the computing speed. Recall that T_{cp} is the size of the normalized computational load in time (see the previous section). Then, the unit of $c_n w_n$ becomes the "cost per load". Now $\alpha_n c_n w_n T_{cp}$ represents the computing "cost" of P_n for the fraction of workload received from the load origination processor. Let us denote the notation C_{total} for the total computing cost for the entire workload, suppose M $(M \leq N)$ processor participate in the distribution of workload and the expression for this is:

$$C_{total} = \sum_{n=1}^{M} \alpha_n c_n w_n T_{cp} \tag{5.12}$$

The optimal selection and sequence of processor $\Theta_{(1,2,...,N)}$ which balances the trade-off between computing cost and finish time should be a dynamic distribution problem between diverse processors. Unlike the traditional distribution strategy only involved in [2] [53] [45], [54] [57], all the processors need not finish their work-load at the same time, which will mandatorily take some part of the load from the cheap processor to the expensive processors, as a result the total computing cost will increase while the finish time can be

set below the finish time requirement flexibly.

To simplify this problem, we just consider the transition from one processor to two processor. To make the cost cheapest if only one processor would be used, the cheapest processor should be chosen. Suppose the first processor is the cheapest processor, P_1, the corresponding inverse computing speed is w_1 and computing cost per second is c_1w_1, $c_1w_1 \leq c_{other}w_{other}$. Note that the physical position between processors is not in consideration. The finish time w_1T_{cp} will definitely be longer than if the second processor participates in the distribution load process. Assume $\Delta\alpha_1$ from P_1 is distributed to P_2, the additional part for P_2 is defined as $\Delta\alpha_2$. Assume that the plus-minus notation stand for that work load been cut or added, thus $\Delta\alpha_1 < 0$ and $\Delta\alpha_2 > 0$. The work-load distributed from P_1 is totally transmitted to P_2, thus $\Delta\alpha_1 = -\Delta\alpha_2$. The finish time change would only depend on the finish time for P_1, otherwise more load is distributed on the expensive processor. Thus, the finish time change defined as ΔT_f is $\Delta\alpha_1 w_1 T_{cp}$, and the computing cost change defined as $\Delta C^{(2)}$ is $\Delta\alpha_1 c_1 w_1 T_{cp} + \Delta\alpha_2 c_2 w_2 T_{cp}$. Combining ΔT_f and $\Delta C^{(2)}$:

$$\begin{aligned} \Delta C^{(2)} &= \frac{\Delta T_f}{w_1 T_{cp}} c_1 w_1 T_{cp} - \frac{\Delta T_f}{w_1 T_{cp}} c_2 w_2 T_{cp} \\ &= \Delta T_f \left(\frac{c_1 w_1 - c_2 w_2}{w_1} \right) \end{aligned} \tag{5.13}$$

To maintain the total cost in the least expensive state, the second processor should be chosen as the second cheapest processor from the remaining processors, that is $c_1w_1 \leq c_2w_2 \leq c_{other}w_{other}$, then the gradient of $\frac{cost}{T_f}$ defined

as $\Delta\frac{C^{(2)}}{T_f} = \frac{1}{w_1}(c_1w_1 - c_2w_2)$ will be less than 0, but larger than other gradients if we choose other more expensive processors for the second processor. This maintains the least expensive optimal state in the load redistribution process.

This transfer continues until the state that both processors finish their work-load at the same time, for which the finish time is the shortest while the total computing cost is highest. Obviously P_1 should not continue to distributed more load to P_2 because the finish time would increase and the total computing cost would also increase.

Recursively, as more and more processors participate in the distribution computer system, the finish time will decrease and the computing cost will increase. The optimal sequence of processors $\Theta_{(1,2,\ldots,M)}$ and distribution are described in the following theorem.

LEMMA 1. The work-load of the new additional processor described as α_i is distributed from the previous processors $P_1, P_2, \ldots, P_{i-1}$, and the computing cost per second for the previous processors $c_1w_1, c_2w_2 \ldots, c_{i-1}w_{i-1}$ are all lower than the new added processor c_iw_i. Moreover, in the transition of distributing work-load from the previous processors to the new additional processor, the previous processors always finish their own work-load at the same time.

PROOF. Suppose that the previous processors do not finish their own work-load at the same time, we can redistribute the work-load from expensive processors to cheap processors until the cheap processors finish their own work-load at the finish time threshold, which is the solution time for

the cheapest processor (T_f for P_1). Thus the finish time remain the same meanwhile the total computing cost will decrease. Repeating this process, at last all the processors will finish computation at the same time except the new processor.

Assume P_i is the specific new processor. The other processors distribute some part of load to P_i, let us say, P_1 distribute its load cut part $\Delta\alpha_1$ work-load to P_i, P_2 distribute its load cut part $\Delta\alpha_2$ work-load to P_i, ..., P_{i-1} distribute its load cut part $\Delta\alpha_{i-1}$ work-load to P_i. From the timing diagram Fig 5.3, the computation time for the cut part of P_1 is the sum of the computation and communication time for the cut part of P_2; the computation time for the cut part of P_2 is the sum of the computation and communication time for the cut part of P_3; recursively, the computation time for the cut part of P_{i-2} is the sum of the computation and communication time for the cut part of P_{i-1}.

$$\Delta\alpha_j w_j T_{cp} = \Delta\alpha_{j+1}(w_{j+1}T_{cp} + zT_{cm}) \tag{5.14}$$

Here $j \leq i-2$. There are $i-2$ equations and $i-1$ unknowns. An additional equation which state the total work-load of cut parts should be remain constant, that means:

$$\Delta\alpha_1 + \Delta\alpha_2 + \ldots + \Delta\alpha_{i-1} + \Delta\alpha_i = 0 \tag{5.15}$$

So far all the cut part of different processors can be described as a function of

