Optimizing Computing Costs Using Divisible Load Analysis

Jeeho Sohn, Thomas G. Robertazzi, *Senior Member*, *IEEE*, and Serge Luryi, *Fellow*, *IEEE*

Abstract—A bus oriented network where there is a charge for the amount of divisible load processed on each processor is investigated. A cost optimal processor sequencing result is found which involves assigning load to processors in nondecreasing order of the cost per load characteristic of each processor. More generally, one can trade cost against solution time. Algorithms are presented to minimize computing cost with an upper bound on solution time and to minimize solution time with an upper bound on cost. 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. The type of questions investigated here are important for future computer utilities that perform distributed computation for some charge.

Index Terms—Bus network, computer utility, cost, divisible load, load sharing.

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 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.

There are many possible ways to classify load sharing problems. One of them is the classification by the type of load (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 so that all of 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 more 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 occur in signal and image processing, experimental data processing, Kalman filtering, cryptography, and genetic algorithms.

- *J. Sohn is with AT&T, 200 Laurel Ave., Middletown, NJ 07748.*
- *T.G. Robertazzi and S. Luryi are with the Department of Electrical Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794. E-mail: tom@sbee.sunysb.edu.*

Manuscript received 20 Oct. 1995; revised 30 Apr. 1997.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100010.

Optimal load sharing for divisible loads was first considered for linear daisy chain networks in [17]. Related results later appeared for tree networks [18], bus networks [19], [20], hypercubes [34], and two-dimensional meshes [35]. There was also work on asymptotic results [21], [23], [28], closed form solutions [22], time-varying models [26], multiple job submission [25], load distribution sequences [27], [29], [30], [31], [33], the modeling of fixed communication delays [36], and real time systems [37]. The first book-length treatment of divisible load analysis appears in [45].

All of this prior work on divisible load analysis 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. 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.

There is a fairly large literature on economic models for computer and telecommunication networks. A representative sample appears in [39], [40], [41], [42], [43], [44]. But, until now, there has been no research in divisible load theory on the effect of computing cost. When investigating the role of computing cost, it is apparent that the user should be able to trade cost against solution time. That is, very fast solutions are more expensive than slower solutions. Put another way, a faster processor usually has a more expensive computing cost and a slower processor has a 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. 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

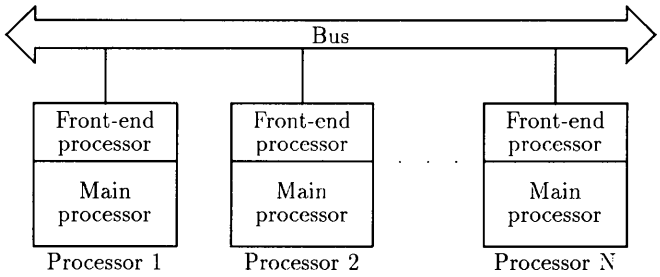


Fig. 1. Distributed computing system consisting of N processors equipped with front-end processors connected through a bus.

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 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 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 [45].

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 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. 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 (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.

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}.

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

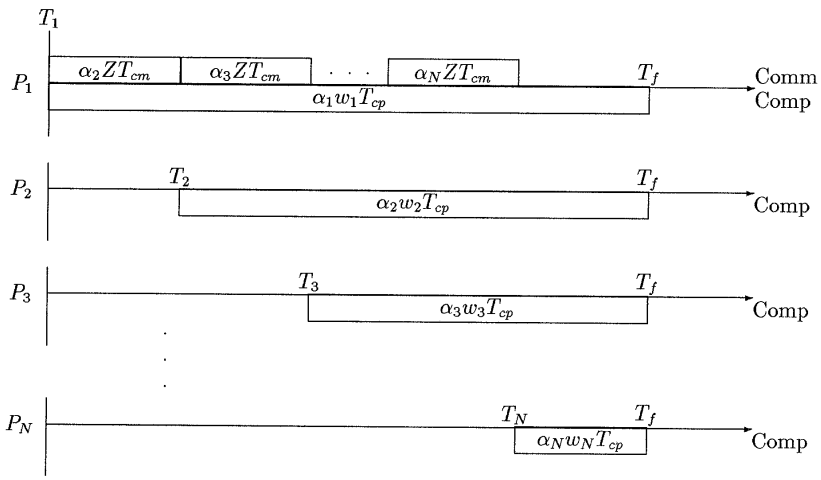


Fig. 2. Timing diagram of N bus interconnected processors with load origination at P_1.

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

$$T_f - T_n = \alpha_n w_n T_{cp} \qquad n = 1, 2, \ldots, N \tag{2}$$

$$T_{n+1} - T_n = \alpha_{n+1} Z T_{cm} \qquad n = 1, 2, \ldots, N-1. \tag{3}$$

Then, (1) can be rewritten using (2) and (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

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

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

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

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

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

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 queuing and Petri networks. It is not clear at this time whether this similarity is more than coincidental.

