UNIVERSITY AT STONY BROOK

CEAS Technical Report 719

Optimizing Computing Costs Using Divisible Load Analysis

J. Sohn, T.G. Robertazzi and S. Luryi

Oct. 30, 1995

Optimizing Computing Costs Using Divisible Load Analysis

Jeeho Sohn

Thomas G. Robertazzi *Senior Member IEEE*

Serge Luryi *Fellow IEEE*

Dept. of Electrical Engineering,

University at Stony Brook,

Stony Brook, N.Y. 11794

Tel: (516) 632-8412/8400

Fax: (516) 632-8494

Abstract

In this paper a load sharing problem involving the optimal load allocation of a divisible load and the optimal sequencing of processors in a distributed computing system consisting of N processors, with communication front-ends, interconnected through a bus oriented network is investigated. For a ***divisible load*** the workload is infinitely divisible so that the workload can be distributed arbitrarily and independently computed on each processor. For the first time in divisible load theory, the computing cost to process a given load is considered along with processing finish time. It is found that an optimal load allocation involves assigning load to processors in increasing order of the cost per load of each processor. Recursive numerical algorithms to find the optimal fractions of the workload assigned to each processor for the minimal computing cost are derived. A trade-off between the optimal processing finish time and the optimal total computing cost when a faster processor has a more expensive computing cost is demonstrated. As an example of the use of this type of analysis, the effect of replacing one fast but expensive processor with a number of cheap but slow processors is also discussed.

Index Terms: Divisible Load, Load Sharing, Cost, Bus Network.

1 Introduction

The emergence of distributed computing as a viable 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 then 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.

Optimal and very efficient load sharing and allocation is essential for achieving minimal processing times. There are many possible ways to classify the load sharing problem. One of them is the classification by the type of job submitted to the system. This leads to *indivisible load* theory and *divisible load* theory. An *indivisible load* (or job) is a load that cannot be divided into more than one fragment so that the load must be processed by one processor. There has been intensive work on indivisible load theory by many parallel and distributed system researchers [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. Only recently has there been interest in multiprocessor scheduling with loads that need to be assigned to more than one processor [14, 15, 16]. A *divisible load* (or job) is a load that can be arbitrarily partitioned in a linear fashion and can be distributed to more than one processor to achieve a faster solution time. Applications include both multiprocessor scheduling and distributed systems. It is particularly suited to the processing of very long linear data files such as occurs in signal and image processing, experimental data processing and Kalman filtering.

The load sharing problem in divisible load theory is not trivial as one must take into account the number of processors, the speed of each processor and communication link, the load origination point and the network architecture. One important issue for the load sharing problem is a trade-off between the communication time and the computation time. This problem is less important when the size of data file to be transmitted is very small or the communication link speed is very fast so that the time to transmit a typical job can be negligible. However, one must consider the relationship between the communication time and the computation time to in order achieve the best performance in the load sharing and scheduling problem when the size of data file to be transmitted is very large or the communication link speed is very slow so that the time to transmit a typical job is not negligible. Even though there has been a great deal of work solely on communication and solely on computation, there has not been that much work which deals with both problems. This paper presents a theory for the optimal divisible load sharing problem which considers both non-negligible communication time and the computation time together.

The study of divisible load theory started from the consideration of intelligent sensor networks by Cheng and Robertazzi [17]. An *intelligent sensor* is a single processor based unit which can make measurements, compute and communicate with other intelligent sensors. Later this intelligent sensor network application was replaced by the application of load sharing in a multiprocessor environment. The main problem in this research is to determine the optimal fraction of the workload to assign to each processors. That is, when a network receives a burst of data to process, the decision of what portion of the entire workload should be kept by the distributing processor and what portion of the entire workload should be distributed to each processor in order to minimize the total processing time becomes an intriguing problem.

In [17], recursive expressions for calculating the optimal load allocation for linear daisy chains of processors were presented. This is based on the simplifying premise that for an optimal allocation of load, all processors must stop processing at the same time. Intuitively, this is because otherwise some processors would be idle while others were still busy. Analogous solutions have been developed for tree networks [18] and bus networks [19, 20]. Asymptotic solutions for systems with a large, or even an infinite number of processors, and limitations in performance when adding processors appear in [21, 28]. Closed form solutions were presented in [22] for bus and tree architectures where processor and link speeds are homogeneous. In [23], the concept of an equivalent processor that behaves identically to a collection of processors in the context of a linear daisy chain of processors and a proof that, for a linear daisy chain of processors load sharing a divisible load, the optimal solution involves all processors stopping at the same time are introduced. An analytic proof for bus networks that for a minimal solution time all processors must finish computing at the same time is shown in [24, 32]. Previous proofs were heuristic.

In [25], a more sophisticated load sharing strategy is proposed for bus networks that exploits the special structure of divisible load theory to yield a smaller solution time when a series of jobs are submitted to the network. In [26], a deterministic analysis is provided for the case when the processor speed and the channel speed are time-varying due to the background jobs submitted to a distributed system. A stochastic analysis which makes use of Markovian queueing theory is also introduced for the case when the arrival and the departure times of the background jobs are not known. The equivalence of first distributing load either to the left or to the right from a point in the interior of a linear daisy chain is demonstrated in [27]. Optimal sequences of load distribution in tree networks are described in [29, 33]. A new load distribution strategy for tree networks [30] and linear daisy chains [31] is also discussed. In [34], a deterministic approach to finding an optimal distribution of the load on a hypercube of processors is proposed. Simple formulae are found to determine the distribution of the task's load and the equivalent speed of the whole network of processors for a two-dimensional mesh architecture in [35]. A uniform methodology is presented in [36] to achieve minimum completion time for a wide range of interconnection architectures, assuming that the communication time is equal to some start-up value plus some amount proportional to the volume of transferred data. Finally, an example of optimal load allocation in a real time system appears in [37].

Until now there has been no research in divisible load theory on the effect of computing cost. Computing cost can be an important characteristic factor of processors that determines the pricing of the processing of a given load. The faster processor usually has the more expensive computing cost and the slower processor has the less expensive computing cost in practical situations. Thus, when each processor has a different computing cost, the total computing cost for a given workload depends on how much each processor is used and for how long. Naturally, in order to reduce the total computing cost, the cheaper processors should be used more than the more expensive processors. This leads an investigation of processor arrangement, or processor sequencing, in a distributed computing system. The minimal total computing cost can be achieved when the processors are optimally arranged. One simple and apparent method for minimizing the total computing cost is to use only the cheapest processor. But if only the cheapest processor is used, the processing finish time will be dramatically increased. So this paper will first examine the minimal total computing cost scheme which does not sacrifice the processing finish time, and then discuss the case for further reducing the total computing cost with some degradation in processing finish time.

This paper is organized as follows: Definitions, the load sharing problem for the determination of the optimal load allocation found in earlier works 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 2. The optimal sequence of processors yielding the minimal computing cost is discussed in section 3, and recursive numerical methods for further reducing the computing cost are proposed in section 4. Performance evaluation results appear in section 5. Finally, this paper concludes with section 6.

2 Preliminaries

The network model to be considered here consists of N processors interconnected through a bus type communication medium as in Fig. 1. Any one of N processors can receive a new arriving measurement 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 the first processor (P_1) and this processor becomes the load origination 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.

The following notations will be used throughout this paper:

α_n :	The fraction of the entire processing load that is assigned to the nth processor (P_n).
w_n :	The inverse of the computing speed of P_n.
Z :	The inverse of the channel speed of the bus.
T_{cp} :	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 normalized communication load in time, i.e., the time that it takes to transmit the entire set of data over the channel 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 delivered to the origination processor at time zero.

The timing diagram for this distributed system is depicted in Fig. 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 the second fraction of the workload (α_2) 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 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.

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}.

$$T_f - T_n = (T_{n+1} - T_n) + (T_f - T_{n+1}) \qquad n = 1, 2, \ldots, N-1 \tag{1}$$

Here, the computation time of P_n and the transmission time of P_{n+1} are:

$$\begin{aligned} T_f - T_n &= \alpha_n w_n T_{cp} & n &= 1, 2, \ldots, N & (2) \\ T_{n+1} - T_n &= \alpha_{n+1} Z T_{cm} & n &= 1, 2, \ldots, N-1 & (3) \end{aligned}$$

Then, Eq.(1) can be rewritten using Eq.(2) and Eq.(3) as:

$$\alpha_n w_n T_{cp} = \alpha_{n+1} Z T_{cm} + \alpha_{n+1} w_{n+1} T_{cp} \qquad n = 1, 2, \ldots, N-1 \tag{4}$$

These equations can be solved

$$\alpha_{n+1} = k_n \alpha_n = \left(\prod_{i=1}^{n} k_i \right) \alpha_1 \qquad n = 1, 2, \ldots, N-1 \tag{5}$$

where:

$$k_n = \frac{\alpha_{n+1}}{\alpha_n} = \frac{w_n T_{cp}}{ZT_{cm} + w_{n+1}T_{cp}} \qquad n = 1, 2, \ldots, N-1 \tag{6}$$

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.

$$\sum_{n=1}^{N} \alpha_n = 1 \tag{7}$$

By putting together Eq.(5) and Eq.(7), one can find the optimal fraction of the workload that minimizes the total processing finish time. The closed-form expressions are:

$$(1) \quad k_n = \frac{w_n T_{cp}}{ZT_{cm} + w_{n+1}T_{cp}} \qquad n = 1, 2, \ldots, N-1 \tag{8}$$

$$(2) \quad \alpha_1 = \left[1 + \sum_{n=2}^{N} \left(\prod_{i=1}^{n-1} k_i\right)\right]^{-1} \tag{9}$$

$$(3) \quad \alpha_n = k_{n-1} \cdot \alpha_{n-1} \qquad n = 2, 3, \ldots, N \tag{10}$$

Note that this solution is a product form solution. It is interesting that this deterministic model has such a solution as product form solutions are usually associated with stochastic queueing and Petri networks.

