Divisible Load Scheduling on Single-Level Tree Networks with Buffer Constraints

XIAOLIN LI, Member, IEEE

V. BHARADWAJ, Member, IEEE

C. C. KO, Senior Member, IEEE
The National University of Singapore

Scheduling a divisible load on a heterogeneous single-level tree network with processors having finite-size buffers is addressed. We first present the closed-form solutions for the case when the available buffer size at each site is assumed to be infinite. Then we analyze the case when these buffer sizes are of finite size. For the first time in the domain of DLT (*divisible load theory*) literature, the problem of scheduling with finite-size buffers is addressed. For this case, we present a novel algorithm, referred to as *incremental balancing strategy* (IBS), to obtain an optimal load distribution. Algorithm IBS adopts a strategy to feed the divisible load in a step-by-step incremental balancing fashion by taking advantage of the available closed-form solutions of the optimal scheduling for the case without buffer size constraints. Based on the rigorous mathematical analysis, a number of interesting and useful properties exhibited by the algorithm are proven. We present a very useful discussion on the implications of this problem on the effect of *sequencing* discussed in the literature [1]. Also, the impact of Rule A [2], a rule that obtains a reduced optimal network to achieve optimal processing time by eliminating a redundant set of processor-link pairs, is also discussed. Numerical examples are presented to ease understanding.

Manuscript received September 12, 1999; revised March 18 and April 28, 2000; released for production April 28, 2000.

IEEE Log No. T-AES/36/4/11375.

Refereeing of this contribution was handled by K-C. Chang.

Authors' address: Dept. of Electrical and Computer Engineering, The National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260, E-mail: ({engp8584,elebv,elekocc}@nus.edu.sg).

I. INTRODUCTION

Computing over distributed environments has attracted more and more research efforts, since it leverages the existing hardware and software and outweighs the supercomputer on the flexibility and performance-price ratio. One of the primary issues in the area of distributed computing is on how to divide and schedule loads in order to achieve the best performance in terms of finish time of the loads. According to the type of loads under processing, tasks scheduling problems can be classified into two categories, indivisible tasks scheduling [3] and divisible tasks scheduling [2]. The divisible load scheduling in multiprocessor systems, commonly referred to as *divisible load theory* (DLT) in the literature, is of recent origin [4]. Attracted by simple and elegant modeling methodology that DLT adopts, in recent years, there has been an increase in interest in this growing area of research. The applications of DLT can be found in the distributed intelligent sensor networks [4], image processing and lots of computationally intensive data processing areas [5]. As a new paradigm, DLT offers the ease of analytical tractability and produces a rich set of results in load scheduling in heterogeneous computing systems [2, 6, 7].

Since the origination of DLT, the research has spanned from addressing the general optimal scheduling problem on different network architectures to various scenarios with different practical constraints, such as time-varying channels [8], minimizing cost factors [9] and fault-tolerance requirements [10] etc. Excellent compilations of results until 1997 can be found in [2, 11]. A divisible load is a kind of load that can be divided arbitrarily into any number of smaller fractions and can be processed independently on different processors. DLT adopts a linear mathematical modeling [2] of the processor speed and channel speed parameters. In this model, the communication time (typically, the transmission time) over a channel is assumed to be proportional to the amount of load that is being transferred over any channel, and the computation time is proportional to the amount of load assigned to a processor. This model is based on a well-accepted and experimentally tested work [12] for practical systems. The primary concern in this research is to determine the optimal fractions of the load to be assigned to each processor such that the total processing time of the entire load is a minimum. To achieve this objective, many researchers propose many strategies to tackle the problems in different environments. The monograph [2] systematically shows a useful set of extensive research results. The closed-form solutions of the optimal processing time are derived for the linear daisy chain, bus networks and the tree networks

in [2]. The *principle of optimality* states that in order to achieve minimum processing time all the processors that are participating in the computation process shall stop processing at the same instant in time. The claim is rigorously proven for bus, single-level tree and linear daisy chain networks in the literature [7]. In [13], a revised model, including the overhead factors is proposed to describe the system in more realistic fashion. Also [14] extends this study, and derives closed-form solution for optimal processing time with this revised model. This work also derives the *necessary and sufficient* conditions for obtaining optimal processing time with a system of m-processors in a bus network. DLT was also extended to the domain of distributed multimedia document retrieval [15]. In recent years, one of the fast upcoming issues is in pricing the users using the network and its resources [16]. DLT addresses this issue by proposing a cost model to minimize the total monetary cost when utilizing the system resources [9, 17]. A recent study on arbitrary tree networks [19] presents a rigorous generalized analytical treatment in deriving optimal sequences when both the load distribution and results collection phases are considered.

In all the work so far in DLT, the available resources at the processing site are always assumed to be plentiful. For instance, the available buffer size to hold the divisible data to be processed at a site is assumed to be infinite. However, in practice, each site has a finite buffer capacity to hold the load. Also, in a distributed system, each processor may be engaged in its local processing or coordinating with other processors. In this process, it may share its available resources [18]. Because of multitasks running at each processor, it has to allocate a quota of its buffer to accommodate the new coming load fractions. Thus, we are naturally confronted with a problem of scheduling a divisible load under the finite-size buffer constraints. Thus, we consider here the above-mentioned scenario in which each of the processors in the single-level tree network has buffers of finite size. For the first time in the domain of DLT literature, the problem of scheduling with finite-size buffers is addressed. Relevant applications of this kind of scheduling problems can be found in the literature [3, 11]. We show a significant impact on the load scheduling strategies due to the limited buffer sizes.

We present the problem formulation in Section II. In Section III, we derive the closed-form solutions for the case when there is no buffer size limitation. Then, we propose an efficient algorithm incremental balancing strategy (IBS) to tackle the case when there are buffer size constraints. We then give a detailed proof for the optimality of our proposed algorithm in Section IV. In Section V, we present the discussions in detail. Section VI concludes the paper.

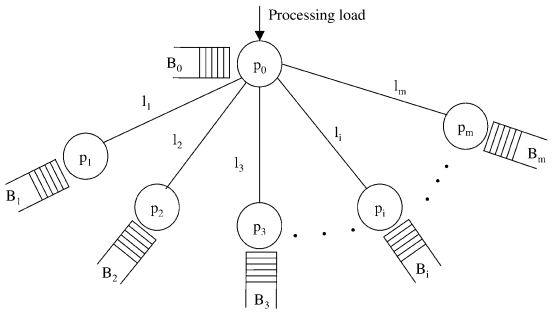


Fig. 1. Single-level tree network with finite-size buffers.