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 processor, 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 inverse 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 inverse processor speeds, $S(w) = \{w_1, w_2, \ldots, w_N\}$, is given, the processing finish time for the type of network being discussed here depends on the speed of the load origination processor (w_1). The finish time is minimized [20] when w_1 is chosen to be the smallest (that is, the highest speed) in the set $S(w)$. As an example, the processing finish time for a three processor network is then [19], [20]:

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

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

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

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 (i.e., fastest) w_i.

Equation (13) will be used in the next section. Note that, as shown in the Appendix, the processing finish time is independent of the load distribution sequence to the processors. 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 [30].

3 Minimizing the Total Computing Cost

If the computing "costs" 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. 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 *inverse computing speed* of the nth 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, 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 (14) is described in the following theorem. Note the abuse of notation, we use the sequence 1,2 ... N irrespective of $P_1, P_2 \ldots P_N$.

LEMMA 1. *If $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is less than or equal to $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$, then $c_j w_j \le c_{j+1} w_{j+1}$.*

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

$$\alpha_1 = \left[1 + \sum_{n=2}^{N}\left(\prod_{i=1}^{n-1} k_i\right)\right]^{-1} = \frac{1}{D}\prod_{i=2}^{N}\left(ZT_{cm} + w_i T_{cp}\right) \tag{15}$$

$$\alpha_2 = k_1\alpha_1 = \frac{1}{D} w_1 T_{cp}\prod_{i=3}^{N}\left(ZT_{cm} + w_i T_{cp}\right) \tag{16}$$

$$\alpha_3 = k_2\alpha_2 = \frac{1}{D} w_1 T_{cp} \cdot w_2 T_{cp}\prod_{i=4}^{N}\left(ZT_{cm} + w_i T_{cp}\right) \tag{17}$$

$$\vdots$$

$$\alpha_n = k_{n-1}\alpha_{n-1} = \frac{1}{D}\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right) \tag{18}$$

$$\vdots$$

$$\alpha_N = k_{N-1}\alpha_{N-1} = \frac{1}{D}\prod_{i=1}^{N-1}\left(w_i T_{cp}\right), \tag{19}$$

where, from (13),

$$D = \sum_{n=1}^{N}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)\right]. \tag{20}$$

Again, D is not changed by switching any set of w_i with any other w_j ($i, j = 1, 2, \ldots, N$ and $i \ne j$).

The total computing cost is, from (14), then

$$C_{total}\left(\Theta_{(1,2,\ldots j,j+1,\ldots N)}\right) = \frac{T_{cp}}{D}\sum_{n=1}^{N}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)c_n w_n\right]. \tag{21}$$

Consider two arbitrary adjacent processors, P_j and P_{j+1}, and let their positions in the sequence be interchanged. In the following, it will be shown that, if $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is less than or equal to $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$, then $c_j w_j \le c_{j+1} w_{j+1}$.

$$\begin{aligned}
&\left\{C_{total}\left(\Theta_{(1,2,\ldots,j,j+1,\ldots N)}\right)\cdot\frac{D}{T_{cp}}\right.\\
&= \sum_{n=1}^{j-1}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)c_n w_n\right]\\
&+ \left(w_1 T_{cp}\right)\left(w_2 T_{cp}\right)\cdots\left(w_{j-1} T_{cp}\right)\\
&\quad\times\left(ZT_{cm} + w_{j+1} T_{cp}\right)\left(ZT_{cm} + w_{j+2} T_{cp}\right)\cdots\left(ZT_{cm} + w_N T_{cp}\right)c_j w_j\\
&+ \left(w_1 T_{cp}\right)\left(w_2 T_{cp}\right)\cdots\left(w_{j-1} T_{cp}\right)\left(w_j T_{cp}\right)\\
&\quad\times\left(ZT_{cm} + w_{j+1} T_{cp}\right)\left(ZT_{cm} + w_{j+3} T_{cp}\right)\cdots\left(ZT_{cm} + w_N T_{cp}\right)c_{j+1} w_{j+1}\\
&\left.+ \sum_{n=j+2}^{N}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)c_n w_n\right]\right\}\\
&\le \left\{C_{total}\left(\Theta_{(1,2,\ldots,j+1,j,\ldots N)}\right)\cdot\frac{D}{T_{cp}}\right.\\
&= \sum_{n=1}^{j-1}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)c_n w_n\right]\\
&+ \left(w_1 T_{cp}\right)\left(w_2 T_{cp}\right)\cdots\left(w_{j-1} T_{cp}\right)\\
&\quad\times\left(ZT_{cm} + w_j T_{cp}\right)\left(ZT_{cm} + w_{j+2} T_{cp}\right)\cdots\left(ZT_{cm} + w_N T_{cp}\right)c_{j+1} w_{j+1}\\
&+ \left(w_1 T_{cp}\right)\left(w_2 T_{cp}\right)\cdots\left(w_{j-1} T_{cp}\right)\left(w_{j+1} T_{cp}\right)\\
&\quad\times\left(ZT_{cm} + w_{j+2} T_{cp}\right)\left(ZT_{cm} + w_{j+3} T_{cp}\right)\cdots\left(ZT_{cm} + w_N T_{cp}\right)c_j w_j\\
&\left.+ \sum_{n=j+2}^{N}\left[\prod_{i=1}^{n-1}\left(w_i T_{cp}\right)\cdot\prod_{i=n+1}^{N}\left(ZT_{cm} + w_i T_{cp}\right)c_n w_n\right]\right\}.
\end{aligned}$$