Finally, the processing finish time T_f is:

$$T_f = \alpha_1 w_1 T_{cp} \tag{11}$$

These equations also happen to model the case when a control ocessor, which does no computing of its own, distributes the load to the other processors. In this case, the order of load distribution does not affect the finish time. However, the network structure in this paper is different. Here the load origination processor does compute. In this case, the processing finish time, described in the above equations, can be further reduced by a special investigation of the relationship between the processing finish time and the speed of the load origination processor (w_1). It can be shown that the processing finish time can be reduced by carefully choosing the load origination processor.

Once a set of N processor speeds, $S(w) = \{w_1, w_2, \ldots, w_N\}$, is given, the processing finish time depends on the speed of the load origination processor (w_1) and is minimized [20] when w_1 is chosen to be the smallest (that is, the highest speed) in the set $S(w)$. The processing finish time is then [19, 20]:

$$T_f = \frac{w_1 T_{cp}(ZT_{cm} + w_2 T_{cp})(ZT_{cm} + w_3 T_{cp})}{(ZT_{cm})^2 + ZT_{cm}(w_1 + w_2 + w_3)T_{cp} + (w_1 w_2 + w_2 w_3 + w_3 w_1)T_{cp}^2} \tag{12}$$

For general N processors case, the finish time is given by:

$$T_f = \frac{w_1 T_{cp} \prod_{n=2}^{N} (ZT_{cm} + w_n T_{cp})}{\sum_{n=1}^{N} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \cdot \prod_{i=n+1}^{N} (ZT_{cm} + w_i T_{cp}) \right]} \tag{13}$$

The denominator of T_f is independent of switching any w_i with any other w_j ($i, j = 1, 2, \ldots, N$ and $i \neq j$). 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. □

Eq.(13) will be used in the next section. Note that, as shown in the Appendix, the processing finish time is independent of the sequence of the processors. That is, no matter whether the load origination processor transmits the workload to the 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 [30].

3 Minimizing the Total Computing Cost

If the computing cost for each processor is different, 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. In order to minimize the total computing cost, therefore, the cheaper processors should be more utilized. This leads to a special arrangement for the sequence of the processors (or the sequence of the load distribution).

Let us denote the set $\Theta_{(1,2,\ldots,N)}$ as an ordered set of N processors. The set $\Theta_{(1,2,\ldots,N)}$ determines the sequence of load distribution. For instance, for the set $\Theta_{(2,3,1)}$, P_2 is the load origination processor and P_3 is the processor which receives the workload from P_2 first, and P_1 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 *computing speed* of the nth processor, w_n, is "second per load" since w_n is defined as the inverse of the computing speed. Then, the unit $c_n w_n$ becomes the "cost per load" and $\alpha_n c_n w_n T_{cp}$ represents the computing "cost" of P_n for the received workload from the load origination processor. Let us denote the notation C_{total} for the total computing cost for the entire workload, and the expression for this is:

$$C_{total} = \sum_{n=1}^{N} \alpha_n c_n w_n T_{cp} \tag{14}$$

The optimal sequence of processors $\Theta_{(1,2,\ldots,N)}$ which minimizes Eq.(14) is described in the following theorem.

Theorem 1 *The total computing cost, C_{total}, is minimized over all load distribution sequences iff the sequence of the load distribution is arranged to satisfy the following condition.*

$$c_1 w_1 < c_2 w_2 < \cdots < c_N w_N \tag{15}$$

□

In other words, the processor with the lowest computing cost per load should be assigned to the load origination processor (P_1) and the processor with the second lowest computing cost per load should receive the workload earlier than any other processors, and the last portion of the load should be delivered to the most expensive processor.

Proof: Consider first the case where there are three processors ($N = 3$). The fractions of the workload are

$$\alpha_1 = [1 + k_1 + k_1 k_2]^{-1} = \frac{1}{D}(ZT_{cm} + w_2 T_{cp})(ZT_{cm} + w_3 T_{cp}) \tag{16}$$

$$\alpha_2 = k_1 \alpha_1 = \frac{1}{D} w_1 T_{cp}(ZT_{cm} + w_3 T_{cp}) \tag{17}$$

$$\alpha_3 = k_2 \alpha_2 = \frac{1}{D} w_1 T_{cp} \cdot w_2 T_{cp} \tag{18}$$

where:

$$D = (ZT_{cm})^2 + ZT_{cm}(w_1 + w_2 + w_3)T_{cp} + (w_1 w_2 + w_2 w_3 + w_3 w_1)T_{cp}^2 \tag{19}$$

Note that D is not changed by switching any set of processors. The total computing cost is:

$$\begin{aligned} C_{total} &= \sum_{n=1}^{3} \alpha_n c_n w_n T_{cp} \\ &= \frac{T_{cp}}{D}[(ZT_{cm} + w_2 T_{cp})(ZT_{cm} + w_3 T_{cp})c_1 w_1 + w_1 T_{cp}(ZT_{cm} + w_3 T_{cp})c_2 w_2 \\ &\quad + w_1 T_{cp} \cdot w_2 T_{cp} \cdot c_3 w_3] \end{aligned} \tag{20}$$

Now, let us first show that if $c_1 w_1 < c_2 w_2$, then the above total computing cost, $C_{total}(\Theta_{(1,2,3)})$, is smaller than the one when P_1 and P_2 are switched, $C_{total}(\Theta_{(2,1,3)})$. One can start with

$$c_1 w_1 < c_2 w_2$$

and get the following inequality by multiplying ZT_{cm} (> 0), adding $(w_2T_{cp} \cdot c_1w_1 + w_1T_{cp} \cdot c_2w_2)$, multiplying $(ZT_{cm}+w_3T_{cp})$, adding $(w_1T_{cp} \cdot w_2T_{cp} \cdot c_3w_3)$, and multiplying $\frac{T_{cp}}{D}$ on both sides of the above inequality.

$$\begin{aligned} & \tfrac{T_{cp}}{D} \;\; [(ZT_{cm} + w_2T_{cp})(ZT_{cm} + w_3T_{cp})c_1w_1 + w_1T_{cp}(ZT_{cm} + w_3T_{cp})c_2w_2 \\ & \qquad\qquad + w_1T_{cp} \cdot w_2T_{cp} \cdot c_3w_3] \\ < \; & \tfrac{T_{cp}}{D} \;\; [(ZT_{cm} + w_1T_{cp})(ZT_{cm} + w_3T_{cp})c_2w_2 + w_2T_{cp}(ZT_{cm} + w_3T_{cp})c_1w_1 \\ & \qquad\qquad + w_2T_{cp} \cdot w_1T_{cp} \cdot c_3w_3] \end{aligned}$$

The left hand side of the inequality is the same as $C_{total}(\Theta_{(1,2,3)})$ and the right hand side is C_{total} when P_1 and P_2 are switched, $C_{total}(\Theta_{(2,1,3)})$. Therefore, if $c_1w_1 < c_2w_2$, then $C_{total}(\Theta_{(1,2,3)}) < C_{total}(\Theta_{(2,1,3)})$.

The reverse order proof is as follows: It can be shown that if $C_{total}(\Theta_{(1,2,3)})$ is smaller than $C_{total}(\Theta_{(2,1,3)})$, then $c_1w_1 < c_2w_2$. To demonstrate this, first write $C_{total}(\Theta_{(1,2,3)})$ and $C_{total}(\Theta_{(2,1,3)})$ while assuming that $C_{total}(\Theta_{(1,2,3)})$ is less than $C_{total}(\Theta_{(2,1,3)})$.

$$\begin{aligned} & \{C_{total}(\Theta_{(1,2,3)}) \cdot \frac{D}{T_{cp}} = (ZT_{cm} + w_2T_{cp})(ZT_{cm} + w_3T_{cp})c_1w_1 \\ & + w_1T_{cp}(ZT_{cm} + w_3T_{cp})c_2w_2 + w_1T_{cp} \cdot w_2T_{cp} \cdot c_3w_3\} \\ < \quad & \{C_{total}(\Theta_{(2,1,3)}) \cdot \frac{D}{T_{cp}} = (ZT_{cm} + w_1T_{cp})(ZT_{cm} + w_3T_{cp})c_2w_2 \\ & + w_2T_{cp}(ZT_{cm} + w_3T_{cp})c_1w_1 + w_2T_{cp} \cdot w_1T_{cp} \cdot c_3w_3\} \end{aligned}$$

After cancellation of several terms, the result is $c_1w_1 < c_2w_2$. One can also prove a similar result for the case when P_2 and P_3 are switched. Then, the two conditions that must be satisfied in order to minimize C_{total} are:

$$c_1w_1 < c_2w_2 \qquad \text{and} \qquad c_2w_2 < c_3w_3$$

Equivalently,

$$c_1w_1 < c_2w_2 < c_3w_3$$

For the general N processors case, the fractions of the workload are

$$\alpha_1 = \left[1 + \sum_{n=2}^{N}\left(\prod_{i=1}^{n-1} k_i\right)\right]^{-1} = \frac{1}{D}\prod_{i=2}^{N}(ZT_{cm} + w_iT_{cp}) \tag{21}$$

$$\alpha_2 = k_1\alpha_1 = \frac{1}{D}w_1T_{cp}\prod_{i=3}^{N}(ZT_{cm} + w_iT_{cp}) \tag{22}$$

$$\alpha_3 = k_2\alpha_2 = \frac{1}{D}w_1T_{cp}\cdot w_2T_{cp}\prod_{i=4}^{N}(ZT_{cm} + w_iT_{cp}) \tag{23}$$

$$\vdots$$

$$\alpha_n = k_{n-1}\alpha_{n-1} = \frac{1}{D}\prod_{i=1}^{n-1}(w_iT_{cp})\cdot\prod_{i=n+1}^{N}(ZT_{cm} + w_iT_{cp}) \tag{24}$$

$$\vdots$$

$$\alpha_N = k_{N-1}\alpha_{N-1} = \frac{1}{D}\prod_{i=1}^{N-1}(w_iT_{cp}) \tag{25}$$

where from Eq.(13):

$$D = \sum_{n=1}^{N}\left[\prod_{i=1}^{n-1}(w_iT_{cp})\cdot\prod_{i=n+1}^{N}(ZT_{cm} + w_iT_{cp})\right] \tag{26}$$

Again, D is not changed by switching any set of w_i with any other w_j ($i, j = 1, 2, \ldots, N$ and $i \neq j$). One can see this point more easily in Eq.(19).