II. PROBLEM SETTING AND SOME PRELIMINARY REMARKS

The network model to be considered here consists of $(m + 1)$ processors interconnected through m links in a single-level tree fashion as shown in Fig. 1. We refer to the processor p_0 as the root node of the tree and other processors $p_1, p_2, \ldots, p_m$ as the leaf nodes. All the leaf nodes are connected to the root node via communication links, denoted as $l_1, l_2, \ldots, l_m$. Initially, the load arrives at p_0. The root node p_0 is assumed to have front-end, which means the communication process can be off- loaded to the front-end and p_0 can also take part in computation of the load. Certainly, a similar analysis can also be applied to the case when p_0 is not equipped with front-end. Further, we focus on the case when each p_i, $i = 1, 2, \ldots, m$ has finite-size buffers. To efficiently distribute the load with the buffer constraints, we have to devise a suitable strategy.

Thus, we are concerned about how to partition and schedule the load on to p_is with finite-size buffers such that the final processing time is minimum. To be specific, we describe the load scheduling process as follows. The divisible load L originates at the scheduler, i.e., the root node p_0. The root node p_0 divides the load into $(m + 1)$ load fractions, denoted as $\alpha_0, \alpha_1, \ldots, \alpha_m$, and distribute them among all the $(m + 1)$ processors. Upon receiving their respective load fractions, the processors buffer their respective load fractions and start processing it. The problem is then to determine the optimal sizes of these load fractions that are assigned to the processors such that the total processing time is minimum. The following notations and definitions are used.

L	The total amount of load originating at p_0 for processing.
α_i	Percentage of the total load assigned to processor p_i.
α	**Load distribution**, defined as an m-tuple $(\alpha_1, \ldots, \alpha_m)$ such that $0 < \alpha_i \leq 1$ and $\sum_{i=1}^{m} \alpha_i = 1$. The equation $\sum_{i=1}^{m} \alpha_i = 1$ is referred to as

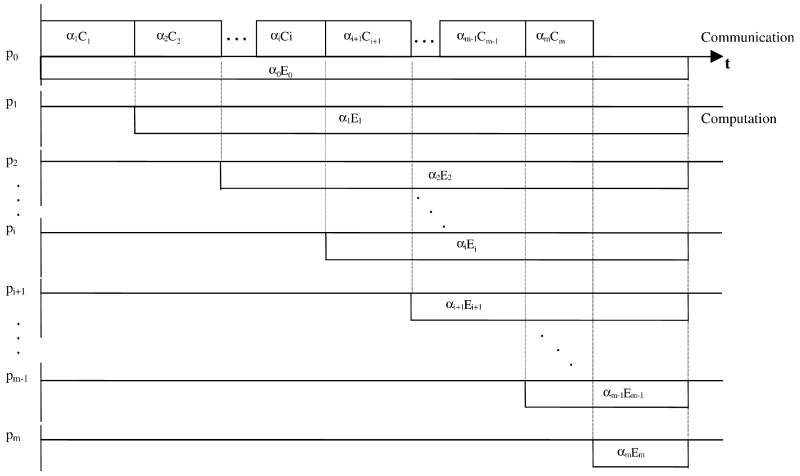


Fig. 2. Timing diagram: Single-level tree network.

normalization equation. Let us denote the space of all possible load distributions as Γ. Thus, $\alpha_i L$ is the actual amount of load assigned to p_i.

E_i The time taken to compute a unit load by the processor p_i.

C_i The time taken to transmit a unit load on the link l_i.

$T_i(\alpha,m)$ **Finish time of processor** p_i, is the time difference between the instant at which the ith processor stops computing and the time instant at which the BCU initiates the process.

$T(\alpha,m)$ **Processing time**, is the time at which the entire load is processed, i.e., $T(\alpha,m) = \max\{T_i(\alpha,m)\}$, $i = 1,2,\ldots,m$, where T_i is the finish time of processor p_i.

$T^*(\alpha^*,m)$ **Optimal processing time**, is the minimum processing time to finish processing the entire load, i.e., $T^*(\alpha^*,m) = \min_{\alpha\in\Gamma}\{T(\alpha,m)\}$.

Thus, using the above notations, we see that the communication time to transmit a fraction of load α_i to p_i is given by $\alpha_i C_i$, and the computation time of this load fraction by p_i is given by $\alpha_i E_i$.

In the literature [2], a condition referred to as Rule A for single-level tree networks was used to eliminate all the redundant processor-link pairs to obtain an optimal reduced network to achieve the optimal processing time. For details of the proof and derivation of this condition, the reader is referred to [1, 2]. Without loss of generality, in this paper we assume that Rule A [2] is satisfied for all the m processor-link pairs in the network. In the discussion section, we present a detailed analysis on the influence of Rule A in the case with finite-size buffer constraints.

III. OPTIMAL LOAD SCHEDULING

In this section, we first introduce the optimal load scheduling strategy with infinite-size buffers and derive the closed-form solutions for the optimal processing time. For the purpose of continuity and comparison, we present the recursive equations and the closed-form solutions for the case with infinite-size buffers first. Readers are referred to the details of the derivation in [1, 2]. The load distribution process by p_0 is illustrated by means of a timing diagram as shown in Fig. 2. After that, we propose our scheduling strategy to tackle the case with finite-size buffers.

A. Load Scheduling with Infinite-Size Buffers

From the timing diagram shown in Fig. 2, we obtain the following recursive equations

$$\alpha_i E_i = \alpha_{i+1}(E_{i+1} + C_{i+1}), \qquad i = 0,\ldots,m-1. \quad (1)$$

The normalizing equation is given by

$$\sum_{i=0}^{m} \alpha_i = 1. \quad (2)$$

From (1), expressing all the α_i, $i = 0,1,\ldots,m-1$, in terms of α_m, we have

$$\alpha_i = \left\{ \prod_{j=i+1}^{m} f_j \right\} \alpha_m \quad (3)$$

where

$$f_{i+1} = \frac{(E_{i+1} + C_{i+1})}{E_i}, \qquad i = 0,1,\ldots,m-1.$$

From (2), we obtain

$$\alpha_m = \frac{1}{\left(1 + \sum_{i=1}^{m} \prod_{j=1}^{m} f_j\right)}. \quad (4)$$