After some cancellations, the only remaining terms are

$$\begin{aligned}
&(ZT_{cm} + w_{j+1} T_{cp})c_j w_j + w_j T_{cp} \cdot c_{j+1} w_{j+1} \le\\
&(ZT_{cm} + w_j T_{cp})c_{j+1} w_{j+1} + w_{j+1} T_{cp} \cdot c_j w_j.
\end{aligned}$$

Further cancellation brings

$$c_j w_j \le c_{j+1} w_{j+1}.$$

This proves Lemma 1 and leads to the next lemma. □

LEMMA 2. *The total computing cost for the bus network of this paper, C_{total}, is minimized over all load distribution sequences only if the sequence of the load distribution is arranged to satisfy the following condition:*

$$c_1 w_1 \le c_2 w_2 \le \cdots \le c_N w_N. \tag{22}$$

PROOF. A proof by contradiction will be given. Suppose that one has N processors in some load distribution sequence other than (22) that minimizes cost over all load distribution sequences. Assume that the $c_n w_n$ n = 1, 2 ... N are not all identically valued. Then, there is at least one pair of adjacent processors such that $c_j w_j > c_{j+1} w_{j+1}$. But, this fact and the presumed cost minimization of the indicated sequence contradict Lemma 1. □

THEOREM 1. *The total computing cost for the bus network of this paper, C_{total}, is uniquely minimized over all load distribution sequences if and only if the sequence of the load distribution is arranged to satisfy the following condition:*

$$c_1 w_1 \le c_2 w_2 \le \cdots \le c_N w_N. \tag{23}$$

PROOF. The "only if" part of the proof is supplied by Lemma 2. For the "if" part, one can show that, if $c_j w_j \le c_{j+1} w_{j+1}$, 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 of

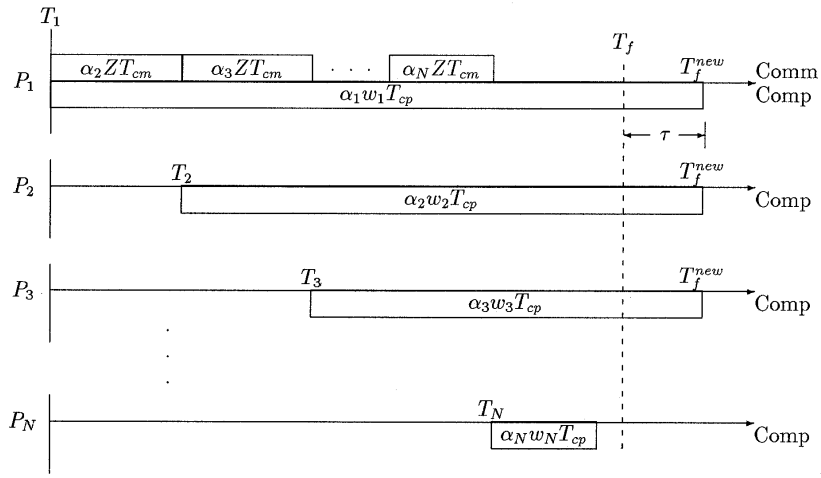


Fig. 3. Timing diagram of N bus interconnected processors for further reducing the total computing cost C_{total} by increasing the processing finish time.

Lemma 1 in the reverse direction. This mainly involves doing equivalent algebraic manipulations to both sides of the equations until it is shown that $C_{total}(\Theta_{(1,2,\ldots,j,j+1,\ldots,N)})$ is less than or equal to $C_{total}(\Theta_{(1,2,\ldots,j+1,j,\ldots,N)})$.

For any initial load distribution sequence, by continuing to interchange selected adjacent processors one can lower or, at least, maintain at the same level, the overall cost. This can be done constructively by interchanging the lowest cost per load processors with its (their) left neighbor(s) until it (they) can not be moved further "left". That is, until it (they) are at the lowest possible index. One then repeats this procedure with the second lowest cost per load processor(s) and so on ... One eventually reaches the sequence

$$c_1 w_1 \le c_2 w_2 \le \cdots \le c_N w_N.$$

Since the above series of transformations can be done starting from any initial load distribution sequence, we have found the *unique* load distribution sequence for which cost can not be further reduced. □

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.

Note that a simplified proof for the $N = 3$ processor case appears in [38].