The total computing cost is then

$$C_{total} = \frac{T_{cp}}{D}\sum_{n=1}^{N}\left[\prod_{i=1}^{n-1}(w_iT_{cp})\cdot\prod_{i=n+1}^{N}(ZT_{cm} + w_iT_{cp})c_nw_n\right] \tag{27}$$

Consider two arbitrary adjacent processors, P_j and P_{j+1}. In the following, it will be shown that if $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is smaller than $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$, then $c_jw_j < c_{j+1}w_{j+1}$.

$$\left\{C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})\cdot\frac{D}{T_{cp}}\right.$$
$$= \sum_{n=1}^{j-1}\left[\prod_{i=1}^{n-1}(w_iT_{cp})\cdot\prod_{i=n+1}^{N}(ZT_{cm} + w_iT_{cp})c_nw_n\right]$$
$$+ (w_1T_{cp})(w_2T_{cp})\cdots(w_{j-1}T_{cp})$$

$$\times (ZT_{cm} + w_{j+1}T_{cp})(ZT_{cm} + w_{j+2}T_{cp}) \cdots (ZT_{cm} + w_N T_{cp}) c_j w_j$$

$$+ (w_1 T_{cp})(w_2 T_{cp}) \cdots (w_{j-1} T_{cp})(w_j T_{cp})$$

$$\times (ZT_{cm} + w_{j+2}T_{cp})(ZT_{cm} + w_{j+3}T_{cp}) \cdots (ZT_{cm} + w_N T_{cp}) c_{j+1} w_{j+1}$$

$$\left. + \sum_{n=j+2}^{N} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \cdot \prod_{i=n+1}^{N} (ZT_{cm} + w_i T_{cp}) c_n w_n \right] \right\}$$

$$< \quad \left\{ C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)}) \cdot \frac{D}{T_{cp}} \right.$$

$$= \sum_{n=1}^{j-1} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \cdot \prod_{i=n+1}^{N} (ZT_{cm} + w_i T_{cp}) c_n w_n \right]$$

$$+ (w_1 T_{cp})(w_2 T_{cp}) \cdots (w_{j-1} T_{cp})$$

$$\times (ZT_{cm} + w_j T_{cp})(ZT_{cm} + w_{j+2}T_{cp}) \cdots (ZT_{cm} + w_N T_{cp}) c_{j+1} w_{j+1}$$

$$+ (w_1 T_{cp})(w_2 T_{cp}) \cdots (w_{j-1} T_{cp})(w_{j+1} T_{cp})$$

$$\times (ZT_{cm} + w_{j+2}T_{cp})(ZT_{cm} + w_{j+3}T_{cp}) \cdots (ZT_{cm} + w_N T_{cp}) c_j w_j$$

$$\left. + \sum_{n=j+2}^{N} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \cdot \prod_{i=n+1}^{N} (ZT_{cm} + w_i T_{cp}) c_n w_n \right] \right\}$$

After some cancellations, the only remaining terms are:

$$(ZT_{cm} + w_{j+1}T_{cp}) c_j w_j + w_j T_{cp} \cdot c_{j+1} w_{j+1} < (ZT_{cm} + w_j T_{cp}) c_{j+1} w_{j+1} + w_{j+1} T_{cp} \cdot c_j w_j$$

Further cancellation brings:

$$c_j w_j < c_{j+1} w_{j+1}$$

Therefore, if $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is smaller than $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$, then $c_j w_j < c_{j+1} w_{j+1}$. One can also show that if $c_j w_j < c_{j+1} w_{j+1}$, then $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is smaller than $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$, by running the above proof in the reverse direction.

Since P_j is an arbitrary jth processor, if one performs the same proof for $j = 1, 2, \ldots, N-1$, the result will be

$$c_1 w_1 < c_2 w_2 < \cdots < c_N w_N$$

in which, with this order of processors, the total computing cost is minimized. □

4 Further Reducing the Total Computing Cost

The total computing cost C_{total} can be further reduced as low as

$$C_{total} = \min_{n=1,2,\ldots,N} c_n w_n T_{cp} \tag{28}$$

But there is a significant difference if one reduces C_{total} to be less than that found in the previous section. The processing finish time T_f will increase. The timing diagram for this situation appears in Fig. 3. It is easy to see that it is possible to algorithmically minimize the cost subject to a bound on the delay. First, arrange the processors such that $c_1w_1 < c_2w_2 < \cdots < c_Nw_N$ is satisfied. Note that it is assumed that the "cheapest" processor receives the workload from the outside environment and becomes the load distribution processor. Next, increase the processing finish time by allowing some delay τ. The new processing finish time is

$$T_f^{new} = T_f + \tau \tag{29}$$

Increase α_1 until $\alpha_1 w_1 T_{cp}$ reaches T_f^{new} while decreasing α_N by the same amount. The idea is that if a greater portion of the workload is processed in the cheaper processor than in the more expensive processor, the total computing cost C_{total} will be reduced. The procedure is repeated for the next processor. That is, increase $\alpha_2, \alpha_3, \ldots$, while decreasing α_N. If α_N becomes zero, then set $\alpha_N = 0$ and the load will be not delivered to the most expensive processor. Go to the next most expensive processor and decrease α_{N-1}. If α_{N-1} becomes zero too, then decrease α_{N-2}, and so forth.

The above algorithm is given for purposes of exposition. For implementation two recursive numerical algorithms will be presented in the following. The first algorithm, the cost minimizer, finds α_n which minimizes the total computing cost further when the processing finish time is bounded by T_f^{new}. The second algorithm, the finish time minimizer, finds α_n which minimizes the processing finish time for a bound on the total computing cost, namely C_{given}.

4.1 Cost Minimizer

The objective function is as follows:

OBJECTIVE: $\min_{T_f \le T_f^{new}} \text{Cost}$

The algorithm is (initially assume that all α_i's are zero.

(a) Find α_1:

$$\alpha_1 w_1 T_{cp} = T_f^{new} \Longrightarrow \alpha_1 = \frac{T_f^{new}}{w_1 T_{cp}} \tag{30}$$

(b) Find α_n (initially $n = 2$):

$$\left.\begin{aligned} \alpha_n w_n T_{cp} &= T_f^{new} - T_n \\ T_n &= \sum_{i=2}^{n} \alpha_i Z T_{cm} \end{aligned}\right\} \Longrightarrow \alpha_n = \frac{T_f^{new} - \sum_{i=2}^{n-1} \alpha_i Z T_{cm}}{Z T_{cm} + w_n T_{cp}} \tag{31}$$

(c) If $\sum_{i=1}^{n} \alpha_i < 1$,

then repeat step (b) with $n = n + 1$.

If $\sum_{i=1}^{n} \alpha_i = 1$,

then stop and $\alpha_{n+1} = \alpha_{n+2} = \cdots = \alpha_N = 0$.

If $\sum_{i=1}^{n} \alpha_i > 1$,

then stop and $\alpha_n = 1 - \sum_{i=1}^{n-1} \alpha_i$

and $\alpha_{n+1} = \alpha_{n+2} = \cdots = \alpha_N = 0$.

As can be seen, this algorithm essentially allocates load for each processor in turn (from least expensive to most expensive in terms of cost/load) up to the upper bound on finish time until all the load has been allocated. Thus the finish time constraint is satisfied while cost is minimized.

4.2 Finish Time Minimizer

The objective function is:

OBJECTIVE: $\min_{C \leq C_{given}} T_f$

The algorithm is as follows:

(a) Find α_n $(n = 1, 2, \ldots, N)$ from Eq.(8–10).

(b) Set $\alpha_r = \alpha_r - \Delta$, $(r = N$ initially) where $\Delta \ll \alpha_r$.

(c) Find α_n $(n = 1, 2, \ldots, r-1)$ by using

$$\begin{aligned}
\alpha_1 w_1 T_{cp} &= (ZT_{cm} + w_2 T_{cp})\alpha_2 \\
\alpha_2 w_2 T_{cp} &= (ZT_{cm} + w_3 T_{cp})\alpha_3 \\
&\vdots \\
\alpha_{r-2} w_{r-2} T_{cp} &= (ZT_{cm} + w_{r-1} T_{cp})\alpha_{r-1} \\
\alpha_1 + \alpha_2 + \cdots + \alpha_{r-1} &= 1 - \sum_{k=r}^{N} \alpha_k
\end{aligned}$$

(d) If $\left(C_{total} = \sum_{n=1}^{N} \alpha_n c_n w_n T_{cp}\right) \leq C_{given}$,

then stop

else repeat step (b) and (c).

But if the above inequality is invalid even with $\alpha_r = 0$,

then set $\alpha_r = 0$ and repeat step (b) and (c) with $r = r - 1$.

Here an initial minimal time allocation is made. Load is subtracted in incremental steps from the processors starting from the most expensive (in terms of cost/load) to the least expensive. A renormalization is performed at each step. The process stops when the cost is brought at or below the desired cost level. Thus the cost constraint is satisfied at the price of a larger finish time. However the finish time is minimal *given* that the cost constraint is satisfied.

Case	Sequence	T_f	C_{total}
1	$\Theta_{(1,2,3)}$	0.667	8.333
2	$\Theta_{(1,3,2)}$	0.667	8.167
3	$\Theta_{(2,1,3)}$	0.889	7.445
4	$\Theta_{(2,3,1)}$	0.889	6.667
5	$\Theta_{(3,1,2)}$	1.000	7.000
6	$\Theta_{(3,2,1)}$	1.000	6.333

Table 1: Processing finish time and total computing cost for six cases of sequences.