TABLE I
Results of Example 1

Iterations(k)	$B_0^{(k)}$	$B_1^{(k)}$	$B_2^{(k)}$	$B_3^{(k)}$	$B_4^{(k)}$	$B_5^{(k)}$	$Y^{(k)}$
0	35.000	15.000	20.000	25.000	10.000	15.000	0
1	5.000	0.000	12.500	21.250	8.125	14.063	59.063
2	0.000	0.000	10.000	20.000	7.500	13.75	9.687
3	0.000	0.000	0.000	15.000	5.000	12.500	18.750
4	0.000	0.000	0.000	7.857	1.429	10.714	12.500
ℓ_i^*	35.000	15.000	20.000	17.143	8.571	4.286	$L = 100.000$
$T_i(\alpha, m)$	35.000	30.000	55.000	69.286	69.286	69.286	$T(\alpha, m) = 69.286$

Thus, we have

$$\alpha_i = \frac{\left(\prod_{j=i+1}^{m} f_j\right)}{\left(1 + \sum_{i=1}^{m} \prod_{j=1}^{m} f_j\right)}, \qquad i = 0, 1, \ldots, m-1. \tag{5}$$

From Fig. 2, it can be seen that the processing time $T^*(\alpha^*, m)$ is the processing time of the root processor given by $\alpha_0 E_0$. Thus, $T^*(\alpha^*, m)$ is as follows:

$$T^*(\alpha^*, m) = \frac{\left(\prod_{j=1}^{m} f_j\right) E_0}{1 + \sum_{i=1}^{m} \prod_{j=1}^{m} f_j}. \tag{6}$$

The above closed-form solution is used frequently in the following section.

B. Load Scheduling with Finite-Size Buffers

In this section, we design and analyze a load distribution strategy when the buffer sizes at the processor sites are finite. We develop an algorithm to obtain an optimal load distribution and prove the optimality of the algorithm. We first introduce some notations, which are used throughout this work.

- $B_i^{(k)}$ — The buffer capacity available at p_i at the kth iteration, with $B_i^{(0)}$ as the initial buffer capacity.
- $S_{\text{vac}}^{(k)}$ — The set of processors whose buffers are left unfilled at the start of kth iteration. Thus, these are the processors available for processing at the kth iteration. Note that $S_{\text{vac}}^{(0)} = \{p_0, \ldots, p_m\}$.
- $S_{\text{filled}}^{(k)}$ — The set of processors whose buffers are filled at the start of kth iteration. Note that $S_{\text{filled}}^{(0)} = \emptyset$.
- S_{buf} — Set of all processors in the network, i.e., $S_{\text{buf}} = \{p_0, \ldots, p_m\}$. Note that $S_{\text{buf}} = S_{\text{filled}}^{(k)} + S_{\text{vac}}^{(k)}$.
- $X^{(k)}$ — The amount of load left unprocessed at the start of the kth iteration. Initially, $X^{(0)} = L$.
- $Y^{(k)}$ — The load scheduled at the kth iteration. Initially, $Y^{(0)} = 0$.
- $\alpha_i^{*(k)}$ — Under the assumption that the buffer size is infinite, the optimal load fraction assigned to the processor p_i for all $p_i \in S_{\text{vac}}^{(k)}$ at the kth iteration.

Now we present an algorithm to determine the optimal processing time for this finite buffer size case. We refer to this algorithm as IBS, as the load scheduling is carried out in an incremental fashion until the entire load is processed. The algorithm is presented in its pseudocode form in Table II.

In essence, at every iteration, the algorithm tries to take advantage of the property that the optimal processing time is achieved when all the processors involved in the computation of the load stop computing at the same time instant for the case with infinite size buffers. Note that, at kth iteration, we consider all the $p_i \in S_{\text{vac}}^{(k)}$. The expression for the amount of load that is processed at the kth iteration, namely, $Y^{(k)} = \min\{B_i^{(k)}/\alpha_i^{*(k)}, \forall p_i \in S_{\text{vac}}^{(k)}\}$, guarantees that all the respective processors in $S_{\text{vac}}^{(k)}$ can accommodate the optimal load portion assigned to them. This means that these processors will stop processing at the same time instant. Besides, note that those buffers, whose size satisfies the right-hand side (RHS) of the expression $Y^{(k)}$ mentioned above, will be filled at this iteration. One of the strengths of this algorithm is in finite step convergence, and this is essentially due to the choice of $Y^{(k)}$. Later, we show that the algorithm takes at most $(m + 1)$ iterations to process the entire load. The feasibility and optimality of this IBS algorithm are proven in the next section. Before presenting the theoretical analysis, we consider a numerical example to gain an easy understanding of the algorithm.

EXAMPLE 1 Consider a single-level tree network with 6 processors and 5 links with the following speed parameters. Let $E_i = E = 1.0$, $i = 0, 1, \ldots, 4$; and let $C_i = C = 1.0$, $i = 1, \ldots, 4$. Also, let the available buffer size at each p_i, $i = 0, 1, \ldots, 4$ be, $B_0 = 35.000$, $B_1 = 15.000$, $B_2 = 20.000$, $B_3 = 25.000$, $B_4 = 10.000$, and $B_5 = 15.000$, respectively. The total load is assumed to be 100 units. The complete set of results is listed in Table I.

TABLE II
Algorithm IBS

Initial state:

$S_{filled}^{(0)} = \emptyset$, $S_{vac}^{(0)} = S_{buf}$, $X^{(0)} = L$, $Y^{(0)} = 0$, initial buffer sizes are $B_i^{(0)}$, $\forall i = 0, 1, ..., m$.

if($\sum_{i=0}^{m} B_i^{(0)} < L$) exit with error message "no feasible solution to process any load greater than $\sum_{j=0}^{m} B_j^{(0)}$"
if($\sum_{i=0}^{m} B_i^{(0)} = L$) the optimal load distribution is $\{\ell_i^*\}$, where $\ell_i^* = B_i^{(0)}$. Exit;

Set k=1.

Do {

Phase I: *Resetting the variables*
$X^{(k)} \leftarrow X^{(k-1)}$, $S_{filled}^{(k)} \leftarrow S_{filled}^{(k-1)}$, $S_{vac}^{(k)} \leftarrow S_{vac}^{(k-1)}$, $B_i^{(k)} \leftarrow B_i^{(k-1)}$.