4 Further Reducing the Total Computing Cost

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

$$C_{total} = \min_{n=1,2,\ldots N} c_n w_n T_{cp}. \tag{24}$$

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

$$T_f^{new} = T_f + \tau. \tag{25}$$

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 α_2, α_3, …, 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 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 with a bound on the total computing cost, namely C_{given}.

4.1 Cost Minimizer

The objective function is as follows:

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

Assume that the $c_i w_i$s are in nondecreasing order. Also, initially assume that all α_is are zero. The algorithm is:

a) Find α_1:

$$\alpha_1 w_1 T_{cp} = T_f^{new} \Rightarrow \alpha_1 = \frac{T_f^{new}}{w_1 T_{cp}} \tag{26}$$

b) Find α_n (initially, $n = 2$):

$$\left.\begin{aligned} \alpha_n w_n T_{cp} &= T_f^{new} - T_n \\ T_n &= \sum_{i=2}^{n} \alpha_i Z T_{cm} \end{aligned}\right\} \Rightarrow \alpha_n = \frac{T_f^{new} - \sum_{i=2}^{n-1} \alpha_i Z T_{cm}}{Z T_{cm} + w_n T_{cp}} \tag{27}$$

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: $\min_{C \le C_{given}} T_f$

One can develop an algorithm for this problem by first ordering the processors in terms of nondecreasing computing cost per load (c_1w_1, c_2w_2, ... c_iw_i ... c_Nw_N). Then, it is clear that, for the subset of $i = 1, 2, \ldots r$ processors, finish time (from (8)-(11)) is a decreasing function of r. One must find the maximal value of r such that the total cost is (just) below the desired threshold value, C_{given}. This can be done efficiently using a classical optimization technique such as binary search. That is, one first checks if cost is below the threshold for $r/2$. Depending on the result, one then checks if cost is below the threshold for either $r/4$ or $3r/4$ and so on ... Once the correct value of r is found, if equality of the bound is desired, one can use the first $r + 1$ processors and increase the finish time so that C_{total} just equals C_{given}.

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 Sections 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, we investigate the use of this methodology to ascertain whether performance can be improved by replacing one fast but expensive processor with a number of cheap but slow processors.

TABLE 1
PROCESSING FINISH TIME AND TOTAL COMPUTING COST FOR SIX CASES OF SEQUENCES

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

5.1 Optimal Sequencing

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 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.

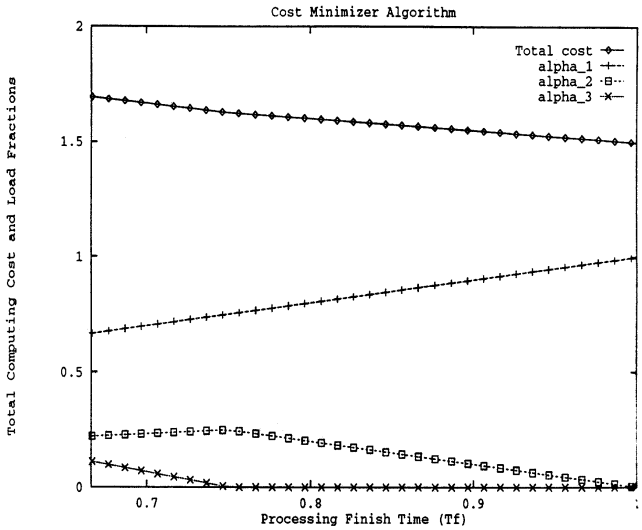


Fig. 4. The total computing cost and αs, according to the cost minimizer algorithm.

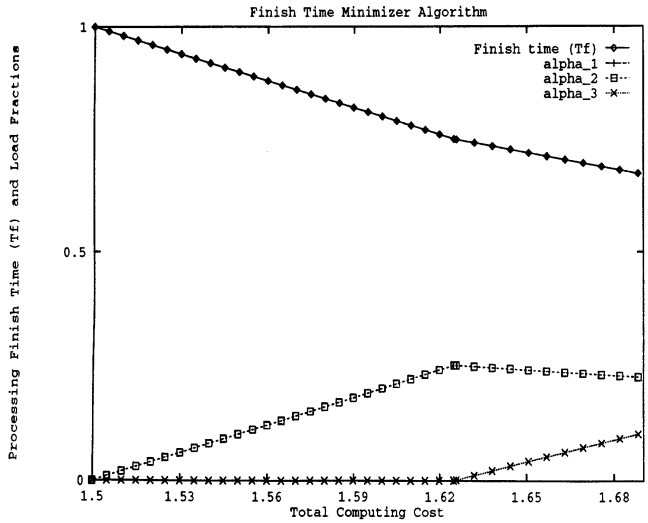
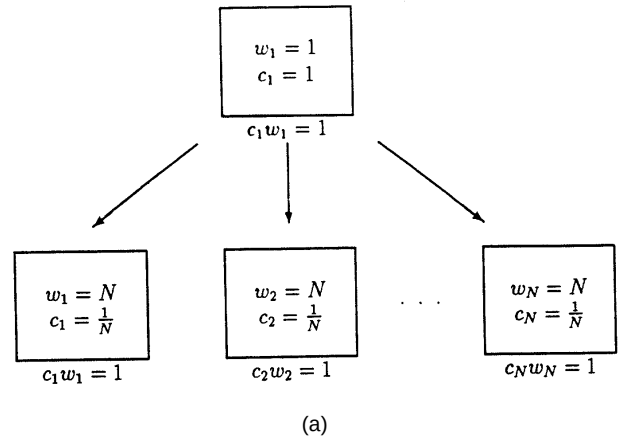


Fig. 5. The processing finish time and αs, according to the finish time minimizer algorithm.