5 Resource Management Evaluation

Based on the previous results, a number of computer resource management results were obtained via simulation for three cases. The first case is the one described in section 2 and 3, namely finding the optimal sequence of processors for the minimal processing finish time and the minimal total computing cost when all the processors finish computing at the same time. Next, some performance observations are described for further reducing the computing cost when the processing finish time is delayed, as discussed in section 4. In the final case, the question is addressed as to whether the performance can be improved by replacing one fast but expensive processor with a number of cheap but slow processors.

5.1 Optimal Sequence

Table 1 is obtained for the values of $Z = T_{cm} = T_{cp} = 1$, $w_1 = 1$, $c_1 = 10$, $w_2 = 2$, $c_2 = 3$, $w_3 = 3$, and $c_3 = 1$. For these values, the fastest processor has the most expensive computing cost and the slowest processor has the cheapest computing cost. A number of basic points are raised in a consideration of this table. Apparently, the odd numbered examples (rows) are not of concern since in these examples C_{total} is higher than that of in the even numbered examples for the same T_f. In example 1, the processing finish time T_f is the smallest while the total processing cost C_{total} is the highest. On the other hand, the largest T_f occurs in example 6, yet C_{total} is the lowest. Here one thus has a choice between a smaller processing finish time or a smaller total computing cost. If one wants a smaller processing finish time regardless of the total computing cost, sequence $\Theta_{(1,3,2)}$ will be an appropriate choice. The sequence $\Theta_{(3,2,1)}$ is suitable when a smaller total computing cost is desirable but a smaller processing finish time is not required. By choosing the sequence $\Theta_{(2,3,1)}$, a moderate processing finish time and a moderate total computing cost will be achieved.

5.2 Further Reducing the Total Computing Cost

Two plots, Fig. 4 and Fig. 5, are obtained from the two algorithms, the cost minimizer (minimizing the total computing cost with a bound on processing finish time) and the finish time minimizer (minimizing the processing finish time with a bound on total computing cost), respectively. In both cases, $Z = T_{cm} = T_{cp} = 1$, $w_1 = 1$, $c_1 = \frac{3}{2}$, $w_2 = 2$, $c_2 = 1$, $w_3 = 3$, and $c_3 = \frac{3}{4}$. Again, for these values, the faster processor has the more expensive computing cost and the slower processor has the cheaper computing cost.

In Fig. 4, each point in the most upper curve represents the lowest possible total computing cost for the corresponding processing finish time. As shown in this figure, the total computing cost is monotonically decreasing as the processing finish time increases, which means that the lower the total computing cost is, the greater the processing finish time is, and vice versa.

In Fig. 5, the total computing cost is initially obtained at $C_{total} = 1.689$ without any delay of the processing finish time ($\tau = 0$ and $T_f = 0.674$). The finish time minimizer algorithm recursively calculates the processing finish time and the fractions of the workload for a given total computing cost ($C_{given} = 1.5$). Note that the example value of C_{given} (1.5) is the lowest possible value that can be achieved, since $\min(c_n w_n T_{cp}) = 1.5$, for all n.

5.3 Splitting Processors

An important question in current computer architecture and microelectronics is whether it is better to design a system using one very fast but expensive processor or to use many cheap but slow processors instead. This question arises, for instance, when one considers fast but expensive semiconductor technologies such as gallium arsenide in relation to the more traditional silicon implementations of computers. To answer this question, the following examination is performed. For all cases, $Z = T_{cm} = T_{cp} = 1$.

Fig. 6 depicts the case when one replaces (splits) one fast but expensive processor ($w_1 = 1, c_1 = 1$) with N cheap but slow processors ($w_n = N, c_n = \frac{1}{N}$). Note that all the processors have the same computing cost per load, i.e., $c_n w_n = 1$ for all n, including the original processor in order to preserve fairness. That is, if one splits one processor into N processors, then the computing speed of each one of N processors becomes N times slower and the computing cost of each one of N processors becomes N times cheaper. When one fast but expensive processor is used, the total computing cost is

$$C_{total} = \alpha_1 c_1 w_1 T_{cp} = 1$$

since $\alpha_1 = c_1 w_1 = T_{cp} = 1$. The total computing cost when N cheap but slow processors are used is

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

since $c_n w_n = T_{cp} = 1$ for all n. Thus the total computing cost is unchanged with respect to the number of splits, or more explicitly, the number of cost equivalent processors, as shown in Fig. 6(b). This is thus a cost conservative splitting of the fast processor. However, the processing finish time becomes larger as one increases the number of cost equivalent processors. Since the computing speed of the load origination processor also becomes N times slower when the processor is split to N cost equivalent processors, the processing finish time increases as the number of splits increases.

The next situation studied is when there are two processors with different costs per load. Which one is better to split for better performance? Some examples are presented in Fig. 7, 8 and 9, in an effort to answer the above question. For all cases, $w_1 = c_1 = 1$ and $w_2 = 2$. Note in all of these cases if P_2 is split then P_1 processes load first, followed by the split P_2 processors. Alternately, if P_1 is split, the split P_1 processors process load first, followed by the P_2 processor. Fig. 7 illustrates the case when $c_1 = 1$, i.e., P_2 is twice expensive in cost per load compared to P_1. As shown in Fig. 7(b), splitting the cheaper cost per load processor (P_1) allows for less total computing cost, while splitting the more expensive cost per load processor (P_2), results in a higher total computing cost.

The underlying explanation for this phenomenon is as follows. In this example cost is inversely proportional to computing speed (as $c_1 = c_2 = 1.0$). The load allocation equations automatically allocate most of the load to the fastest (cheapest) processors and allocate very little load to the slower (more expensive) processors. Thus a split of P_1, the cheapest processor, allows a better subset of very cheap processors to be created. This leads to the lowest cost solution.

Fig. 8 shows the total computing cost when the cost per load of P_1 and P_2 are reversed with respect to that of in Fig. 7. The computing cost per load of P_2 is less than that of P_1, so splitting P_2 produces the lower total computing cost.

The processing finish time is independent of any cost factor and it increases as one splits P_1 because splitting P_1 reduces the speed of the load origination processor. This is shown in Fig. 9.

6 Conclusions

In this paper the optimal sequence of the processors for the minimal processing finish time and the minimal total computing cost was examined. It was found that a smaller total computing cost can be obtained by arranging the sequence of processors such that the cheaper processor receives the workload earlier than the more expensive processor does. Here cost is defined in terms of cost per load, $c_n w_n$. But these two factors, the processing finish time and the total computing cost, were found to involve a trade-off between each other when the faster processor has a more expensive computing cost and the slower processor has a less expensive computing cost, as in most practical situations.

Splitting a fast but expensive single processor into a number of slower but cheaper processors in a linear and conservative manner was found to actually raise the processing finish time. It was also found in this paper that when there is more than one processor with different computing cost per load one can reduce the total computing cost by splitting the less expensive one.

7 Appendix

Theorem 2 *For a given set of processor speed, $S(w) = \{w_1, w_2, \ldots, w_N\}$, the processing finish time is minimized if*

$$w_1 = \min(w_1, w_2, \ldots, w_N) \tag{32}$$

□

Proof: Let us first prove this theorem in the case where there are three processors ($N = 3$). From Eq.(9), the fraction of the workload for load origination processor is

$$\alpha_1 = [1 + k_1 + k_1 k_2]^{-1} \tag{33}$$

where:

$$k_1 = \frac{w_1 T_{cp}}{ZT_{cm} + w_2 T_{cp}} \tag{34}$$

$$k_2 = \frac{w_2 T_{cp}}{ZT_{cm} + w_3 T_{cp}} \tag{35}$$

The processing finish time is then [19, 20]:

$$\begin{aligned} T_f &= \alpha_1 w_1 T_{cp} \\ &= \frac{w_1 T_{cp}}{1 + \frac{w_1 T_{cp}}{ZT_{cm} + w_2 T_{cp}} + \frac{w_1 T_{cp}}{ZT_{cm} + w_2 T_{cp}} \frac{w_2 T_{cp}}{ZT_{cm} + w_3 T_{cp}}} \\ &= \frac{w_1 T_{cp}(ZT_{cm} + w_2 T_{cp})(ZT_{cm} + w_3 T_{cp})}{(ZT_{cm})^2 + ZT_{cm}(w_1 + w_2 + w_3)T_{cp} + (w_1 w_2 + w_2 w_3 + w_3 w_1)T_{cp}^2} \end{aligned} \tag{36}$$

The value of denominator of T_f is not changed by switching any w_i with any other w_j ($i, j = 1, 2, 3$ and $i \neq j$). However, the numerator is minimized when w_1 is chosen to be the smallest among w_1, w_2 and w_3. This can be explained as follows: If $w_1 T_{cp}(ZT_{cm} + w_2 T_{cp}) < w_2 T_{cp}(ZT_{cm} + w_1 T_{cp})$, then it is true with cancellation that $w_1 < w_2$ since $ZT_{cm} > 0$. Similarly, if $w_1 T_{cp}(ZT_{cm} + w_3 T_{cp}) < w_3 T_{cp}(ZT_{cm} + w_1 T_{cp})$, it is also true that $w_1 < w_3$.