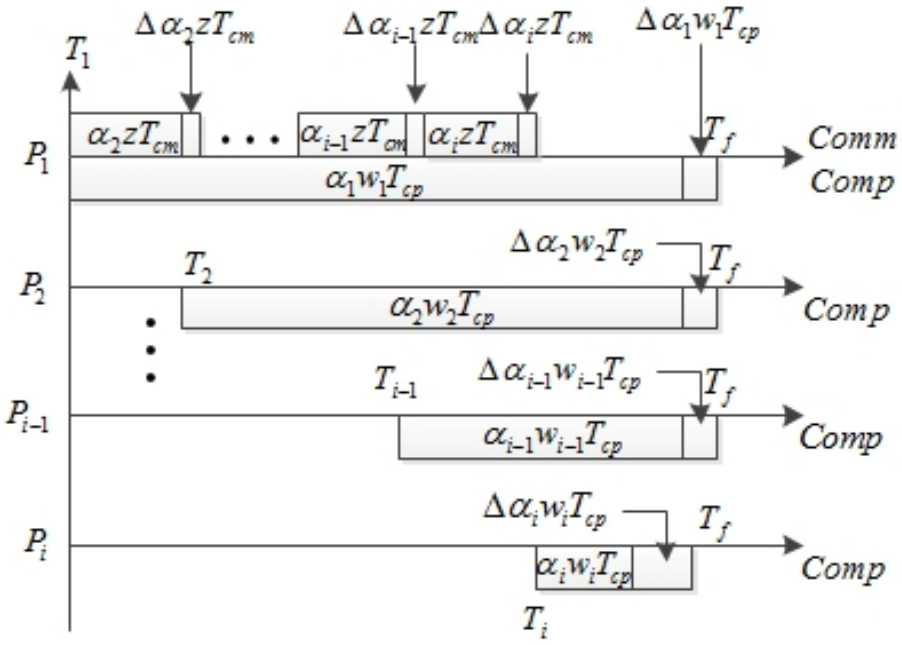

Figure 5.3: Timing diagram of i bus interconnected processors with load origination at P1 when i-1 processors redistribute load to the ith processor

$\Delta\alpha_1$. The finish time change only depends on P_1, that is $\Delta T_f = \Delta\alpha_1 w_1 T_{cp}$. However, the total computing cost change depends on all the processors.

$$\begin{aligned}\Delta C^{(i)} &= \sum_{n=1}^{i} \Delta\alpha_n c_n w_n T_{cp} \\ &= \Delta\alpha_1 \sum_{n=1}^{i-1} (\prod_{j=1}^{n-1} k_j \cdot (c_n w_n T_{cp} - c_i w_i T_{cp}))\end{aligned} \tag{5.16}$$

k_j here has been defined in equation (5.8). To maintain the least computing cost in the distribution process, the gap of the computing cost between two different solution times should be as small as possible. That means the gradient between computing cost and finish time should be smallest. The gradient defined as $S^{(i)}$ is :

$$S^{(i)} = \frac{\Delta C^{(i)}}{\Delta T_f} = \frac{1}{w_1} \sum_{n=1}^{i-1} (\prod_{j=1}^{n-1} k_j \cdot (c_n w_n - c_i w_i)) \tag{5.17}$$

Thus the gradient is related to the computation speed of master processor (here P_1 is the master processor), the computation speed of processor $P_{i-1}, P_{i-2}, \ldots, P_2$ and the monetary computing cost per load for processor $P_i, P_{i-1}, \ldots, P_1$. Notice that the selection of P_i is considered and $P_{i-1}, P_{i-2}, \ldots, P_1$ have been chosen out. Thus $S^{(i)}$ is only depend on the computing cost per load of P_i. Moreover, the more expensive computing cost P_i is, the larger gap of computing cost between two different solution times. Thus to maintain the least expensive state here, the cheapest processor should be selected from the idle processors. Also we explore the gradient

for $S^{(i)}$ here to analyse this issue:

$$\begin{aligned}
\Delta S^{(i)} &= S^{(i+1)} - S^{(i)} \\
&= \frac{1}{w_1}[\sum_{n=1}^{i}(\prod_{j=1}^{n-1} k_j \cdot (c_n w_n - c_{i+1} w_{i+1})) \\
&- \sum_{n=1}^{i-1}(\prod_{j=1}^{n-1} k_j \cdot (c_n w_n - c_i w_i))] \\
&= \frac{1}{w_1}[\sum_{n=1}^{i-1}(\prod_{j=1}^{n-1} k_j \cdot (c_n w_n - c_{i+1} w_{i+1} - c_n w_n + c_i w_i)) \\
&+ (\prod_{j=1}^{i-1} k_j \cdot (c_i w_i - c_{i+1} w_{i+1}))] \\
&= \frac{1}{w_1}\sum_{n=1}^{i}(\prod_{j=1}^{n-1} k_j \cdot (c_i w_i - c_{i+1} w_{i+1}))
\end{aligned} \tag{5.18}$$

The gradient for $S^{(i)}$ stands for the gap of the state transition trade-off between computing cost and solution time. From equation (5.18) we know that it only depends on the two processor most recently selected. To minimize this gradient, the least expensive processor should be selected out from the left idle processors. This is the same with the analysis above. Thus to make the total computing cost as low as possible, we should choose the processor as this arrangement: $c_1 w_1 \leq c_2 w_2 \leq \ldots \leq c_i w_i$. The transition for total N processors between computing cost and finish time is plot in Fig 5.4 as below. Here N is 5, and the processor speed and monetary cost per load is depicted in table 5.1.

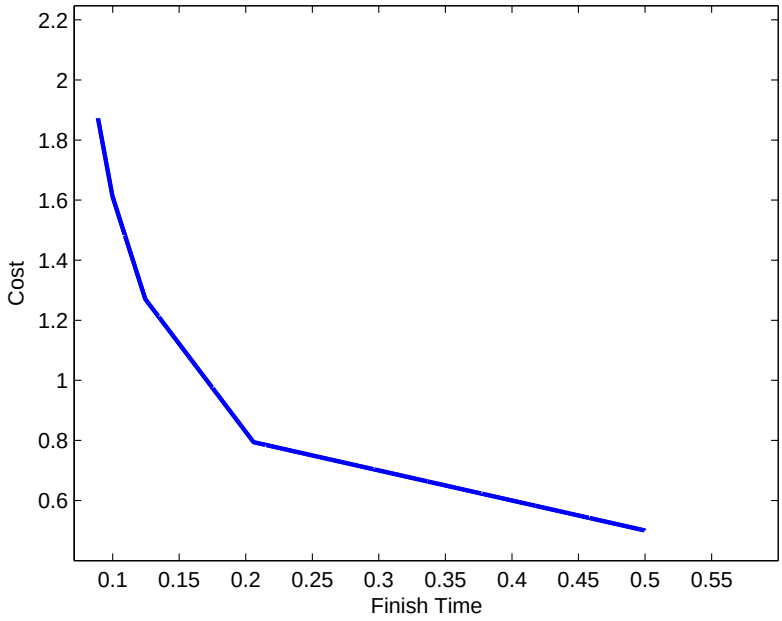

Figure 5.4: Optimal computing cost and finish time for five unique processors and the first processor as the master processor