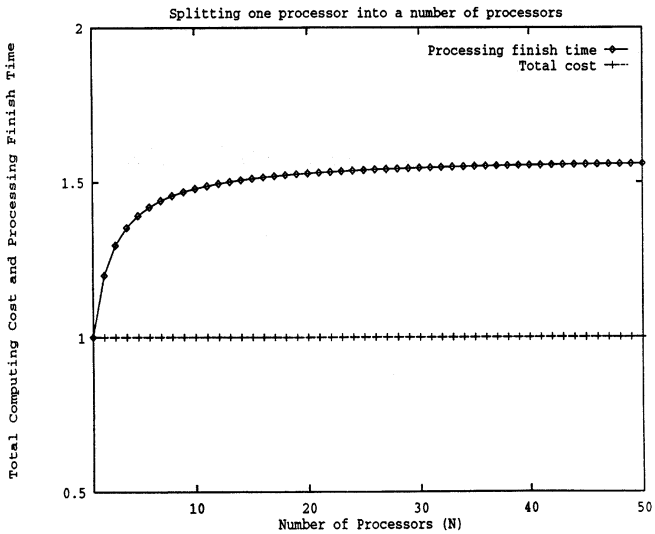
In Fig. 5, processing finish time and the processor load fractions are plotted versus the total computing cost for finish time minimization with an upper bound on total computing cost (using an algorithm in [38] producing equivalent results to those of the algorithm in this paper). In the diagram, $T_f = \alpha_1$ for this parameterization ($w_1 = T_{cp} = 1$) from (11). Also, in this diagram, a simple minimal finish time solution, without regard to cost, occurs at $C_{total} = 1.689$ and $T_f = 0.674$. Cost can be reduced to a minimal value of $C_{total} = 1.5$ with a corresponding increase in finish time at $T_f = 1.0$ and $\alpha_1 = 1.0$. That is, only processor 1, the most cost effective of the processors in terms of $c_i w_i$, is used.

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. The methodology developed here can be used as a tool to answer questions of this type. Three example cases will be presented in this regard. For all cases, $Z = T_{cm} = T_{cp} = 1$.



(a)

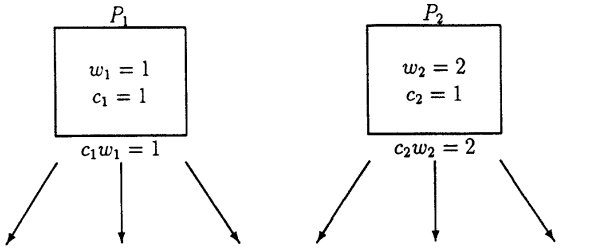


(b)

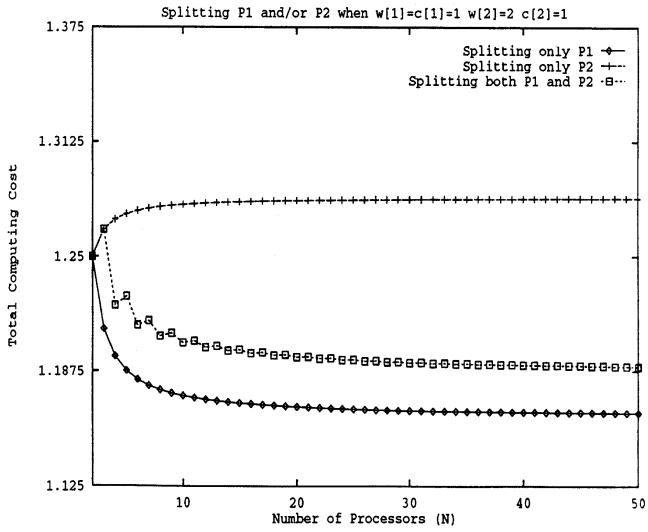
Fig. 6. (a) Replacing one processor with N cost equivalent processors. (b) The total computing cost and processing finish time against the number of splits (processors).