For general N processors case, the first fraction of the workload is

$$\alpha_1 = [1 + k_1 + k_1 k_2 + \cdots + (k_1 k_2 \cdots k_{N-1})]^{-1} \tag{37}$$

where:

$$k_n = \frac{w_n T_{cp}}{ZT_{cm} + w_{n+1} T_{cp}} \qquad n = 1, 2, \ldots, N-1 \tag{38}$$

The processing finish time is now:

$$\begin{aligned} T_f &= \alpha_1 w_1 T_{cp} \\ &= \frac{w_1 T_{cp}}{1 + k_1 + k_1 k_2 + \cdots + (k_1 k_2 \cdots k_{N-1})} \\ &= \frac{w_1 T_{cp} \prod_{n=2}^{N} (Z T_{cm} + w_n T_{cp})}{\sum_{n=1}^{N} \left[\prod_{i=1}^{n-1} (w_i T_{cp}) \cdot \prod_{i=n+1}^{N} (Z T_{cm} + w_i T_{cp}) \right]} \end{aligned} \tag{39}$$

Again, the denominator of T_f is independent of switching any w_i with any other w_j ($i, j = 1, 2, \ldots, N$ and $i \neq j$). 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. $\square$

References

[1] I. Ahmad, A. Ghafoor, and G. C. Fox, "Hierarchical scheduling of dynamic parallel computations on hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 20, pp. 317–329, 1994.

[2] S. H. Bokhari, "A network flow model for load balancing in circuit-switched multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 6, pp. 649–657, June, 1993.

[3] C.-H. Lee, D. Lee, and M. Kim, "Optimal task assignment in linear array networks," *IEEE Transactions on Computers*, vol. 41, no. 7, pp. 877–880, July, 1992.

[4] K. K. Goswami, M. Devarakonda, and R. K. Iyer, "Prediction-based dynamic load-sharing heuristics," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 6, pp. 638–648, June, 1993.

[5] G. Huang and W. Ongsakul, "An efficient load-balancing processor scheduling algorithm for parallelization of gauss-seidel type algorithms," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 350–358, 1994.

[6] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, and J. Telle, "Mapping divide and conquer algorithms to parallel computers," In *Proceedings of the 1990 International Conference on Parallel Architectures*, 1990, pp. 128–135.

[7] K. Ramamrithamm, J. A. Stankovic, and P.-F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184–194, April, 1990.

[8] X. Qian and Q. Yang, "An analytical model for load balancing on symmetric multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 20, pp. 198–211, 1994.

[9] Y.-C. Chang and K. G. Shin, "Optimal load sharing in distributed real-time systems," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 38–50, September, 1993.

[10] K. G. Shin and M.-S. Chen, "On the number of acceptable task assignments in distributed computing systems," *IEEE Transactions on Computers*, vol. 39, no. 1, pp. 99–110, January, 1990.

[11] D.-T. Peng and K. G. Shin, "A new performance measure for scheduling independent real-time tasks," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 11–26, September, 1993.

[12] G. C. Sih and E. A. Lee, "Declustering: A new multiprocessor scheduling technique," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 6, pp. 625–637, June, 1993.

[13] J. Xu and K. Hwang, "Heuristic methods for dynamic load balancing in a message-passing multicomputer," *Journal of Parallel and Distributed Computing*, vol. 18, no. 1, pp. 1–13, May, 1993.

[14] J. Blazewicz, M. Drabowski, and J. Weglarz, "Scheduling multiprocessor tasks to minimize schedule length," *IEEE Transactions on Computers*, vol. C-35, pp. 389–398, May, 1986.

[15] J. Du and J. Y.-T. Leung, "Complexity of scheduling parallel task systems," *SIAM Journal on Discrete Mathematics*, vol. 2, pp. 473–487, November, 1989.

[16] W. Zhao, K. Ramamritham, and J. A. Stankovic, "Preemptive scheduling under time and resource constraints," *IEEE Transactions on Computers*, vol. C-36, pp. 949–960, August, 1987.

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

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

[19] S. Bataineh and T. G. Robertazzi, "Distributed computation for a bus network with communication delays," In *Proceedings of the 1991 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore, MD. March, 1991, pp. 709–714.

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

[21] S. Bataineh and T. G. Robertazzi, "Ultimate performance limits for networks of load sharing processors," In *Proceedings of the 1992 Conference on Information Science and Systems*, Princeton University, Princeton, NJ. March, 1992, pp. 794–799.

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

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

[24] J. Sohn and T. G. Robertazzi, "Optimal divisible job load sharing for bus networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 1,, January, 1996.

[25] J. Sohn and T. G. Robertazzi, "A multi-job load sharing strategy for divisible jobs on bus networks," Technical Report 697, SUNY at Stony Brook College of Engineering and Applied Science, August, 1994.

[26] J. Sohn and T. G. Robertazzi, "An optimal load sharing strategy for divisible jobs with time-varying processor speed and channel speed," Technical Report 706, SUNY at Stony Brook College of Engineering and Applied Science, January, 1995. Conference version: *Proceedings of the ISCA International Conference on Parallel and Distributed Computing Systems*, Orlando FL, Sept. 1995, pp. 27-32.

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

[28] D. Ghose and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," *Journal of Parallel nd Distributed Computing*, vol. 23, pp. 293–305, November, 1994.

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

[30] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 31, No. 2, pp. 555-567, April 1995.

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

[32] V. Bharadwaj, D. Ghose, and V. Mani, "A study of optimality conditions for load distribution in tree networks with communication delays," Technical Report 423/GI/02-92, Dept. of Aerospace Engineering, Indian Institute of Science, Bangalore, India, December, 1992.

[33] H. J. Kim, G. I. Jee, and J. G. Lee, "Optimal load distribution for tree network processors," submitted for publication.

[34] J. Blazewicz and M. Drozdowski, "Scheduling divisible jobs on hypercubes," Technical Report R-94/002, Poznan University of Technology Institute of Computing Science Research Report, April, 1994.

[35] J. Blazewicz and M. Drozdowski, "The performance limits of a two-dimensional network of load-sharing processors," Technical Report R-94/007, Poznan University of Technology Institute of Computing Science Research Report, November, 1994.

[36] J. Blazewicz and M. Drozdowski, "Distributed processing of divisible jobs with communication startup costs," Technical Report R-94/006, Poznan University of Technology Institute of Computing Science Research Report, December, 1994.

[37] E. Haddad, "Communication Protocol for Optimal Redistribution of Divisible Load in Distributed Real-time Systems," *Proceedings of the ISMM International Conference on Intelligent Information Management Systems*, Washington DC, June 1994, pp. 39-42.

Figure Captions

Fig. 1. Distributed computing system consisting of N processors equipped with front-end processors connected through a bus.

Fig. 2. Timing diagram of N bus interconnected processors with load origination at P_1

Fig. 3. Timing diagram of N bus interconnected processors for further reducing the total computing cost C_{total} by increasing the processing finish time.

Fig. 4. The total computing cost and α's according to the cost minimizer algorithm.

Fig. 5. The total computing cost and α's according to the finish time minimizer algorithm.

Fig. 6. (a) Replacing one processor with N cost equivalent processors.
(b) The total computing cost and processing finish time against the number of splits (processors).

Fig. 7. (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the cost per load of P_1 is less than that of P_2.
(b) The total computing cost C_{total} against the number of splits.

Fig. 8. (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the cost per load of P_1 is higher than that of P_2.
(b) The total computing cost C_{total} against the number of splits.

Fig. 9. (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the computing speed of P_1 is faster than that of P_2.
(b) The processing finish time T_f against the number of splits.

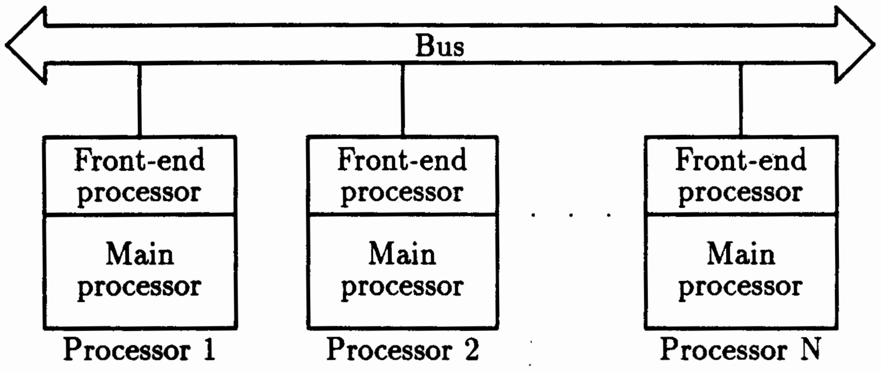


Figure 1: Distributed computing system consisting of N processors equipped with front-end processors connected through a bus.

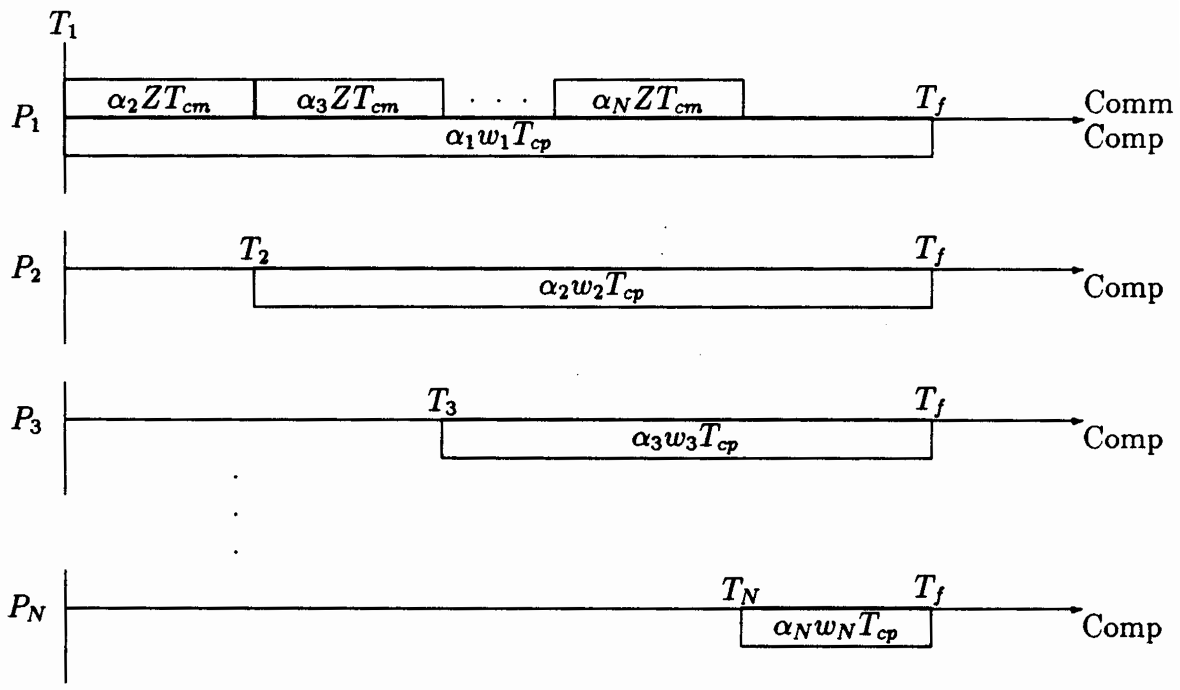


Figure 2: Timing diagram of N bus interconnected processors with load origination at P_1.