5.3.1 Different master processors

The optimal sequence discussed so far only guarantees that the total computing cost $C_{total} = \sum_{n=1}^{i} c_n w_n T_{cp}$ is at the lowest level when the master

	w_n	c_n
No.	inverse computation speed	monetary cost per load
1	0.5	1
2	0.25	4
3	0.125	16
4	0.1	30
5	0.0625	64

Table 5.1: list of processors

processor is chosen to be the cheapest processor. However, from section two, given a certain list of processors, the minimal finish time depends on the computation speed of the master processor. Thus the cheapest processor may be not the fastest processor, actually this would not happen practically. Besides, for the processors choosing a strategy described above, the finish time also can not be smaller compared with that where we always choose the fastest processor from the remaining processors to guarantee the finish time will be minimal in the whole transition process. Thus the trade-off between computing cost and finish time also should be considered.

Just like the analysis above, assume P_m is the chosen master processor $(1 < m \leq N)$, to maintain the minimal cost in the transition, the sequence discussed above should be adopted. However, the finish time here is composed of the computation and communication time of P_1, that is:

$$T_f = \alpha_1(zT_{cm} + w_1T_{cp}) \tag{5.19}$$

Also we use the gradient between the cost difference and the finish time difference to analyze this problem. When the master processor does not participate in the computation process, which is different with the finish time change above, here P_1 receives the work load from P_m, thus $\Delta T_f = \Delta\alpha_m w_m T_{cp} = \Delta\alpha_1(w_1T_{cp} + zT_{cm})$. With the same computing cost change, we obtain:

$$\frac{\Delta C^{(i)}}{\Delta T_f} = \frac{1}{z\delta + w_1}\sum_{n=1}^{i-1}(\prod_{j=1}^{n-1} k_j \cdot (c_n w_n - c_i w_i)) \tag{5.20}$$

Noted that $\delta = \frac{T_{cm}}{T_{cp}}$ and $i < m$.

When P_m participates in the computation process, the gradient should be the same as before where P_m finishes its own part work-load at the same time with the other $m-1$ processors. The only difference is that the master processor is different from the others for it can compute from the beginning. Thus it can receive more load compared with other processors whose sum of the computation and communication time is equal to the computation time of prior processor. When P_{m+1} participate in the computation process, the sum of the computation and communication time is not equal to the computation time of P_m, but P_{m-1}. Thus $\Delta\alpha_{m-1}w_{m-1}T_{cp} = \Delta\alpha_{m+1}(w_{m+1}T_{cp} + zT_{cm})$. Combined with all the factors, the final equations between the computing cost and finish time is shown as below, here assuming that $M_i^j = \sum_{n=i}^{j}\prod_{u=1}^{n-1}k_u$ and $\Delta C_i^j = c_i w_i - c_j w_j$.

$$Cost = (T_f - T_f^{(i)}) \cdot \frac{\Delta C^{(i)}}{\Delta T_f} + cost^{(i)}, i = 2, 3, \ldots, N \qquad (5.21)$$

$$T_f^{(i)} \geq T_f \geq T_f^{(i+1)}$$

This equation is true whether or nor P_m participate in the computation process or not. The solution time $T_f^{(i)}$, the gradient between the computing cost and solution time $S^{(i)}$, the total computing cost at the moment that all the participated processor finish their work load at the same time $cost^{(i)}$, and

the gradient for $S^{(i)}$ which is $\Delta S^{(i)}$ are described below.

$$T_f^{(i)} = \begin{cases} \frac{zT_{cm}+w_1T_{cp}}{M_1^{i-1}} & i < m+1 \\ \frac{zT_{cm}+w_1T_{cp}}{M_1^{m-1}+\frac{z\delta+w_1}{w_m}} & i = m+1 \\ \frac{zT_{cm}+w_1T_{cp}}{M_1^{m-1}+\frac{z\delta+w_1}{w_m}+M_{m+1}^{i-1}\frac{z\delta+w_m}{w_m}} & i > m+1 \end{cases} \tag{5.22}$$

$$S^{(i)} = \frac{\Delta C^{(i)}}{\Delta T_f} \tag{5.23}$$

$$= \begin{cases} \frac{1}{w_1+z\delta}M_1^i\Delta C_n^i & i < m+1 \\ \frac{1}{w_1+z\delta}\left[M_1^{m-1}\Delta C_n^{m+1} + \frac{z\delta+w_1}{w_m}\Delta C_m^{m+1}\right] & i = m+1 \\ \frac{1}{w_1+z\delta}[M_1^{m-1}\Delta C_n^i + \frac{z\delta+w_1}{w_m}\Delta C_m^i + \\ \frac{z\delta+w_m}{w_m}M_{m+1}^{i-1}\Delta C_n^i] & i > m+1 \end{cases}$$

$$cost^{(i)} = \begin{cases} \frac{M_1^{i-1}c_nw_nT_{cp}}{M_1^{i-1}} & i < m+1 \\ \frac{M_1^{m-1}c_nw_nT_{cp}+\frac{z\delta+w_1}{w_m}c_mw_mT_{cp}}{M_1^{m-1}+\frac{z\delta+w_1}{w_m}} & i = m+1 \\ \frac{M_1^{m-1}c_nw_nT_{cp}+\frac{z\delta+w_1}{w_m}c_mw_mT_{cp}}{M_1^{m-1}+\frac{z\delta+w_1}{w_m}+M_{m+1}^{i-1}\frac{z\delta+w_m}{w_m}} + \\ \frac{\frac{z\delta+w_m}{w_m}M_{m+1}^{i-1}c_nw_nT_{cp}}{M_1^{m-1}+\frac{z\delta+w_1}{w_m}+M_{m+1}^{i-1}\frac{z\delta+w_m}{w_m}} & i > m+1 \end{cases} \tag{5.24}$$

$$\Delta S^{(i)} = S^{(i+1)} - S^{(i)} \tag{5.25}$$

$$= \begin{cases} \frac{1}{w_1+z\delta} M_1^i \Delta C_i^{i+1} & i < m \\ \frac{1}{w_1+z\delta}(M_1^{m-1} + \frac{z\delta+w_1}{w_m})\Delta C_m^{m+1} & i = m \\ \frac{1}{w_1+z\delta}(M_1^{m-1} + \frac{z\delta+w_1}{w_m} + \frac{z\delta+w_m}{w_m} M_{m+1}^i \Delta_i^{i+1}) & i > m \end{cases}$$