Fig. 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

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

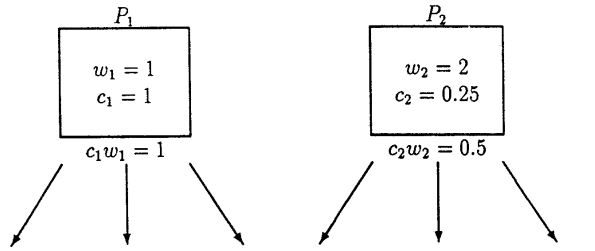


(a)

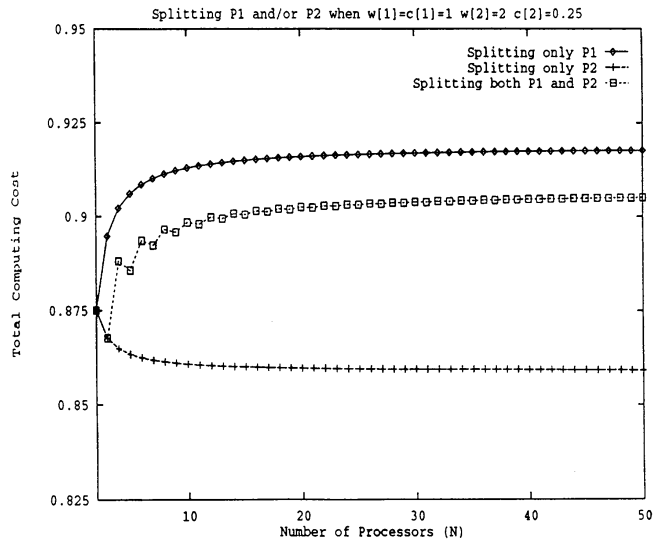


(b)

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.



(a)



(b)

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.

since $\alpha_1 = c_1w_1 = T_{cp} = 1$. 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 $c_nw_n = T_{cp} = 1$ 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. 6b. 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 (cost conservatively) split for better performance? Two examples are presented in Figs. 7 and 8 to demonstrate the types of comparisons that can be performed. For all cases, $w_1 = c_1 = 1$ and $w_2 = 2$. Note, in both 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 a case where P_2 is twice as expensive in cost per load compared to P_1 (though P_1 is faster). This choice of parameters is included to demonstrate the flexibility of the cost modeling. As shown in Fig. 7b, 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 behavior is as follows: In this example, cost is inversely proportional to computing speed (as $c_1 = c_2 = 1.0$). The load allocation equations, which are only based on finish time, 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 Fig. 7. Note that, here, the slower processor is also the less expensive one. 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.

6 Conclusions

In this paper, the optimal load distribution sequencing of bus network processors processing a divisible load was examined. It was found that a minimal total computing cost

can be obtained by arranging the sequence of processors such that the cheaper processors receive the workload earlier than the more expensive processors do. Here, cost is defined in terms of cost per load, $c_n w_n$. But, two factors, the processing finish time and the total computing cost, were found to involve a trade-off between each other. Algorithms to optimize such trade-offs were presented.

It was demonstrated that one can investigate questions of the cost and performance of different configurations of processors. This includes questions of whether small numbers of high performance/cost processors or large numbers of lower performance/cost processors are a better architectural choice.

Most importantly, this work demonstrates the tractability of solving cost oriented models using divisible load analysis. Divisible load analysis is, thus, a viable tool for the design of algorithms and policies for future computer utilities.

APPENDIX

THEOREM 2. *For the network of this paper with a given set of inverse processor speeds, $S(w) = \{w_1, w_2, \ldots, w_N\}$, the processing finish time is minimized if*

$$w_1 = \min(w_1, w_2, \ldots, w_N). \tag{28}$$

PROOF. The same abuse of notation as in Theorem 1 is maintained. For the general N processors case, the first fraction of the workload is, from (9),

$$\alpha_1 = [1 + k_1 + k_1k_2 + \cdots + (k_1k_2 \cdots k_{N-1})]^{-1}, \tag{29}$$

where

$$k_n = \frac{w_n T_{cp}}{ZT_{cm} + w_{n+1}T_{cp}} \qquad n = 1, 2, \ldots, N-1. \tag{30}$$

The processing finish time is now

$$\begin{aligned} T_f &= \alpha_1 w_1 T_{cp} \\ &= \frac{w_1 T_{cp}}{1 + k_1 + k_1k_2 + \cdots + \left(k_1k_2 \cdots k_{N-1}\right)} \\ &= \frac{w_1 T_{cp} \prod_{n=2}^{N} \left(ZT_{cm} + w_n T_{cp}\right)}{\sum_{n=1}^{N} \left[\prod_{i=1}^{n-1} \left(w_i T_{cp}\right) \cdot \prod_{i=n+1}^{N} \left(ZT_{cm} + w_i T_{cp}\right)\right]}. \end{aligned} \tag{31}$$

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. □

Note that a simplified proof for the $N = 3$ processor case appears in [38].

ACKNOWLEDGMENTS

We thank the anonymous reviewers for many helpful comments, and one reviewer in particular for suggesting the concept of the finish time minimizer algorithm of Section 4.2. We also wish to thank A. Zemanian for reading a portion of the manuscript and making a valuable suggestion. This work was performed while Jeeho Sohn was a PhD student at the State University of New York at Stony Brook.