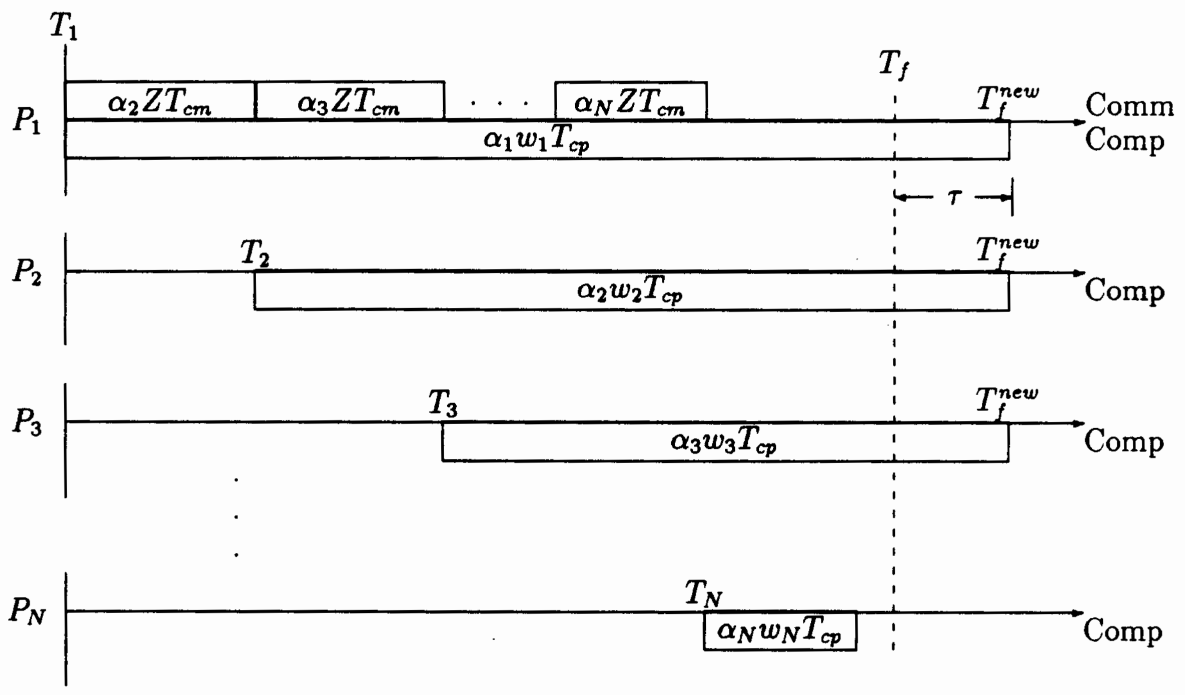


Figure 3: Timing diagram of N bus interconnected processors for further reducing the total computing cost C_{total} by increasing the processing finish time.

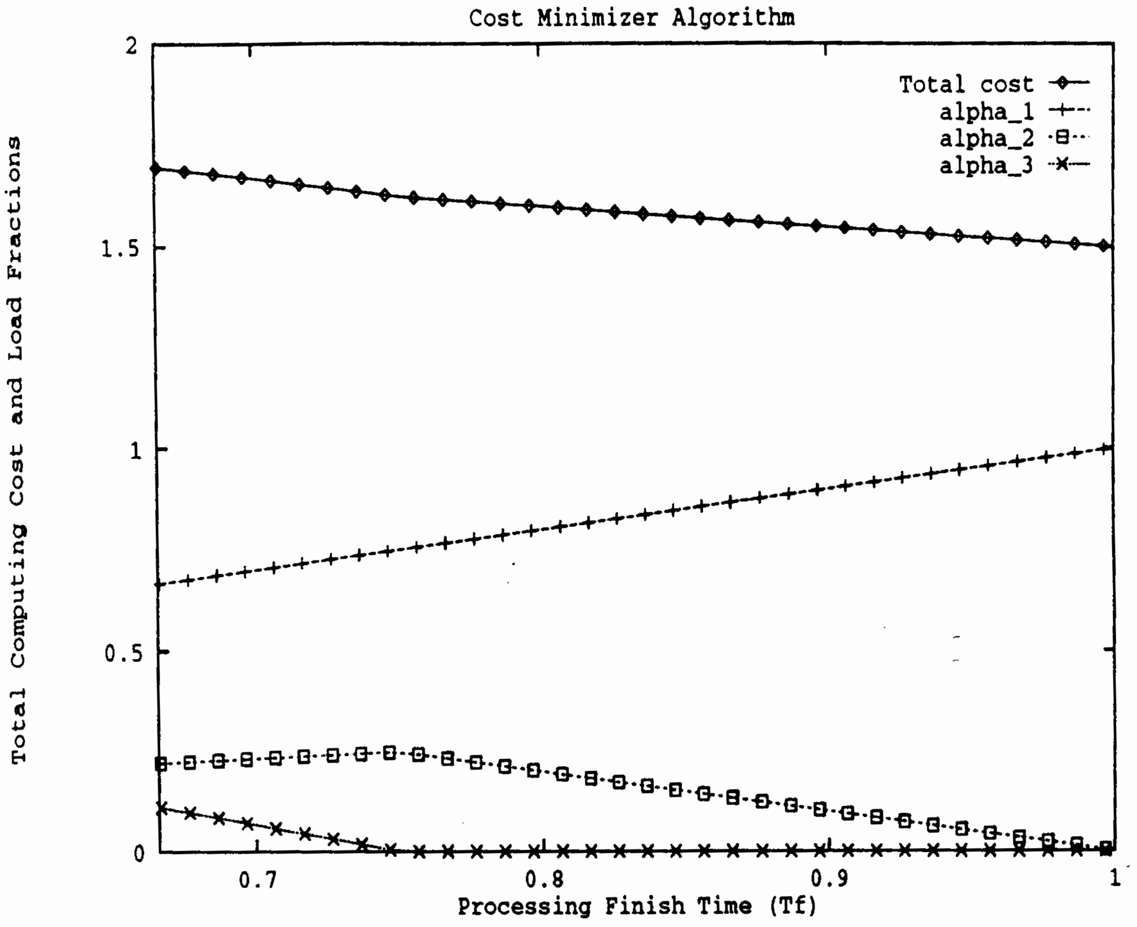


Figure 4: The total computing cost and α's according to the cost minimizer algorithm.