Here each quantity is defined in term of the i_{th} processor and its relation to the master's location. As is done by Shakhlevich in [64] we plot out the corresponding gradient figure in Fig 5.5. Here five processors are in our selection. We assume that $c_1w_1 < c_2w_2 < c_3w_3 < c_4w_4 < c_5w_5$, $w_1 > w_2 > w_4 > w_3 > w_5$. The specification for these five processors are shown in table 5.2. From Fig 5.5 we know that to obtain the lowest computing cost the optimal selection for master processor depend on the finish time threshold. However, compared with the two lines with P_3 and P_4 as master processor, the line for P_4 is not optimal for the entire finish time range. That means we would not consider P_4 at all given a certain finish time threshold. We will demonstrate this in Section 5.3.2. The ultimate distribution strategy is the

	w_n	c_n
No.	inverse computation speed	monetary cost per load
1	0.5	1
2	0.25	4
3	0.1	15
4	0.125	16
5	0.0625	64

Table 5.2: list of processors

mixed line composed of different lines with different line as the master line,

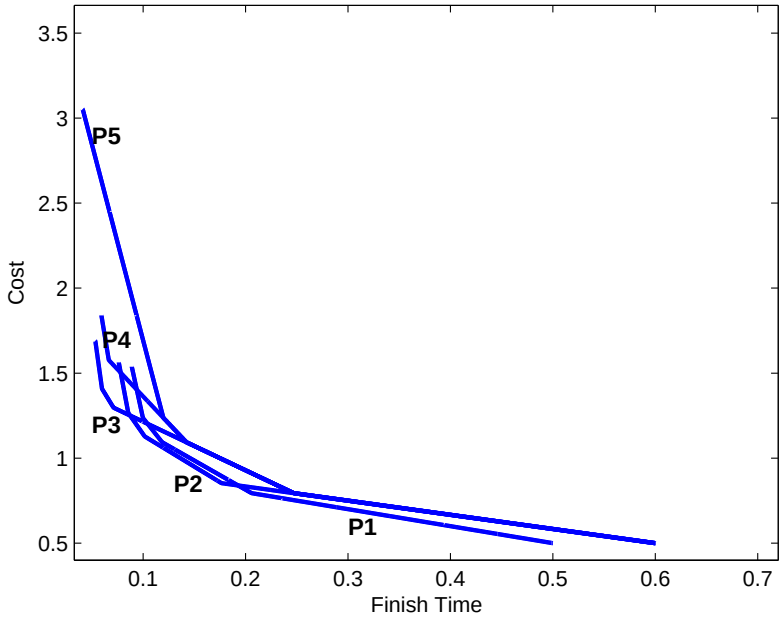

Figure 5.5: Optimal computing cost and finish time for five unique processors with multiple master processors

picking out the minimum computing cost for certain finish time. Compared with the curve with P_1 as the master processor, the new method described here has larger range in the finish time and less computing cost.

5.3.2 Boundary with different master processors

In this section we try to find out the relationship of finish time, computing cost and the gradient when the master processor is arbitrarily either m or $m+1$. We discuss the i_{th} part of these two cases to find out the difference using the same methodology in section 5.3.

When $i < m$, P_m and P_{m+1} both do not participate in the computation process. They just transfer their initial load to other processors to work. With the same communication speed, the master difference has no effect on the total computing cost and solution time. Equation (5.22), (5.23), (5.24), (5.25) also show that the finish time, computing cost and gradient are all the same.

When $i = m$, The solution time $T_f^{(m+1)}$ when the master processor is m_{th} processor denoted as $T_{f_m}^{(m+1)}$ is $\frac{zT_{cm}+w_1T_{cp}}{\sum_{n=1}^{m-1}\prod_{j=1}^{n-1}k_j+\frac{z\delta+w_1}{w_m}}$ and the finish time when the master processor is $(m+1)_{th}$ processor is $T_{f_{m+1}}^{(m+1)} = \frac{zT_{cm}+w_1T_{cp}}{\sum_{n=1}^{m+1-1}\prod_{j=1}^{n-1}k_j} = \frac{zT_{cm}+w_1T_{cp}}{\sum_{n=1}^{m-1}\prod_{j=1}^{n-1}k_j+\prod_{j=1}^{m-1}k_j}$. Comparing the denominator of the two equations, we

obtain:

$$\begin{aligned}\prod_{j=1}^{m-1} k_j &= \frac{w_1}{z\delta + w_2}\frac{w_2}{z\delta + w_3}\cdots\frac{w_{m-1}}{z\delta + w_m} \\ &= \frac{w_1}{z\delta + w_m}\frac{w_2}{z\delta + w_2}\frac{w_3}{z\delta + w_3}\cdots\frac{w_{m-1}}{z\delta + w_{m-1}} \\ &< \frac{w_1}{z\delta + w_m} < \frac{z\delta + w_1}{w_m}\end{aligned} \tag{5.26}$$

Thus $T_{f_m}^{(m+1)} < T_{f_{m+1}}^{(m+1)}$. From equation (5.24) $cost_m^m = cost_{m+1}^m$ and also from equation (5.23) the gradient for both are the same.