Phase II: *Scheduling*
Under the assumption that all the processors p_i ($p_i \in S_{vac}^{(k)}$) have infinite buffers, using the closed-form solution (5) for these processors p_i ($p_i \in S_{vac}^{(k)}$), we obtain the optimal load fractions $\{\alpha_i^{*(k)}\}$, $p_i \in S_{vac}^{(k)}$. Note that, we set $\alpha_i^{*(k)} = 0$ when $p_i \notin S_{vac}^{(k)}$ at this iteration. Taking the buffer constraints into consideration, we determine $Y^{(k)} = min\{B_i^{(k)}/\alpha_i^{*(k)}, p_i \in S_{vac}^{(k)}\}$.

Phase III: *Update*
If($Y^{(k)} \geq X^{(k)}$)
{ let $Y^{(k)} \leftarrow X^{(k)}$, break.}
else
{
Load left unprocessed is $X^{(k)} \leftarrow X^{(k)} - Y^{(k)}$.
For(for all $p_i \in S_{vac}^{(k)}$)
{ if $B_i^{(k)} = \alpha_i^{*(i)} * Y^{(k)}$, then $S_{filled}^{(k)} \leftarrow S_{filled}^{(k)} \cup \{p_i\}$ }.
$S_{vac}^{(k)} \leftarrow S_{buf} \backslash S_{filled}^{(k)}$.
For(for all $p_i \in S_{vac}^{(k)}$) { $B_i^{(k)} = B_i^{(k)} - \alpha_i^{*(k)} * Y^{(k)}$ }
}

Set $k = k + 1$
}while($k \leq m$)

Optimal Solution:

Total number of iterations is $r = k$;
The optimal schedule is given by { ℓ_i^*}, where ℓ_i^* is the optimal load portion assigned to p_i, $\ell_i^* = \sum_{k=1}^{r} (\alpha_i^{*(k)} * Y^{(k)})$, $i = 0, 1, ..., m$.

From the results shown in Table I, we see that the optimal load fractions are $\ell_0^* = 35.000$, $\ell_1^* = 15.000$, $\ell_2^* = 20.000$, $\ell_3^* = 17.143$, $\ell_4^* = 8.571$, $\ell_5^* = 4.286$. Thus, the scheduler (the root processor) shall distribute these load fractions to the processors, respectively, to gain the minimum processing time, i.e., ℓ_0 to p_0, ℓ_1 to p_1, etc. Then, we obtain the optimal processing time $T^*(\alpha^*, m) = 69.286$. The timing diagram shown in Fig. 3(a) shows the communication of the load fractions by the root processor and the finish times of the processors. A number of interesting points can be observed at this juncture. First, we find that at every iteration (with the exception of the last iteration), there is at least one buffer that is completely filled and hence, is not available in the successive iterations. Also, it is to be noted that in general, even in the last iteration, the respective buffers may be filled. This is not a coincidence, but an intrinsic property of the IBS algorithm, which is shown in the next section.

Also, it may be observed that at each iteration, the amount of load that is accepted for scheduling is processed optimally using the *principle of optimality* stated in the literature [2, 11]. The algorithm utilizes the available buffer at a site to the fullest and best possible extent, as shown in Fig. 3(b). From Fig. 3(a), we see that in the last iteration, only the processors p_3 to p_5 alone participate in processing the load, as the available buffer spaces with the rest of the processors are already consumed in the previous iterations. Further, another interesting and useful observation is that the processors p_3 to p_5 stop processing at the same time, thus contributing to the optimal processing time. In Table I, the last column specifies the total amount of load consumed at every iteration by all the participating processors. Each row, on the other hand, specifies the amount of buffer that is left unfilled (available for the next iteration) at each site.

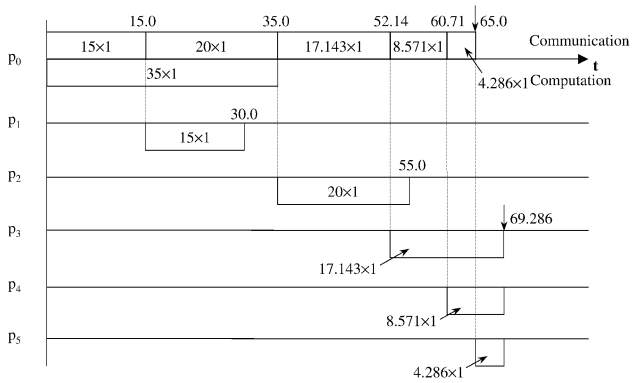


Fig. 3(a). Timing diagram for example 1.

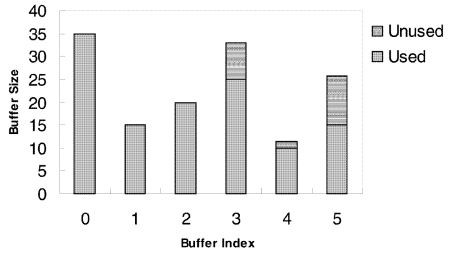


Fig. 3(b). Buffer utilization for example 1.

To appreciate the practical application of the proposed IBS algorithm, we now present one more example using some realistic system parameters.

EXAMPLE 2 Consider a single-level tree network with 6 processors and 5 links with the following speed parameters. Let E_0 = 16 s/Mb, E_1 = 114 s/Mb, E_2 = 20 s/Mb, E_3 = 351 s/Mb, E_4 = 16 s/Mb, E_5 = 20 s/Mb [20]. Note that, in [20], E_0 = 1.6 s/Mflops, however, we assume that 10 flop operations on each bit as an additional computational delay for the purpose of demonstration. The application could be an image processing or any other computationally intensive operations. As for the communication links, let C_1 = 0.1 s/Mb, C_2 = 0.65 s/Mb, C_3 = 0.15 s/Mb, C_4 = 0.1 s/Mb, C_5 = 0.15 s/Mb [20]. Also, let the available buffer size at each p_i, $i = 0,1,\ldots,5$ be, B_0 = 128 Mb, B_1 = 64 Mb, B_2 = 160 Mb, B_3 = 256 Mb, B_4 = 128 Mb, B_5 = 64 Mb, respectively. The total load is assumed to be 400 Mb.