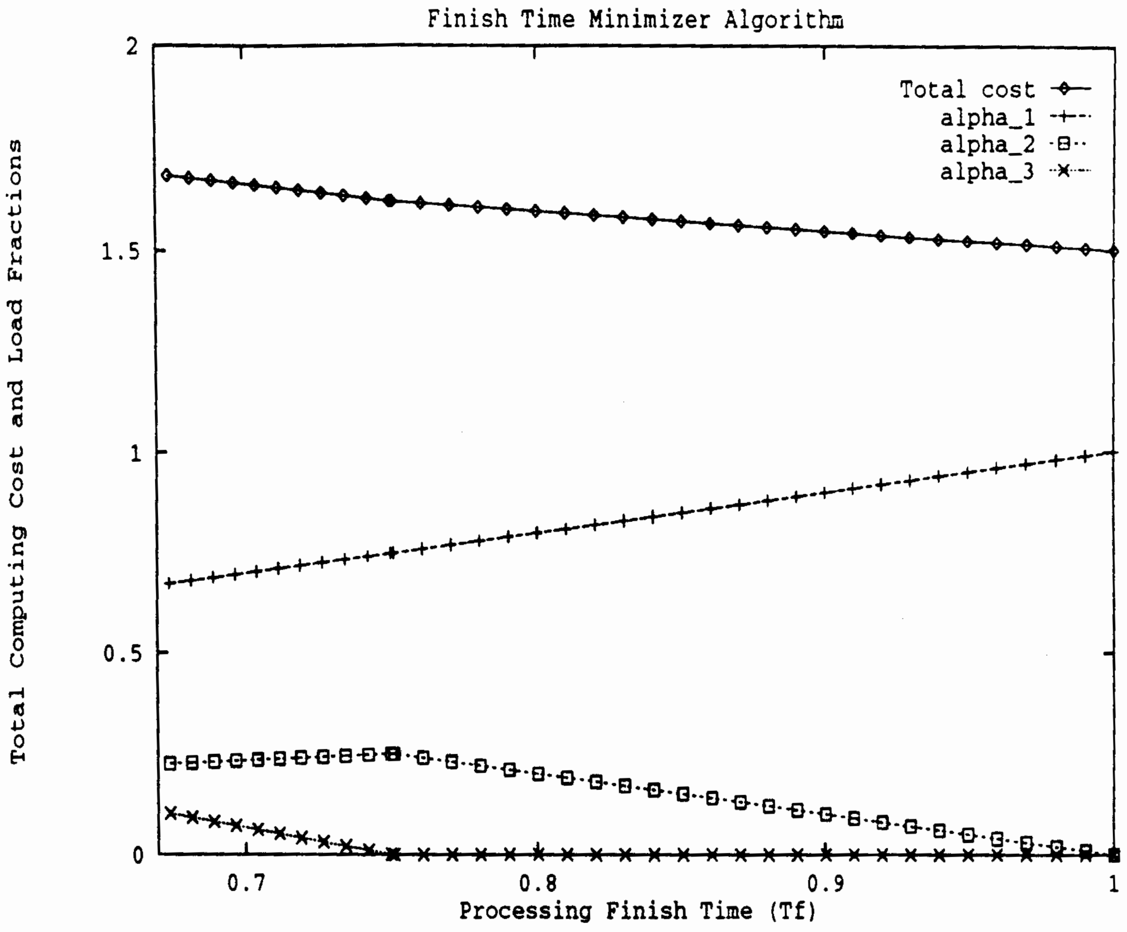
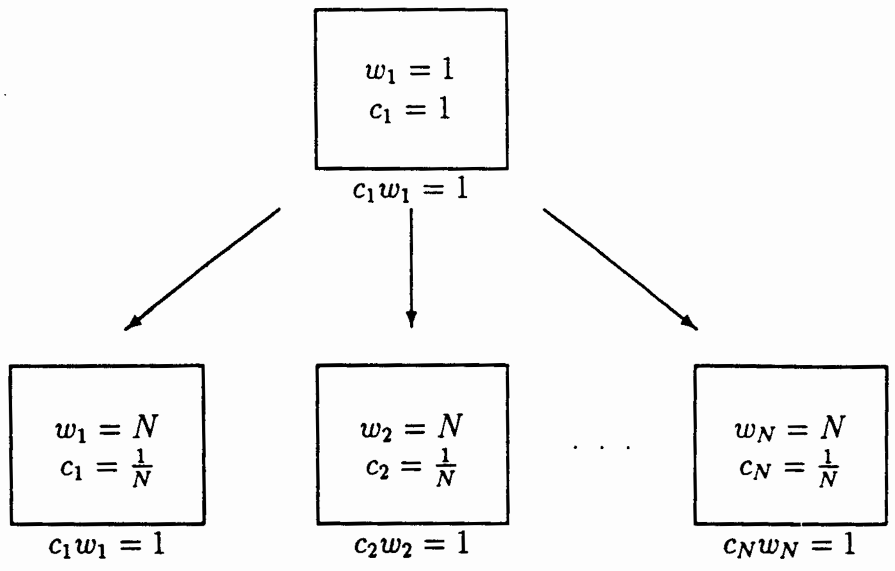
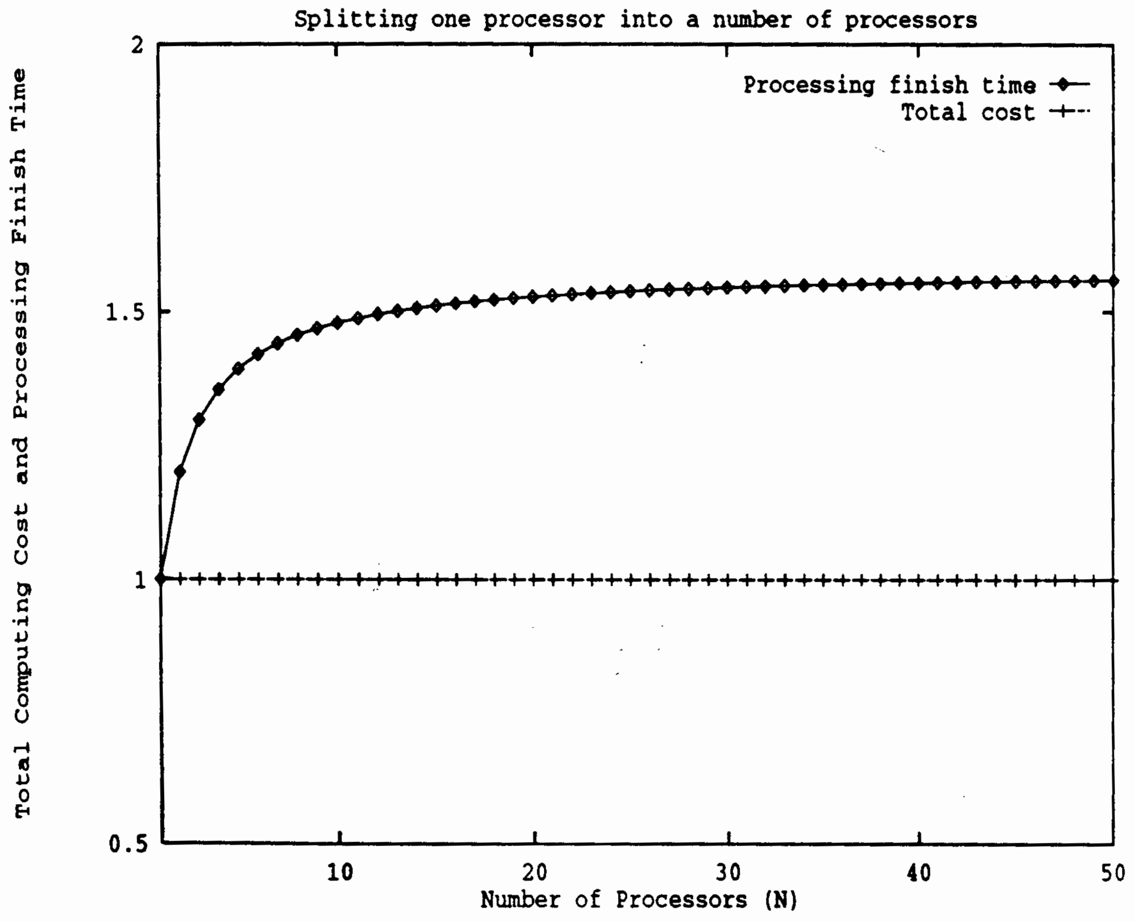


Figure 5: The total computing cost and α's according to the finish time minimizer algorithm.

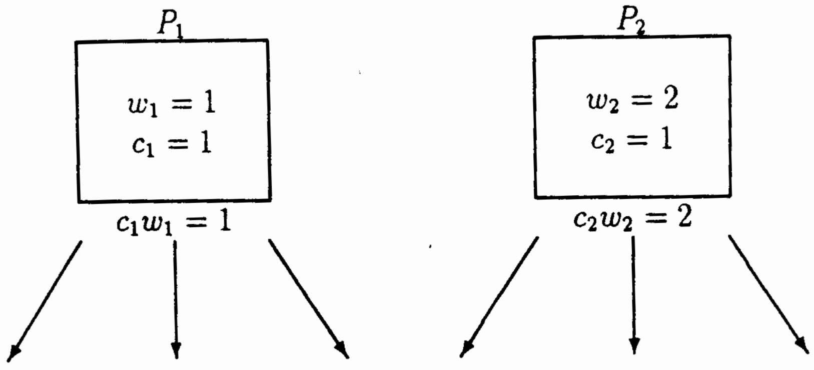


(a)

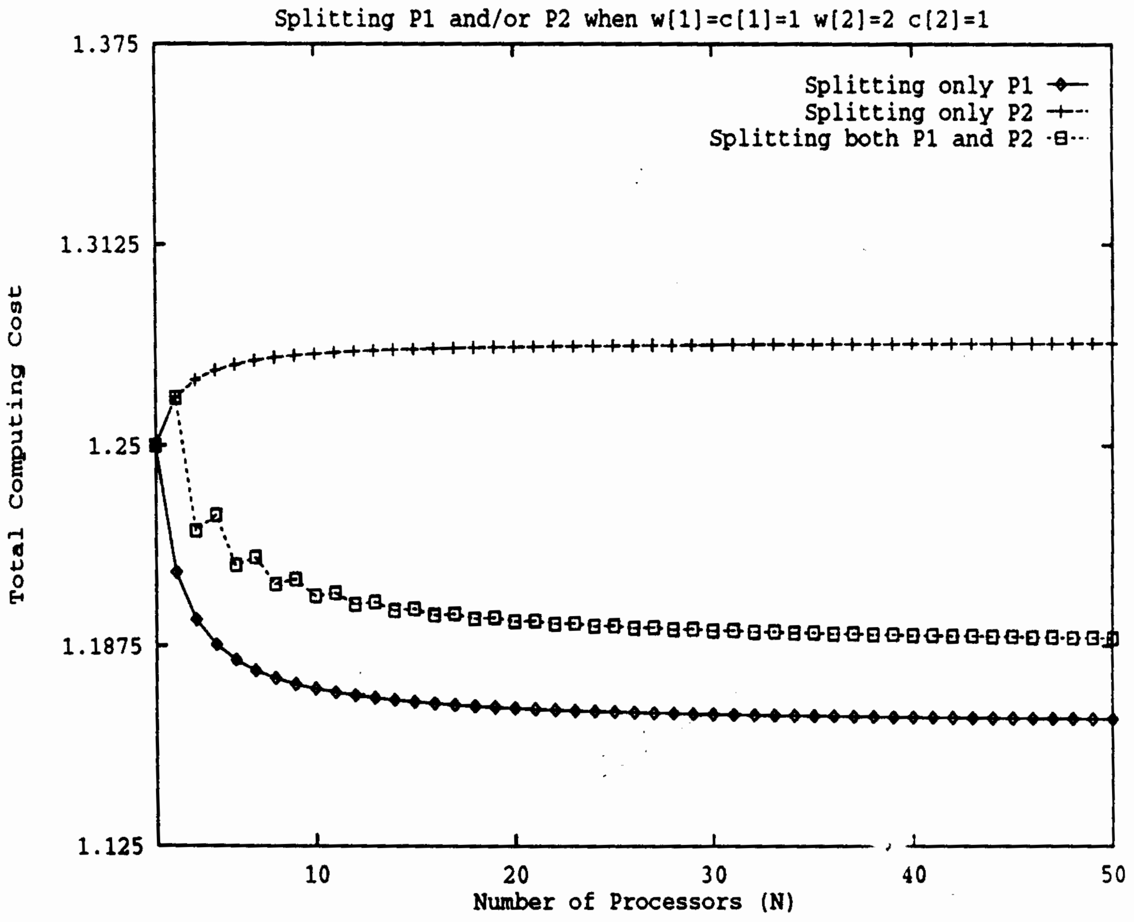


(b)

Figure 6: (a) Replacing one processor with N cost equivalent processors. (b) The total computing cost and processing finish time against the number of splits (processors).