When $i = m + 1$, the gradient between computing cost and solution time when the master processor is m_{th} is

$$\begin{aligned}(\frac{\Delta C^{m+1}}{\Delta T_f})_m = \frac{1}{w_1 + z\delta}[&\sum_{n=1}^{m-1}\prod_{j=1}^{n-1} k_j(c_n w_n - c_{m+1}w_{m+1}) \\ &+ \frac{z\delta + w_1}{w_m}(c_m w_m - c_{m+1}w_{m+1})]\end{aligned} \tag{5.27}$$

The gradient between computing cost and solution time when the master

processor is $(m+1)_{th}$ is

$$
\begin{aligned}
&(\frac{\Delta C^{m+1}}{\Delta T_f})_{m+1} = \\
&\frac{1}{w_1+z\delta}\left[\sum_{n=1}^{m+1-1}\prod_{j=1}^{n-1}k_j(c_nw_n-c_{m+1}w_{m+1})\right] \\
&=\frac{1}{w_1+z\delta}[\sum_{n=1}^{m-1}\prod_{j=1}^{n-1}k_j(c_nw_n-c_{m+1}w_{m+1}) \\
&+\prod_{j=1}^{m-1}k_j(c_mw_m-c_{m+1}w_{m+1})] \\
&>(\frac{\Delta C^{m+1}}{\Delta T_f})_m
\end{aligned}
\tag{5.28}
$$

Also $cost_m^{(m+1)} > cost_{m+1}^{m+1}$ can be easily demonstrated. For the finish time, it can be demonstrated that when $w_m < w_{m+1}$, $T_{f_m}^{(m+2)} < T_{f_{m+1}}^{(m+2)}$, otherwise if $\prod_{j=1}^{m} k_j < \frac{z\delta+w_1}{z\delta w_{m+1}}(w_m - w_{m+1})$ then $T_{f_m}^{(m+2)} > T_{f_{m+1}}^{(m+2)}$. This constraint is not so simple and perhaps it is better not to check this instead of comparing the computing costs with different master processors directly. More specifically, the finish time, gradient between computing cost and finish time, computing cost when all the participated processors finish their load at the same time, and the transition of the gradient between computing cost and finish time are shown below. These factors are significant when we obtain the trade-off between total cost and solution time.

For example in the Table if $i = m+1$, both T_f and $S^{(i)}$ are greater when the master processor is $m+1$ compared to m. From this table it can be concluded that when $w_m < w_{m+1}$, $T_{f_m}^{(i)}$ will always be less than $T_{f_{m+1}}^{(i)}$. The

gradient between computing cost and finish time when master processor is m_{th} processor is less than that when master processor is $(m+1)_{th}$ if $w_m < w_{m+1}$ for i_{th} part ($i <= m+2$), when $i > m+2$, also this is true which can be proved from the fact that the transition of gradient when the master processor is m_{th} processor, is less than that when the master processor is $(m+1)_{th}$ processor if $w_m < w_{m+1}$. Thus if $w_m < w_{m+1}$, the gradient between computing cost and finish time when the master processor is m_{th} processor will be always less than that when the master processor is $(m+1)_{th}$ processor. Thus the computing cost when the master processor is the m_{th} processor will be always less than that when the master processor is the $(m+1)_{th}$ processor when the finish time is the same. Thus we can conclude that we will definitely NOT choose the trade-off cost and solution time boudary for the master processor as $(m+1)_{th}$ under any finish time threshold if $w_m < w_{m+1}$. On the

	T_f	$S^{(i)}$
$i < m$	same	same
$i = m$	same	same
$i = m+1$	$p_{m+1} > p_m$	$p_{m+1} > p_m$
$i = m+2$	$p_{m+1} > p_m$	$p_{m+1} > p_m$
$i = m+3$	$p_{m+1} > p_m$	$p_{m+1} > p_m$
$i > m+3$	$p_{m+1} > p_m$	$p_{m+1} > p_m$
	cost	ΔS
$i < m$	same	same
$i = m$	same	$p_{m+1} > p_m$
$i = m+1$	$p_{m+1} < p_m$	$p_{m+1} > p_m$
$i = m+2$	$p_{m+1} > p_m$	$p_{m+1} > p_m$
$i = m+3$	do not know	$p_{m+1} > p_m$
$i > m+3$	do not know	$p_{m+1} > p_m$

Table 5.3: Comparison when $w_m < w_{m+1}$ for p_m and p_{m+1}

other hand, if $w_m > w_{m+1}$, the comparison would be complex if we try to obtain the constraint that differentiate the finish time, computing cost and gradient under strict constraint condition. We would prefer to compare the computing cost with different master processors under same solution time. Thus we choose the min function to choose out the smaller computing cost under the same finish time. The combination equation should be:

$$cost \tag{5.29}$$
$$= \begin{cases} cost_{m+1} & T^N_{f_{m+1}} \leq T_f < T^N_{f_m} \\ min(cost_m, cost_{m+1}) & T^N_{f_m} \leq T_f < T^m_{f_{m+1}} \\ cost_m & T^{m+1}_{f_m} \leq T_f \leq T^2_{f_m} \end{cases}$$

Thus in the above we evaluate the cost for different line segments in the tradeoff curve (i.e. combine two segments). Here $T^2_{f_m} = zT_{cm} + w_1 T_{cp}$ and $min(cost_i, cost_j)$ means select the minimum value from the two lines whose x coordinate is finish time and y coordinate is computing cost, of the master processor is i or j. Thus we can conclude recursively that the total cost

function in equation (5.29), assumed that $w_1 > w_2 > \ldots > w_N$

$$Cost = \tag{5.30}$$

$$\begin{cases} cost_N & T_{f_N}^N \leq T_f < T_{f_{N-1}}^N \\ min(cost_N, cost_{N-1}) & T_{f_{N-1}}^N \leq T_f < T_{f_{N-2}}^N \\ \ldots & \\ min(cost_N, cost_{N-1}, \ldots, cost_i) & T_{f_i}^N \leq T_f < T_{f_{i-1}}^N \\ \ldots & \\ min(cost_N, cost_{N-1}, \ldots, cost_3) & T_{f_3}^N \leq T_f < T_{f_2}^N \\ min(cost_{N-1}, cost_{N-2}, \ldots, cost_2) & T_{f_2}^N \leq T_f < T_{f_{N-2}}^{N-1} \\ \ldots & \\ min(cost_i, cost_{i-1}, cost_2) & T_{f_i}^{i+1} \leq T_f < T_{f_{i-1}}^i \\ \ldots & \\ min(cost_3, cost_2) & T_{f_3}^4 \leq T_f < T_{f_2}^3 \\ cost_2 & T_{f_2}^3 \leq T_f < T_{f_1}^2 \end{cases}$$

Equation (5.30) involves all segments where (5.29) involves only two segments.