REFERENCES

[1] I. Ahmad, A. Ghafoor, and G.C. Fox, "Hierarchical Scheduling of Dynamic Parallel Computations on Hypercube Multicomputers," *J. 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 Trans. 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 Trans. 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 Trans. 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," *J. 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," *Proc. 1990 Int'l Conf. Parallel Architectures*, pp. 128-135, 1990.

[7] K. Ramamritham, J.A. Stankovic, and P.-F. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 184-194, Apr. 1990.

[8] X. Qian and Q. Yang, "An Analytical Model for Load Balancing on Symmetric Multiprocessor Systems," *J. 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," *J. Parallel and Distributed Computing*, vol. 19, no. 1, pp. 38-50, Sept. 1993.

[10] K.G. Shin and M.-S. Chen, "On the Number of Acceptable Task Assignments in Distributed Computing Systems," *IEEE Trans. Computers*, vol. 39, no. 1, pp. 99-110, Jan. 1990.

[11] D.-T. Peng and K.G. Shin, "A New Performance Measure for Scheduling Independent Real-Time Tasks," *J. Parallel and Distributed Computing*, vol. 19, no. 1, pp. 11-26, Sept. 1993.

[12] G.C. Sih and E.A. Lee, "Declustering: A New Multiprocessor Scheduling Technique," *IEEE Trans. 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," *J. 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 Trans. Computers*, vol. 35, no. 5, pp. 389-398, May 1986.

[15] J. Du and J.Y.-T. Leung, "Complexity of Scheduling Parallel Task Systems," *SIAM J. Discrete Mathematics*, vol. 2, pp. 473-487, Nov. 1989.

[16] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Preemptive Scheduling Under Time and Resource Constraints," *IEEE Trans. Computers*, vol. 36, no. 8, pp. 949-960, Aug. 1987.

[17] Y.C. Cheng and T.G. Robertazzi, "Distributed Computation with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems*, vol. 24, no. 6, pp. 700-712, Nov. 1988.

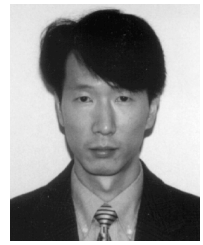
[18] Y.C. Cheng and T.G. Robertazzi, "Distributed Computation for a Tree Network with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems*, vol. 26, no. 3, pp. 511-516, May 1990.

[19] S. Bataineh and T.G. Robertazzi, "Distributed Computation for a Bus Network with Communication Delays," *Proc. 1991 Conf. Information Sciences and Systems*, pp. 709-714, Johns Hopkins Univ., Baltimore, Md., Mar. 1991.

[20] S. Bataineh and T.G. Robertazzi, "Bus Oriented Load Sharing for a Network of Sensor Driven Processors," *IEEE Trans. Systems, Man and Cybernetics*, vol. 21, no. 5, pp. 1,202-1,205, Sept. 1991.

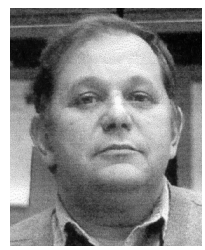
[21] S. Bataineh and T.G. Robertazzi, "Performance Limits for Processor Networks with Divisible Jobs," *IEEE Trans. Aerospace and Electronic Systems*, vol. 33, no. 4, pp. 1,189-1,198, Oct. 1997.