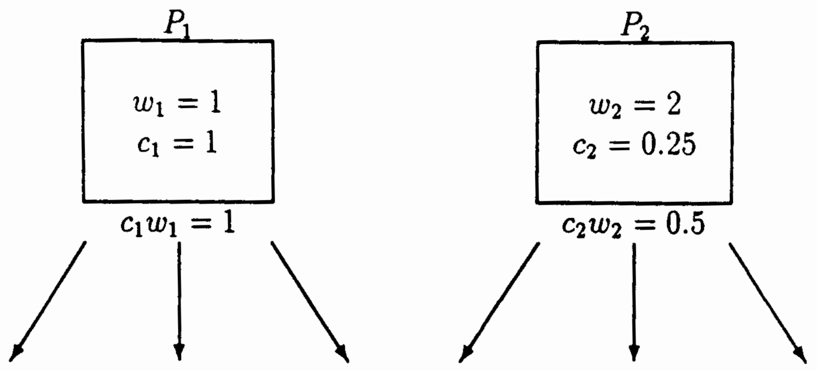


(a)

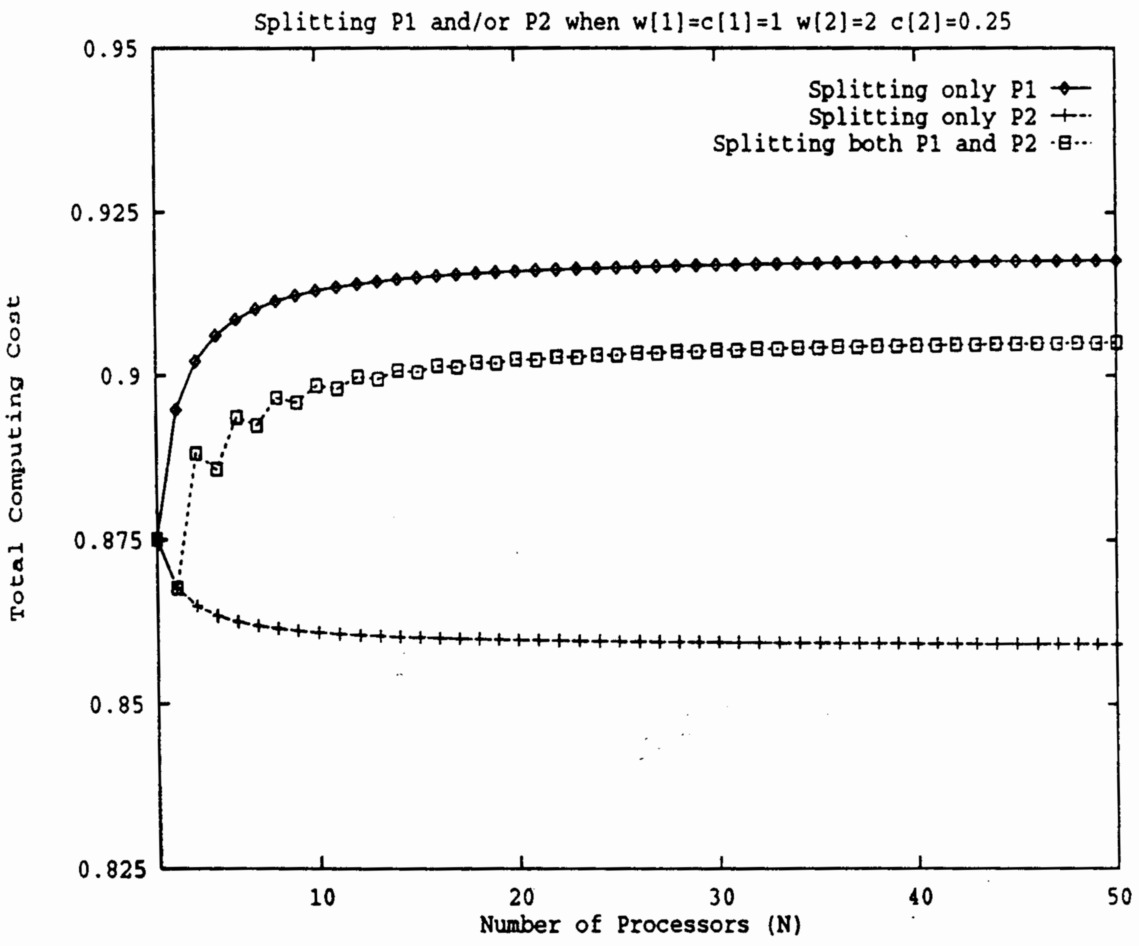


(b)

Figure 7: (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the cost per load of P_1 is less than that of P_2. (b) The total computing cost C_{total} against the number of splits.

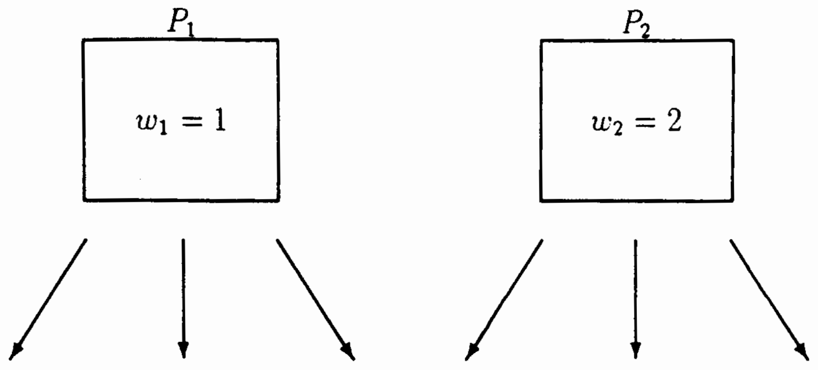


(a)

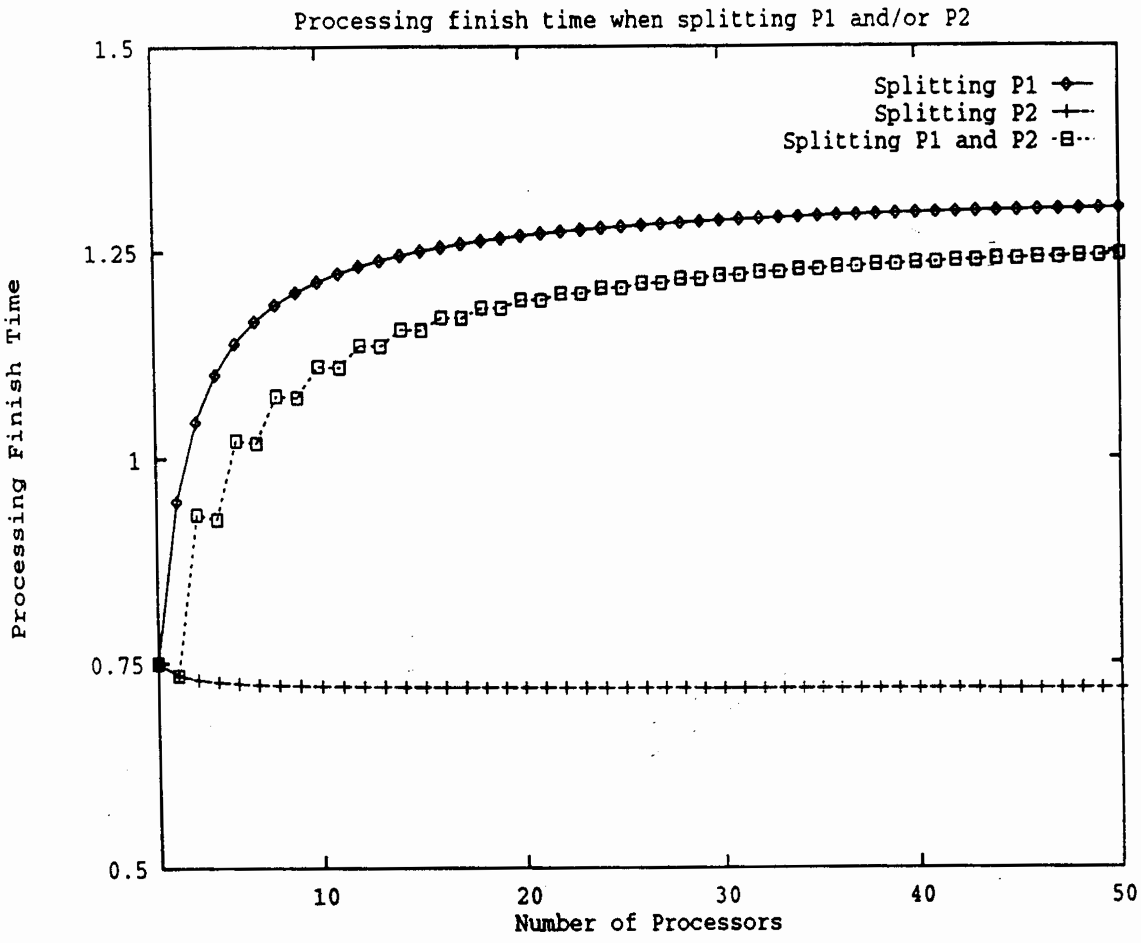


(b)

Figure 8: (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the cost per load of P_1 is higher than that of P_2. (b) The total computing cost C_{total} against the number of splits.



(a)



(b)

Figure 9: (a) Splitting P_1 and/or P_2 into a number of cost equivalent processors when the computing speed of P_1 is faster than that of P_2. (b) The processing finish time T_f against the number of splits.