5.4 Scheduling Improvement

In this section the improvement of the strategy described above and the distribution strategy described in [73] is compared. Five processors, whose specifications show in table 5.1, are chosen to plot out the trade-off between

computing cost and solution time in fig 5.6 and the comparison between the optimal trade-off and the previous distribution strategy is shown in fig 5.7.

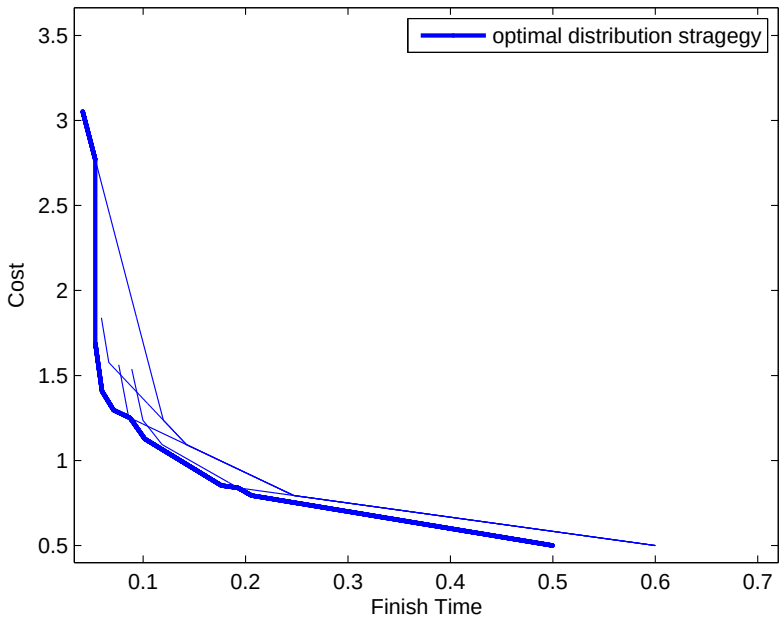

Figure 5.6: Optimal computing cost and finish time for five unique processors with multiple master processors

From figure 5.7 it is obvious that the distribution strategy is significantly better than the previous strategy. Not only it using a dynamic programming algorithm here, which can obtain a better specific state for processors under a specific solution time threshold, but also it spans the width of the solution time for different master processors.

Hence, given a set of processors, to get the minimal computing cost under a certain finish time threshold we should do this step by step. First, we should sort the computing cost per load for these processors. Second, using

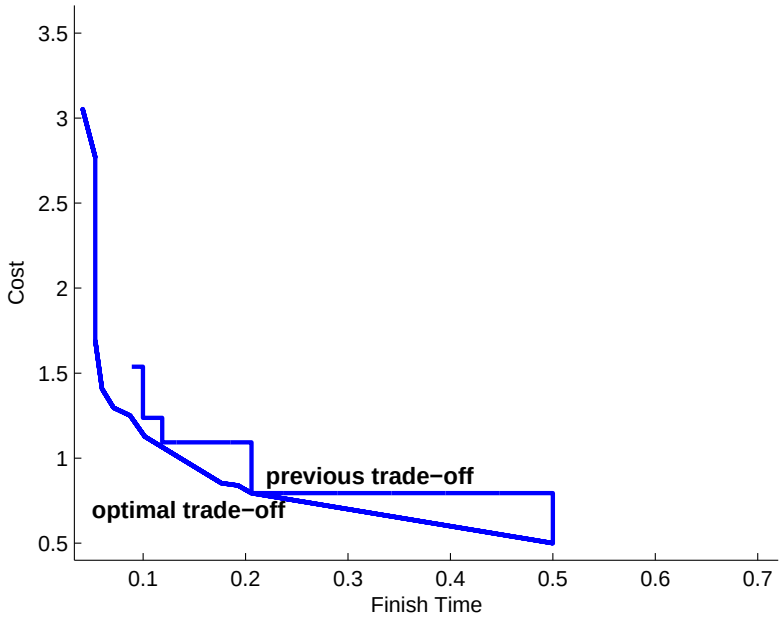

Figure 5.7: Comparison between the optimal distribution strategy and previous strategy

equation (5.30) to calculate the number of processors which participate in the computation process and obtain the optimal computing cost with different master processors. Third, determine the best master processor and calculate the load distribution for every processor.

5.5 Conclusion

In this work we have found:

• Confirmation: We confirm the work of Shaklevich that the trade-off between monetary cost and solution time (i.e. makespan) leads to a piecewise linear dual optimization criteria boundary function.

• Mathematical Model: Divisible load scheduling modeling with monetary cost aspects allows an examination of optimality for this problem in an analytical framework.

• Gradient Technique: Gradient techniques provide a tractable means for conducting this analysis.

• Computer Utility Applicability: This work shows that computer utility-like services have a foundation in mathematical optimization that is flexible and realistic.

This is an important problem area that involves a joint optimization of monetary cost and processing time.

Chapter 6

Conclusion

In this thesis, the speedup for searching for signature in different types of networks are studied. Chapter 2 introduces signature problem for multi-level tree. Different cases including number of signature known or unknown, each file having at most one signature or multiple signatures can exist in one file, only one node commanding all the nodes, every root node of each local tree commanding the nodes below it, all the files in every level are searched in parallel and levels are searched sequentially, all the files in the tree are searched in parallel, are discussed. These cases are compared and the speedup figures are plotted out. In chapter 3, we extend signature searching problem in two kinds of mesh network and hypercube network. Compared with the searching time in multi-level tree network, we obtained the optimal searching time under different cases. In chapter 4, a new kind of network: cyclic network is discussed. Load distributed with or without front-end are analyzed. Using different load distribution strategy, optimal solution time

is investigated. Homogeneous computation speeds and the link speed for processor 3 have a significant effect on the load distribution. In chapter 4, a trade-off between computing cost and solution time for an oriented bus network. The optimal sequence of processors is derived, the trade-off line for one master processor is plotted out and the ultimate mixed trade-off boundary is investigated with multiple master processors.

Bibliography

[1] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, 2003.

[2] Y. Cheng and T. Robertazzi, "Distributed computation with communication delay," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 700–712, 1988.

[3] S. Bataineh and T. G. Robertazzi, "Bus-oriented load sharing for a network of sensor driven processors," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 5, pp. 1202–1205, 1991.

[4] G. D. Barlas, "Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 5, pp. 429–441, 1998.