Running our algorithm IBS, we obtain the following results after 2 iterations. The optimal load fractions assigned to each of the processors are ℓ_0^* = 115.080 Mb, ℓ_1^* = 16.137 Mb, ℓ_2^* = 89.088 Mb, ℓ_3^* = 5.074 Mb, ℓ_4^* = 110.621 Mb, ℓ_5^* = 64.000 Mb, respectively. The unused buffer size at each site is, B_0 = 12.920 Mb, B_1 = 47.863 Mb, B_2 = 70.912 Mb, B_3 = 250.926 Mb, B_4 = 17.379 Mb, B_5 = 0 Mb, and the finish time of each processor is given by T_0 = 1841.278 s, T_1 = 1841.278 s, T_2 = 1841.278 s, T_3 = 1841.278 s, T_4 = 1841.278 s, T_5 = 1360.944 s, respectively. Thus, the optimal processing time is given by $T^*(\alpha^*, m)$ = 1841.278 s. It may be observed that there is only one buffer, B_5, which is fully consumed. Another interesting observation is that at every iteration, the processors p_1, p_2, p_3, p_4 stop processing at the same time until all the load is processed, as they all have unused buffer spaces throughout the entire load distribution process, i.e., in all the iterations.

IV. ANALYSIS OF THE ALGORITHM

In this section, to realize the optimality of the solution generated by the algorithm IBS, we present rigorous proofs for the following important properties exhibited by the algorithm.

LEMMA 1 *At the end of kth iteration, if there is any excess load that is left unprocessed, i.e., if $X^{(k)} > 0$, then there is at least one buffer that is just filled at this kth iteration.*

PROOF From the IBS algorithm, we determine $Y^{(k)}$ by setting $Y^{(k)} = \min\{B_i^{(k)}/\alpha_i^{*(k)}, p_i \in S_{\text{vac}}^{(k)}$. There is at least one equality, without loss of generality, say, $Y^{(k)} = B_x^{*(k)}/\alpha_x^{*(k)}, p_x \in S_{\text{vac}}^{*(k)}$. Thus, when we assign the load portion $Y^{(k)}\alpha_x*(k)$ to p_x with the available buffer capacity $B_x^{(k)}$, the buffer of p_x will be filled exactly. Hence the proof.

This lemma signifies the fact that in every iteration, until all the load is processed, there is at least one buffer that will be filled completely. This means that at any iteration, if needed, the algorithm uses the buffers to the fullest possible extent. The numerical example in the previous section demonstrates this point too. We use it to prove Lemma 2.

LEMMA 2 *The total number of iterations (complexity of the algorithm) before the algorithm terminates, denoted as r, is no more than $(m + 1)$, $r \leq m + 1$.*

PROOF From Lemma 1, we know that there is at least one buffer filled at each iteration. In our system, we have totally $(m + 1)$ processors with $(m + 1)$ buffers. Under the constraint that the total load is less than the sum of all the buffer sizes, $L < \sum_{i=0}^{m} B_i$, the load will never completely fill all the buffers. Since at least one buffer is filled at each iteration, the total number of iterations required to obtain the final solution is no more than $(m + 1)$. Thus, the complexity of the algorithm is $O(m)$. Hence the proof.

Lemma 2 guarantees the feasibility aspect of the algorithm and derives the complexity of the proposed algorithm.

LEMMA 3 *At the end of any iteration k, all the processors p_i, $p_i \in S_{\text{vac}}^{(k)}$, stop processing at the same time.*

PROOF At the kth iteration, the time to process load $Y^{(k)}$ for all p_i, ($p_i \in S_{\text{vac}}^{(k)}$), is given by,

$$t^{*(k)} = t_i^{*(k)} = Y^{(k)} \left(\alpha_i^{*(k)} w_i T_{cp} + \sum_{j=1}^{k} \alpha_i^{*(k)} z_j T_{cm} \right). \tag{7}$$

If $p_0 \in S_{\text{vac}}^{(k)}$, $t_0^{*(k)} = Y^{(k)} \alpha_0^{*(k)} w_0 T_{cp}$. After kth iteration, the total load portion distributed to p_i ($i = 0,1,\ldots,m$) is given by

$$\ell_i^{*(k)} = \sum_{j=1}^{k} (\alpha_i^{*(j)} Y^{(j)}). \tag{8}$$

For p_i ($i = 1,2,\ldots,m$), the finish time is

$$T_i^{(k)} = \ell_i^{*(k)} w_i T_{cp} + \sum_{j=1}^{i} \ell_j^{*(k)} z_j T_{cm}. \tag{9}$$

For p_0, its processing time is given by, $T_0^{(k)} = \ell_0^{*(k)} w_0 T_{cp}$. We use induction to prove the lemma. At the first iteration, load distributed is $Y^{(1)}$. All the processors stop processing at the same time. The lemma holds. We assume after kth iteration, all the processors with vacancy stop processing at the same time $T^{(k)}$, which is given by

$$T^{(k)} = T_i^{(k)} = \ell_i^{*(k)} w_i T_{cp} + \sum_{j=1}^{i} \ell_j^{*(k)} z_j T_{cm}, \qquad p_i \in S_{\text{vac}}^{(k)}. \tag{10}$$

Now consider after $(k+1)$-th iteration, for $p_i \in S_{\text{vac}}^{(k+1)}$. Clearly, $p_i \in S_{\text{vac}}^{(k)}$

$$T_i^{(k+1)} = \ell_i^{*(k+1)} w_i T_{cp} + \sum_{j=1}^{i} \ell_j^{*(k+1)} z_j T_{cm} \tag{11}$$

$$= T_i^{(k)} + Y^{(k+1)} \alpha_i^{*(k+1)} w_i T_{cp} + \sum_{j=1}^{i} Y^{(k+1)} \alpha_j^{*(k+1)} z_j T_{cm} \tag{12}$$

$$= T^{(k)} + t^{*(k+1)}. \tag{13}$$

Thus, we see that for all processors p_i ($p_i \in S_{\text{vac}}^{(k)}$) their processing time is same and given by,

$$T^{(k+1)} = T^{(k)} + t^{*(k+1)}. \tag{14}$$

Since the lemma holds for kth and $(k+1)$th iteration, the lemma is proven true.

The numerical example in the last section shows this property at its last iteration. At the end of the last iteration, all the processors with buffer vacancy, i.e., p_3, p_4, p_5, stop processing at the same time. This is one of the important properties that the algorithm exhibits. At any iteration, say k, a set of processors whose buffers have vacancies, referred to as *vacant buffers*, are considered for scheduling the load $Y^{(k)}$. Since this total amount of load corresponds to the processor that receives minimum load (optimal load distribution) under no buffer constraints case, it is natural that its buffer gets filled at this iteration. Also, since we use the principle of optimality to balance the load among the processors at every iteration, all the participant processors stop at the same instant in time. Following lemma extends this case.