[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 Trans. Computers*, vol. 43, no. 10, pp. 1,184-1,196, Oct. 1994.
[23] T.G. Robertazzi, "Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors," *IEEE Trans. Aerospace and Electronic Systems*, vol. 29, no. 4, pp. 1,216-1,221, Oct. 1993.
[24] J. Sohn and T.G. Robertazzi, "Optimal Divisible Job Load Sharing for Bus Networks," *IEEE Trans. Aerospace and Electronic Systems*, vol. 32, no. 1, pp. 34-40, Jan. 1996.
[25] J. Sohn and T.G. Robertazzi, "A Multi-Job Load Sharing Strategy for Divisible Jobs on Bus Networks," Technical Report 697, State Univ. of New York at Stony Brook, College of Eng. and Applied Science, Aug. 1994. Also appears in chapter 12 of [45].
[26] J. Sohn and T.G. Robertazzi, "An Optimal Load Sharing Strategy for Divisible Jobs with Time-Varying Processor Speed and Channel Speed," Conf. version: *Proc. ISCA Int'l Conf. Parallel and Distributed Computing Systems*, pp. 27-32, Orlando Fla., Sept. 1995. Journal version: Accepted for *IEEE Trans. Aerospace and Electronic Systems*, July 1998.
[27] D. Ghose and V. Mani, "Distributed Computation in a Linear Network: Closed-form Solutions and Computational Techniques," *IEEE Trans. Aerospace and Electronic Systems*, vol. 30, no. 2, pp. 471-483, Apr. 1994.
[28] D. Ghose and V. Mani, "Distributed Computation with Communication Delays: Asymptotic Performance Analysis," *J. Parallel and Distributed Computing*, vol. 23, pp. 293-305, Nov. 1994.
[29] V. Bharadwaj, D. Ghose, and V. Mani, "Optimal Sequencing and Arrangement in Distributed Single-Level Tree Networks with Communication Delays," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 9, pp. 968-976, Sept. 1994.
[30] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Trans. Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 555-567, Apr. 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 Eng., Indian Inst. of Science, Bangalore, India, Dec. 1992.
[33] H.J. Kim, G.I. Jee, and J.G. Lee, "Optimal Load Distribution for Tree Network Processors," *IEEE Trans. Aerospace and Electronic Systems*, vol. 32, no. 2, pp. 607-612, Apr. 1996.
[34] J. Blazewicz and M. Drozdowski, "Scheduling Divisible Jobs on Hypercubes," *Parallel Computing*, vol. 21, pp. 1,945-1,956, 1995.
[35] J. Blazewicz 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.
[36] J. Blazewicz and M. Drozdowski, "Distributed Processing of Divisible Jobs with Communication Startup Costs," *Discrete Applied Mathematics*, vol. 76, issues 1-3, pp. 21-41, June 1997.
[37] E. Haddad, "Communication Protocol for Optimal Redistribution of Divisible Load in Distributed Real-Time Systems," *Proc. ISMM Int'l Conf. Intelligent Information Management Systems*, pp. 39-42, Washington, D.C., June 1994.
[38] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," State Univ. of New York at Stony Brook, College of Eng. and Applied Science, Technical Report 719, Oct. 30, 1995. Also related: US patent application, *Load Sharing Controller for Optimizing Monetary Cost*, 1996.
[39] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in Computer Networks: Motivation, Formulation and Example," *IEEE Trans. Networking*, vol. 1, no. 6, pp. 614-627, Dec. 1993.
[40] A. Faragó, S. Blaabjerg, L. Ast, G. Gordos, and T. Henk, "A New Degree of Freedom in ATM Network Dimensioning: Optimizing the Logical Configuration," *IEEE J. Selected Areas in Comm.*, vol. 13, no. 7, pp. 1,199-1,205, Sept. 1995.
[41] J.F. Kurose and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems," *IEEE Trans. Computers*, vol. 38, no. 5, pp. 705-717, May 1989.
[42] S.H. Low and P.P Varaiya, "A New Approach to Service Provisioning in ATM Networks," *IEEE Trans. Networking*, vol. 1, no. 5, pp. 547-553, Oct. 1993.
[43] Y.A. Korilis, A.A. Lazar, and A. Orda, "Architecting Noncooperative Networks," *IEEE J. Selected Areas in Comm.*, vol. 13, no. 7, pp. 1,241-1,251, Sept. 1995.
[44] D. Menasce and V. Almeida, "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures," *Proc. IEEE/ACM Supercomputing '90*, pp. 169-177, 1990.
[45] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, Calif.: IEEE CS Press, 1996.



Jeeho Sohn received the BS degree in electrical engineering from Yonsei University, Seoul, Korea, in 1987, and the MS degree from the University of Colorado at Boulder in 1989. He received the PhD degree in electrical engineering from the State University of New York at Stony Brook in 1995. He is currently a senior technical staff member at AT&T. His research interests include parallel and distributed computing, stochastic and queuing processes, and the performance evaluation of communication and computer systems.

Thomas G. Robertazzi received the PhD from Princeton University in 1981 and the BEE from Cooper Union in 1977. He is presently an associate professor of electrical engineering at the State University of New York at Stony Brook. From 1982-1983, he was an assistant professor in the Electrical Engineering Department of Manhattan College, Riverdale, New York. Prof. Robertazzi has served as an editor for books for the IEEE Communications Society and an associate editor of the journal *Wireless Networks*. He has authored one book, coauthored a second, and edited a third in the area of performance evaluation. Prof. Robertazzi is a senior member of the IEEE. His research interests involve the performance evaluation of computer and communication systems.



Serge Luryi received his PhD degree in physics from the University of Toronto in 1978. Between 1980 and 1994, he was a member of the technical staff at AT&T Bell Laboratories in Murray Hill, New Jersey. During this period, he served as a group supervisor in several device research departments, dealing with VLSI, quantum phenomena, and optoelectronics. He has published more than 140 papers and filed 28 U.S. patents in the areas of high-speed electronic and photonic devices, material science, and advanced packaging. In 1990, Bell Laboratories recognized him with the Distinguished Member of Technical Staff award. From 1986-1990, Dr. Luryi served on the editorial board of *IEEE Transactions on Electron Devices*, first as an associate editor and, then, the editor. In 1989, Dr. Luryi was elected a fellow of the IEEE "for contributions in the field of heterojunction devices" and, in 1993, a fellow of the American Physical Society "for theory of electron transport in low-dimensional systems and invention of novel electron devices." In 1994, Dr Luryi joined the faculty of the State University of New York at Stony Brook, where is currently a leading professor and chairman of the Department of Electrical Engineering.