[5] H. J. Kim, G.-I. Jee, and J. G. Lee, "Optimal load distribution for tree network processors," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 32, no. 2, pp. 607–612, 1996.

[6] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang, "Scheduling divisible loads on star and tree networks: results and open problems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 3, pp. 207–218, 2005.

[7] J. Błazewicz and M. Drozdowski, "Scheduling divisible jobs on hypercubes," *Parallel computing*, vol. 21, no. 12, pp. 1945–1956, 1995.

[8] W. Głazek, "A multistage load distribution strategy for three-dimensional meshes," *Cluster Computing*, vol. 6, no. 1, pp. 31–39, 2003.

[9] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 31, no. 2, pp. 555–567, 1995.

[10] Y. Yang and H. Casanova, "Umr: A multi-round algorithm for scheduling divisible workloads," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International.* IEEE, 2003, pp. 9–pp.

[11] O. Beaumont, A. Legrand, and Y. Robert, "Scheduling divisible workloads on heterogeneous platforms," *Parallel Computing*, vol. 29, no. 9, pp. 1121–1152, 2003.

[12] V. Bharadwaj and G. Barlas, "Scheduling divisible loads with processor release times and finite size buffer capacity constraints in bus networks," *Cluster Computing*, vol. 6, no. 1, pp. 63–74, 2003.

[13] J.-T. Hung, H.-J. Kim, and T. Robertazzi, "Scalable scheduling in parallel processors," 2003.

[14] A. Piriyakumar and C. S. R. Murthy, "Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, no. 2, pp. 245–251, 1998.

[15] H.-J. Kim, "A novel optimal load distribution algorithm for divisible loads," *Cluster Computing*, vol. 6, no. 1, pp. 41–46, 2003.

[16] M. Adler, Y. Gong, and A. L. Rosenberg, "Optimal sharing of bags of tasks in heterogeneous clusters," in *SPAA*, vol. 3, 2003, pp. 1–10.

[17] P. Dutot, "Divisible load on heterogeneous linear array," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS03), Nice, France*, 2003.

[18] M. Drozdowski and P. Wolniewicz, "Optimum divisible load scheduling on heterogeneous stars with limited memory," *European Journal of Operational Research*, vol. 172, no. 2, pp. 545–559, 2006.

[19] M. Drozdowski, M. Lawenda, and F. Guinand, "Scheduling multiple divisible loads," *International Journal of High Performance Computing Applications*, vol. 20, no. 1, pp. 19–30, 2006.

[20] M. Drozdowski and M. Lawenda, "The combinatorics in divisible load scheduling," *Foundations of Computing and Decision Sciences*, vol. 30, pp. 297–308, 2005.

[21] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Bandwidth-centric allocation of independent tasks on heterogeneous platforms," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM.* IEEE, 2001, pp. 6–pp.

[22] B. Veeravalli and N. Viswanadham, "Suboptimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 30, no. 6, pp. 680–691, 2000.

[23] V. Bharadwaj, H. Li, and T. Radhakrishnan, "Scheduling divisible loads in bus networks with arbitrary processor release times," *Computers & Mathematics with Applications*, vol. 32, no. 7, pp. 57–77, 1996.

[24] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Computing*, vol. 6, no. 1, pp. 7–17, 2003.

[25] M. Drozdowski and P. Wolniewicz, "Experiments with scheduling divisible tasks in clusters of workstations," in *Euro-Par 2000 Parallel Processing.* Springer, 2000, pp. 311–319.

[26] K. Ko and T. Robertazzi, "Signature search time evaluation in flat file databases," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 44, no. 2, pp. 493–502, 2008.

[27] Y. Kyong and T. G. Robertazzi, "Greedy signature processing with arbitrary location distributions: A divisible load," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, 2012.

[28] R. Baeza-Yates, "Algorithms for string searching," in *AcM sIGIR Forum*, vol. 23, no. 3-4. ACM, 1989, pp. 34–58.

[29] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.

[30] P. Michailidis and K. Margaritis, "String matching algorithms: Survey and experimental results," in *International Journal of Computer Mathematics*, vol. 76, 2000, pp. 411–434.

[31] Z. Galil and R. Giancarlo, "Data structures and algorithms for approximate string matching," *Journal of Complexity*, vol. 4, no. 1, pp. 33–72, 1988.

[32] H. Kitakami, T. Shin-I, K. Ikeo, Y. Ugawa, N. Saitou, T. Gojobori, and Y. Tateno, "Yamato and asuka: Dna database management system," in *System Sciences, 1995. Vol. V. Proceedings of the Twenty-Eighth Hawaii International Conference on*, vol. 5. IEEE, 1995, pp. 72–80.

[33] M. Hoffman and D. Carver, "Reverse engineering data requirements," in *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, vol. 2. IEEE, 1996, pp. 269–277.

[34] M. Lubeck, D. Geppert, and K. Nienartowicz, "An overview of a large-scale data migration," in *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on.* IEEE, 2003, pp. 49–55.

[35] T. Robertazzi, *Networks and grids: technology and theory.* Springer Publishing Company, Incorporated, 2010.

[36] M. Drozdowski, *Scheduling for Parallel Processing.* Springer-Verlag New York Inc, 2009.

[37] H. Casanova, A. Legrand, and Y. Robert, *Parallel algorithms.* CRC Press, 2009.

[38] M. Drozdowski and W. Glazek, "Scheduling divisible loads in a three-dimensional mesh of processors," *Parallel Computing*, vol. 25, no. 4, pp. 381–404, 1999.

[39] K. Li, "Speed-up of parallel processing of divisible loads on k-dimensional meshes and tori," *The Computer Journal*, vol. 46, no. 6, pp. 625–631, 2003.

[40] J. Błażewicz and M. Drozdowski, "The performance limits of a two dimensional network of load-sharing processors," *Foundations of Computing and Decision Sciences*, vol. 21, no. 1, pp. 3–15, 1996.

[41] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 1, 2006.

[42] E. Salminen, T. Kangas, V. Lahtinen, J. Riihimäki, K. Kuusilinna, and T. Hämäläinen, "Benchmarking mesh and hierarchical bus networks in system-on-chip context," *Journal of Systems Architecture*, vol. 53, no. 8, pp. 477–488, 2007.

[43] S. Stuijk, T. Basten, M. Geilen, A. Ghamarian, and B. Theelen, "Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip," *Journal of Systems Architecture*, vol. 54, no. 3-4, pp. 411–426, 2008.