LEMMA 4 *At the end of any iteration k, the processing time of all $p_i \in S_{\text{vac}}^{(k)}$ is not less than the processing time of all $p_i \in S_{\text{filled}}^{(k)}$.*

PROOF From Lemma 3, we know that all the processors p_i, $p_i \in S_{\text{vac}}^{(k)}$ stop processing at the same time $T_{\text{vac}}^{(k)}$, and it is given by

$$T_{\text{vac}}^{(k)} = T_i^{(k)} = \ell_i^{*(k)} w_i T_{cp} + \sum_{j=1}^{i} \ell_j^{*(k)} z_j T_{cm}, \qquad \forall \quad p_i \in S_{\text{vac}}^{(k)}. \tag{15}$$

For processors p_j ($p_j \in S_{\text{filled}}^{(k)}$), there are two cases. In one case, if p_j is just filled in this kth iteration, its processing time is equal to $T_{\text{vac}}^{(k)}$. This is consistent with the equality in the lemma. In the other case, if p_j is filled before the kth iteration, we denote their processing time as $T_{\text{filled}}^{(k)}$.

Now we use induction to prove the lemma in the latter case. At the first iteration, all the processors stop processing at the same time. Further, we can check that at the second iteration $T_{\text{vac}}^{(2)} > T_{\text{filled}}^{(k)}$.

We assume that at kth iteration ($k > 1$) $T_{\text{vac}}^{(k)} > T_{\text{filled}}^{(k)}$. Then at $(k+1)$th iteration, from Lemma 3 we have

$$T_{\text{vac}}^{(k+1)} = T_{\text{vac}}^{(k)} + t^{*(k+1)} \tag{16}$$

where $t^{*(k+1)}$ can be obtained by replacing k in (7) with $k+1$.

Assuming that the closest processor with vacancy ordered before p_j, $p_j \in S_{\text{filled}}^{(k+1)}$, is p_x, $p_x \in S_{\text{vac}}^{(k+1)}$ and $x < j$, through the induction assumption and (7) we have

$$T_j^{(k+1)} = T_{j(\text{filled})}^{(k)} + \sum_{u=1}^{x} Y^{(k+1)} \alpha_u^{*(k+1)} z_u T_{cm} \tag{17}$$

$$< T_{\text{vac}}^{(k)} + \sum_{u=1}^{x} Y^{(k+1)} \alpha_u^{*(k+1)} z_u T_{cm} \tag{18}$$

$$< T_{\text{vac}}^{(k)} + t_x^{*(k+1)}. \tag{19}$$

From (7) and (16), we know $T_{\text{vac}}^{(k)} + t_x^{*(k+1)} = T_{\text{vac}}^{(k+1)}$. From (19), we obtain $T_{\text{filled}}^{(k+1)} < T_{\text{vac}}^{(k+1)}$, thus completing the induction. Hence the proof.

We use Lemma 3 and 4 to prove the following critical property.

LEMMA 5 *The algorithm guarantees that there is no load left unprocessed when it terminates. Besides, the total processing time is contributed by processors*

with vacant buffers. Furthermore, these processors stop processing at the same time instant.

PROOF From Lemma 2, the algorithm terminates in at most $(m + 1)$ iterations. By applying Lemma 3 and Lemma 4 to the final iteration, we see that the proof follows immediately. Hence the proof.

This lemma shows that the proposed algorithm adapts to the finite-size buffer constraints and obtains a schedule similar to the case without these constraints. This is a crucial point to obtain the optimal solution.

All the properties presented in the above lemmas lead to the following conjecture.

Conjecture. The IBS algorithm produces the optimal schedule.

The proof of the above claim is quite intuitive and evident from the construction of the algorithm and also from the properties shown through Lemmas 1–5. With just two processors, one can easily show the workings of the algorithm in a systematic fashion and obtain the optimal solution. However, for systems with three or more processors, a formal proof is yet to be attempted. This remains an open problem at this juncture.

V. DISCUSSIONS

In this paper, we devised a load distribution strategy to minimize the processing time of a divisible load among the processors in a single-level tree network with finite-size buffers. This strategy, referred to as IBS algorithm, works in an incremental fashion by optimally scheduling the load among the processors at every iteration. This algorithm uses the *principle of optimality*, proven in the literature [2] which states that in order to achieve the optimal processing time all the participating processors must stop computing at the same instant in time. Intuitively, it is true, as otherwise some processors may be idle while others keep busy. But in the case when the buffer size constraints exist, we cannot always let the processors involved in the computation stop processing at the same time, since some processors may not accommodate all the assigned load fractions decided by the optimal load distribution under infinite buffer assumption. Thus, obviously all the processors do not stop processing at the same time in this finite buffer case. To tackle this new problem, we propose an IBS algorithm, which is a judicious mutation of the previous strategies. IBS algorithm exploits the *principle of optimality* property to minimize the processing time at each iteration. After a finite number of iterations (as shown in Lemma 2), IBS algorithm will obtain the optimal load fractions to gain the minimum processing time using all the available processors. Now we present a numerical example that will elicit some interesting observations.

EXAMPLE 3 Given a single-level tree network with 5 processors, let the load to be processed be of length $L = 100$ units. We assume that p_0 is equipped with a front-end. The parameters of the system, the speeds of the processors, links and the available buffer capacities are $E_0 = 1.0$, $E_1 = 5.0$, $E_2 = 4.0$, $E_3 = 3.0$, $E_4 = 2.0$; $C_1 = 4.0$, $C_2 = 3.0$, $C_3 = 2.0$, $C_4 = 1.0$; the available buffer volume is $B_0 = 10.0$, $B_1 = 20.0$, $B_2 = 45.0$, $B_3 = 15.0$, and $B_4 = 30.0$, respectively. We distribute the load to the processors $p_0, p_1, \ldots, p_4$ in that order. We refer to this fixed sequence of load distribution as α_I. Using our IBS algorithm, we obtain the *optimal distribution* as $l_0 = 10.000$, $l_1 = 20.000$, $l_2 = 27.857$, $l_3 = 15.000$, $l_4 = 27.143$. The finish time of each processor is $T_0(\alpha_I, m) = 10.000$, $T_1(\alpha_I, m) = 180.000$, $T_2(\alpha_I, m) = 275.000$, $T_3(\alpha_I, m) = 238.571$, and $T_0(\alpha_I, m) = 275.000$, respectively. Thus, we have, the total processing time as $T(\alpha_I, m) = \max_{i=0}^{4} T_i(\alpha_I, m) = 275.000$.

We now consider the implications of Rule A [2] to this problem. Applying Rule A to the sequence α_I, we find that Rule A is violated for processor-link pairs (p_1, l_1) and (p_2, l_2). As per Rule A, we eliminate these two processor-link pairs to improve the performance. However, we cannot eliminate them all since the sum of the remaining buffer capacities cannot accommodate the total load of 100 units. Also, note that we cannot eliminate (p_2, l_2) pair as the total load is 100 units. If we try to eliminate processor-link pair (p_1, l_1), we obtain a reduced single-level tree network that has enough buffer capacities to accommodate the total load. Applying IBS algorithm to this reduced network, we obtain the total processing time as $T_{\text{reduced}} = 315.0$. Clearly, we see $T_{\text{reduced}} > T(\alpha_I, m)$. Now we will demonstrate a case where Rule A improves the time performance. Let $L = 50$ units. Without applying Rule A, using the IBS algorithm, we obtain the total processing time, $T'(\alpha_I, m) = \max_{i=0}^{4} T_i(\alpha_I, m) = 126.0$. In this case, we eliminate (p_1, l_1) and (p_2, l_2) pairs since these violate Rule A and when we eliminate these pairs, the sum of the remaining buffer capacities is enough to accommodate the total load. Applying IBS algorithm to the reduced network, we obtain the total processing time, $T'_{\text{reduced}} = 105.0$. Now, we see that $T'_{\text{reduced}} < T'(\alpha_I, m)$.

From the above two examples, we observe that the Rule A cannot be applied directly and may deteriorate the performance in some cases. Thus, the effect of Rule A could be detrimental as it does not account for the finite-size buffer constraints. At this juncture it remains to be explored in obtaining a revised version of the Rule A that includes these buffer constrains in order to achieve an optimal processing time for a given sequence using an optimal reduced network. It may be verified that in the first part of the above example, an exhaustive search shows that the optimal

solution for a given sequence α_I demands the use of all the processor-link pairs, which is what we have considered.

To see the effect of finite-size buffer constraints on the sequencing, we consider a sequence in which p_0 distributes the load to the child processors by following the order in which the link speeds decrease. In our example we first distribute to p_4 and then to p_3 and so on until p_1. We refer to this sequence as α_{II}. Using the IBS algorithm, with $L = 100$, we obtain the optimal load distribution as $l_0 = 10.000$, $l_4 = 30.000$, $l_3 = 15.000$, $l_2 = 31.154$, and $l_1 = 13.846$, respectively. The finish time of each processor is, respectively, $T_0(\alpha_{II},m) = 10.000$, $T_4(\alpha_{II},m) = 90.000$, $T_3(\alpha_{II},m) = 105.000$, $T_2(\alpha_{II},m) = 278.077$, $T_1(\alpha_{II},m) = 278.077$. Thus, the total processing time is $T(\alpha_{II},m) = \max_{i=0}^{4} T_i(\alpha_{II},m) = 287.077$. Note that $T(\alpha_I,m) < T(\alpha_{II},m)$. Observe that α_I has the distribution order in which the link speeds increase. In the literature [1, 2], it has been shown that the α_{II} is an optimal sequence when the buffer sizes are infinite in capacities. However, this example clearly shows that for a system of m-processors with finite-size buffer constraints, such a sequence need not be optimal. Thus, the finite size buffer constraints influence the overall processing time with respect to the choice of load distribution sequence. However, it may be noted that if $B_i \geq L$, $\forall i = 1,\ldots,m$, then the sequence α_{II} is the optimal sequence and the results in [1, 2] hold. In general, the *optimal sequence* with finite-size buffer constraints at the processing sites, remains *an open problem* at this juncture.

It may be noted that the algorithm is generic in its design and can be applied on the single-level tree network when the root processor is not equipped with a front-end. The difference lies in the computation of the optimal load distribution (α^*) [1, 2] at each iteration. All the results of Lemmas 1–5 hold for this case too. In addition to Rule A, here, we have Rule A$'$ [2] to be satisfied by the root processor. Again, the application of these rules has to be decided on the basis of the available buffer capacities. As pointed out in the DLT literature [2], all the results of the single-level tree network apply directly to the case of bus topology, and hence, the algorithm proposed here works for the bus network too. It may be noted that in this case, Rule A is automatically satisfied [2] and all the processors in the network participate in the load computation.

VI. CONCLUSIONS

The problem of scheduling a divisible load onto a set of processors with finite-size buffers in heterogeneous single-level tree networks was investigated. We presented the results of an optimal scheduling for the case when there is no buffer size constraint. Taking into consideration that the buffer size equipped on each processor is limited, we proposed a novel algorithm with time complexity $O(m)$, *IBS*, to achieve the optimal processing time. The algorithm was designed in such a way that at every iteration, the amount of load that was accepted for scheduling is optimally balanced among all the nodes of the network whose buffer capacities are adequate enough to accommodate the assigned load fraction. A number of interesting properties are elicited through lemmas to realize the optimality of the solution. An illustrative example is presented to demonstrate the workings of the algorithm in a step-by-step manner. The implications of Rule A [2] together with the buffer constraints considered in the problem are discussed. Also, the effect of sequencing is demonstrated using the buffer constraints. Although the optimality of the solution generated by the IBS algorithm is quite intuitive, a formal proof is yet to be attempted.

The extensions to our current work are of utmost importance to the domain DLT since the problem of scheduling with finite-size buffer constraints is very common in network-based computing systems. This is the first time in the domain of DLT that such a practical constraint is included in the analysis and an algorithm is designed. It would be interesting to tackle the problem on different architectures like hypercubes and meshes, which are of practical importance, with this constraint. We have exposed the difficulty of the problem in obtaining a single rule (see Example 3) that accounts for the speeds of the nodes and links together with the available buffer constraints in achieving the optimal solution. At this juncture, an interesting and a useful practical problem to address is to derive a condition similar to Rule A [2] (and Rule A$'$ for the without front-end case) that considers both the buffer constraints and the speed parameters of the network in identifying the redundant processor-link pairs. Also, it would be interesting to explore the effect of sequencing on the processing time and to determine an optimal sequence, if it exists.

REFERENCES

[1] Bharadwaj, V., Ghose, D., and Mani, V. (1994)
Optimal sequencing and arrangement in distributed single-level networks with communication delays.
IEEE Transactions on Parallel and Distributed Systems, **5** (1994).