[44] R. Agrawal and H. Jagadish, "Partitioning techniques for large-grained parallelism," *Computers, IEEE Transactions on*, vol. 37, no. 12, pp. 1627–1634, 1988.

[45] T. G. Robertazzi, "Processor equivalence for daisy chain load sharing processors," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 29, no. 4, pp. 1216–1221, 1993.

[46] V. Mani and D. Ghose, "Distributed computation in linear networks: Closed-form solutions," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 30, no. 2, pp. 471–483, 1994.

[47] D. Ghose and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," *Journal of Parallel and Distributed Computing*, vol. 23, no. 3, pp. 293–305, 1994.

[48] V. Bharadwaj, D. Ghose, and V. Mani, "An efficient load distribution strategy for a distributed linear network of processors with communication delays," *Computers & Mathematics With Applications*, vol. 29, no. 9, pp. 95–112, 1995.

[49] J. Sohn and T. Robertazzi, "Optimal load sharing for a divisible job on a bus network," in *Proceedings of the 1993 Conference on Information Sciences and Systems*. Citeseer, 1993, pp. 835–840.

[50] S. Bataineh and M. Al-Ibrahim, "Effect of fault tolerance and communication delay on response time in a multiprocessor system with a bus topology," *Computer Communications*, vol. 17, no. 12, pp. 843–851, 1994.

[51] D. Ghose and H. J. Kim, "Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays," *Journal of Parallel and Distributed Computing*, vol. 55, no. 1, pp. 32–59, 1998.

[52] J. EWICZ and M. DROZDOWSKI, "The performance limits of a two-dimensional network of load-sharing processors."

[53] Y.-C. Cheng and T. Robertazzi, "Distributed computation for a tree network with communication delays," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 511–516, 1990.

[54] S. Bataineh, T.-Y. Hsiung, and T. G. Robertazzi, "Closed form solutions for bus and tree networks of processors load sharing a divisible job," *Computers, IEEE Transactions on*, vol. 43, no. 10, pp. 1184–1196, 1994.

[55] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal sequencing and arrangement in distributed single-level tree networks with communication delays," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 9, pp. 968–976, 1994.

[56] S. Bataineh and T. G. Robertazzi, "Performance limits for processor networks with divisible jobs," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 33, no. 4, pp. 1189–1198, 1997.

[57] J. Sohn and T. Robertazzi, "An optimum load sharing strategy for divisible jobs with time-varying processor speed and channel speed," in *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems*, 1995, pp. 27–32.

[58] J. Sohn, T. G. Robertazzi, J. Sohn, T. G. Robertazzi *et al.*, "A multi-job load sharing strategy for divisible jobs on bus networks," in *State Univ. of New York at Stony Brook, College of Eng.* Citeseer, 1994.

[59] J. Błażewicz and M. Drozdowski, "Distributed processing of divisible jobs with communication startup costs," *Discrete Applied Mathematics*, vol. 76, no. 1, pp. 21–41, 1997.

[60] E. Haddad, "Communication protocol for optimal redistribution of divisible load in distributed real-time systems," in *Proc. ISMM Intl Conf. Intelligent Information Management Systems*, 1994, pp. 39–42.

[61] X. Lin, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling for cluster computing," in *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE.* IEEE, 2007, pp. 303–314.

[62] A. Mamat, Y. Lu, J. Deogun, and S. Goddard, "Real-time divisible load scheduling with advance reservation," in *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on.* IEEE, 2008, pp. 37–46.

[63] V. Bharadwaj, *Scheduling divisible loads in parallel and distributed systems.* John Wiley & Sons, 1996, vol. 8.

[64] N. V. Shakhlevich, "Scheduling divisible loads to optimize the computation time and cost," in *Economics of Grids, Clouds, Systems, and Services.* Springer, 2013, pp. 138–148.

[65] H. A. Gil, F. D. Galiana, and E. L. da Silva, "Nodal price control: a mechanism for transmission network cost allocation," *Power Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 3–10, 2006.

[66] S. Kaneda, T. Uyematsu, N. Nagatsu, and K.-i. Sato, "Network design and cost optimization for label switched multilayer photonic ip networks," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 8, pp. 1612–1619, 2005.

[67] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in computer networks: Motivation, formulation, and example," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 6, pp. 614–627, 1993.

[68] A. Farago, S. Blaabjerg, L. Ast, G. Gordos, and T. Henk, "A new degree of freedom in atm network dimensioning: optimizing the logical configuration," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 7, pp. 1199–1206, 1995.

[69] J. F. Kurose and R. Simha, "A microeconomic approach to optimal resource allocation in distributed computer systems," *Computers, IEEE Transactions on*, vol. 38, no. 5, pp. 705–717, 1989.

[70] S. H. Low and P. P. Varaiya, "A new approach to service provisioning in atm networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 5, pp. 547–553, 1993.

[71] Y. A. Korilis, A. A. Lazar, and A. Orda, "Architecting noncooperative networks," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 7, pp. 1241–1251, 1995.

[72] D. Menascé and V. Almeida, "Cost-performance analysis of heterogeneity in supercomputer architectures," in *Proceedings of the 1990*

ACM/IEEE conference on Supercomputing. IEEE Computer Society Press, 1990, pp. 169–177.

[73] J. Sohn, T. G. Robertazzi, and S. Luryi, "Optimizing computing costs using divisible load analysis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 3, pp. 225–234, 1998.

[74] S. Charcranoon, T. G. Robertazzi, and S. Luryi, "Parallel processor configuration design with processing/transmission costs," *Computers, IEEE Transactions on*, vol. 49, no. 9, pp. 987–991, 2000.

[75] F. D. Galiana, A. J. Conejo, and H. A. Gil, "Transmission network cost allocation based on equivalent bilateral exchanges," *Power Systems, IEEE Transactions on*, vol. 18, no. 4, pp. 1425–1431, 2003.

[76] F. J. Rubio-Odériz and I. J. Perez-Arriaga, "Marginal pricing of transmission services: A comparative analysis of network cost allocation methods," *Power Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 448–454, 2000.

[77] Y. Xi and E. M. Yeh, "Distributed algorithms for minimum cost multicast with network coding in wireless networks," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*. IEEE, 2006, pp. 1–9.