[2] Bharadwaj, V., Ghose, D., Mani, V., and Robertazzi, T. G. (1996)
Scheduling Divisible Loads in Parallel and Distributed Systems.
Los Almitos, CA: IEEE Computer Society Press, 1996.

[3] Stankovic, J. A., Spuri, M., Ramamritham, K., and Buttazzo, G. C. (1998)
Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms.
Boston: Kluwer Academic Publishers, 1998.

[4] Cheng, Y. C., and Robertazzi, T. G. (1988)
Distributed computation with communication delays.
IEEE Transactions on Aerospace and Electronic Systems, **24** (1988), 700–712.

[5] Bharadwaj, V., Li, X., and Ko, C. C. (1999)
Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis.
In *Image and Vision Computing*.
New York: Elsevier, 1999.

[6] Eshaghian, M. M. (Ed.) (1996)
Heterogeneous Computing.
Boston: Artech House, 1996.

[7] Haddad, E. (1995)
Optimal load sharing in dynamically heterogeneous systems.
In *Proceedings of the Seventh Annual IEEE Symposium on Parallel and Distributed Processing*, San Antonio, TX, Oct. 1995.

[8] Sohn, J., and Robertazzi, T. G. (1998)
Optimal time-varying Load sharing divisible jobs.
IEEE Transactions on Aerospace and Electronic Systems, **34** (July 1998).

[9] Sohn, J., Robertazzi, T. G., and Luryi, S. (1998)
Optimizing computing costs using divisible load analysis.
IEEE Transaction on Parallel and Distributed Systems, **9**, 3 (Mar. 1998), 225–234.

[10] Bataineh, S., and Alibrahim, M. (1994)
Effect of fault tolerance and communication delay on response time in a multiprocessor system with a bus topology.
Computer Communications, **17** (Dec. 1994).

[11] Drozdowski, M. (1997)
Selected problems of scheduling tasks in multiprocessor computer systems.
Wydawnictwa Politechniki Poznanskiej (in English), **321**, Poznan, Poland, 1997.

[12] Lee, C., Wang, Y-F., and Yang, T. (1994)
Static global scheduling for optimal computer vision and image processing operations on distributed-memory multiprocesors.
TRC 94-23, University of California, Santa Barbara, Dec. 1994.

[13] Blazewicz, J., and Drozdowski, M. (1997)
Distributed processing of divisible jobs with communication startup costs.
Discrete Applied Mathematics, **76**, 1-3 (June 13, 1997).

[14] Bharadwaj, V., Li, X., and Ko, C. C. (1998)
On the influence of start-up costs in scheduling divisible loads on bus networks.
IEEE Transactions on Parallel and Distributed Systems (Oct. 1998).

[15] Bharadwaj, V., and Barlas, G. D. (1999)
Access time minimization for distributed multimedia applications.
In *Multimedia Tools and Applications*.
Boston: Kluwer Academic Press, 1999.

[16] Clark, D. D. (1996)
A model for cost allocation and pricing in the internet.
The Journal of Electronic Publishing, **2**, 1 (May 1996).

[17] Robertazzi, T. G., Luryi, S., and Sohn, J. (1999)
Load sharing controller for optimizing monetary cost.
U.S. Patent 5,889,989, Mar. 1999.

[18] Bertsekas, D. P., and Tsitsiklis, J. N. (1989)
Parallel and distributed computation: Numerical methods.
Englewood Cliffs, NJ: Prentice-Hall, 1989.

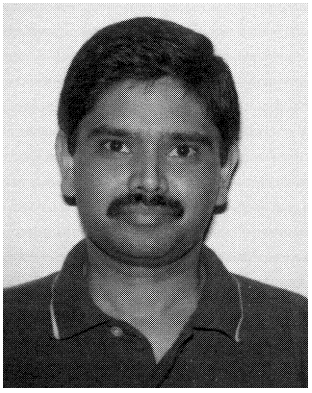
[19] Barlas, G. D. (1998)
Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees.
IEEE Transactions on Parallel and Distributed Systems, **9**, 5 (May 1998), 429–441.

[20] Gerasoulis, A., and Yang, T. (1993)
On the granularity and clustering of directed acyclic task graphs.
IEEE Transactions on Parallel and Distributed Systems, **4**, 6 (Jun. 1993), 66–701.

Xiaolin Li (M'98) received his B.Eng. in mechanical and electrical engineering from Qingdao University, PRC, 1995, M.Eng. in engineering design methodology from Zhejiang University, PRC, 1998.

At present, he is working towards his Ph.D. degree in the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests include *heterogeneous computing*, *load scheduling and balancing*, *image processing* and *intelligent CAD*.

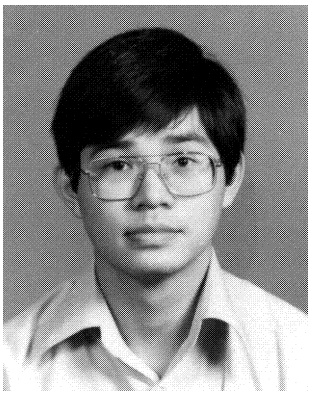
Mr. Li is a member of IEEE Computer Society.



Veeravalli Bharadwaj (M'98) received his BSc in Physics, from Madurai-Kamaraj Uiversity, India in 1987, Master's in electrical communication engineering from Indian Institute of Science, Bangalore, India in 1991 and Ph.D. from the Department of Aerospace Engineering, Indian Institute of Science, in 1994.

He is currently with the Department of Electrical and Computer Engineering in the Computer Engineering Division at National University of Singapore, Singapore, as an Assistant Professor. His research interests include *high speed heterogeneous computing*, *scheduling in parallel and distributed systems*, *design of VoD/MoD multimedia systems*, and *caching on the WWW*.

Dr. Bharadwaj is the principal author of the book *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, Aug. 1996. He is a member of IEEE Computer Society.



Chi Chung Ko (M'82—SM'93) received the B.Sc. (1st Class Honours) and Ph.D. degrees in electrical engineering from Loughborough University of Technology, U.K.

In 1982, he joined the Department of Electrical and Computer Engineering, National University of Singapore, where he is presently an Associate Professor and Deputy Head (Research) of the Department. His current research interests include digital signal processing, speech processing, adaptive arrays and computer networks.

Dr. Ko has written over 100 technical publications in these areas. He is an Associate Editor of the *IEEE Transactions on Signal Processing*.