*International Journal of Computers and Applications, Vol. 25, No. 2, 2003*

# DISTRIBUTED IMAGE PROCESSING ON A NETWORK OF WORKSTATIONS

X.L. Li,* B. Veeravalli,** and C.C. Ko***


## Abstract

In distributed computing systems, a critical concern is to efficiently partition and schedule the tasks among available processors in such a way that the overall processing time of the submitted tasks is at a minimum. On a network of workstations, using parallel virtual machine communication library, we conducted distributed image-processing experiments following two different scheduling and partitioning strategies. In this article, following the recently evolved paradigm, referred to as divisible load theory (DLT), we conducted an experimental study on the time performance to process a very large volume of image data on a network of workstations. This is the first time in the domain of DLT such an experimental investigation has been carried out. As a case study, we use edge detection using Sobel operator as an application to demonstrate the performance of the strategy proposed by DLT. Then, we present our program model and timing mechanism for the distributed image processing. Following our system models, we compare two different partitioning and scheduling strategies: the partitioning and scheduling strategy following divisible load-scheduling theory (PSSD) and the traditional equal-partitioning strategy (EQS). From the experimental results and performance analysis using different image sizes, kernel sizes, and number of workstations, we observe that the time performance using PSSD is much better than that obtained using EQS. We also demonstrate the speed-up achieved by these strategies. Furthermore, we observe that the theoretical analysis using DLT agrees with the experimental results quite well, which verifies the feasibility of DLT in practical applications.



## Key Words

Distributed image processing, divisible load theory, heterogeneous computing, parallel virtual machine, parallel programming


## 1. Introduction

With the proliferation of high-speed communication networks and powerful computers, network-based computing systems have proven to be an economical and efficient alternative choice to supercomputers [1, 2]. To efficiently utilize network-based computing resources, much research effort is devoted to designing the load/task scheduling and balancing strategies [3, 4].

A recently evolved paradigm, referred to as divisible load-scheduling theory (DLT) [3, 5, 6], proposed a simplified model to schedule and process a very large volume of data on a network of computers [3, 4]. The theory adopts a linear mathematical modelling of the processor speed and communication link speed parameters. In this model, the communication time over a channel is assumed to be proportional to the amount of load that is transferred over that channel, and the computation time is proportional to the amount of load assigned to that processor. Also, the processing load is assumed to be arbitrarily divisible in the sense that each partitioned portion of the load can be independently processed on any processor on the network. The primary concern in the research domain of DLT is to determine the optimal fractions of the total load to be assigned to the processors in such a way that the total processing time of the entire load is a minimum. Compilation of all the research contributions in DLT until 1995 can be found in [3, 4]. We will now present a brief survey of some of the significant contributions in this area.

### 1.1 Related Work

In [6], a heterogeneous linear network of processors was considered, and a computational algorithm was developed to obtain the optimal load fractions by assuming that *all the processors participating in the computation stop computing at the same time instant.* In fact, this has been shown to be a necessary and sufficient condition for obtaining optimal processing time in linear networks by using the concept of processor equivalence [7]. An analytic proof of this assumption in bus networks was presented in [8]. This condition is referred to as an *optimality principle* in the literature. In [9], closed-form solutions for optimal schedule for bus and tree networks were derived. Studies in [10] analysed the load distribution problem on a variety of computer networks such as linear, mesh, and hypercube. The concepts of *optimal sequencing* and *optimal network arrangement* were introduced in [5]. Barlas [11] investigated and proved the optimal sequencing in a single-level tree network by including the results collection phase. Load partitioning of intensive computations


* Center for Wireless Communications, National University of Singapore, Singapore, 119260; e-mail: cwclixl@nus.edu.sg

** Open Source Software Laboratory, National University of Singapore, Singapore, 119260; e-mail: elebv@nus.edu.sg

*** Department of Electrical and Computer Engineering, National University of Singapore, Singapore, 119260; e-mail: elekocc@nus.edu.sg


*(paper no. 202-1236)*

of large matrix-vector product in a multicast bus network was investigated in [12]. To find the ultimate speed-up using DLT analysis, Li [13] conducted the asymptotic analysis for various network topologies. With finite-size buffer constraint on each processor, [14] proposed an efficient incremental balancing algorithm to find an optimal schedule that satisfies with buffer constraint. Very recently, the results of DLT are shown to be applicable to the domain of network-based multimedia video/movie-on-demand systems. Here, the role of DLT is to recommend an optimal size of portions of a long-duration movie to be retrieved from a set of servers to meet the demand of a client. Thus, in this method, a pool of servers cooperate to serve a single request, so that the access time of the movie (start time instant of the playback) is minimized [15].

Because image-processing applications are computationally intensive and have immense potential for exploiting parallelism, a great deal of research is devoted to improving the performance of distributed and parallel image-processing techniques on a variety of multiprocessor environments, such as SIMD and MIMD [16]. In [17], the mapping of several image-processing algorithms, such as histogram, smoothing, and FFT, onto SIMD/MIMD systems was demonstrated. The authors proposed parallel processing schemes for these operations in detail under the assumption that the communication between adjacent processors can take place concurrently and all the processors are *identical*. However, the actual distribution of all the data among the processors was not considered in the analysis. Hence, equal partitioning becomes the natural choice for optimal performance. Parallel algorithms on other novel image-processing applications were also studied extensively in this work. An excellent analytical treatment on concurrent processing of various algorithms is given in [17]. In [18, 19], some strategies were presented to map a sequence of processing image operations onto a distributed memory system with different data partitioning and processor allocation policies for each task to be scheduled. As this involves a hybrid mix of strategies, the authors explicitly considered the overheads incurred, while redistributing the data between operations from one strategy to the other. However, the entire body of data was partitioned equally among all the processors.

### 1.2 Our Contribution

All research in DLT has so far focused on designing efficient load distribution strategies and evaluating their performance on a variety of network architectures. This work is the first of its kind to realize a practical implementation of a load distribution strategy on a network of workstations (NOWs). As an application domain, we identified the image-processing domain as a natural candidate to implement the strategy proposed by DLT. As low-level pixel-wise processing of an image for several application requirements, in general, requires a lot of CPU power, image partitioning, and computing on a network of computers is a natural solution to minimize the processing time, as above mentioned [17]. As image data are divisible in nature, they naturally fit in the DLT research domain. As mentioned before, DLT attempts to find the optimal fractions of image data to each processor. In most of the parallel and distributed image-processing literature [17–19], the image data are partitioned into equal proportions and distributed among the processors in the network. This is under the assumption that the communication delays are negligible, and hence we ignore the initial distribution delay of the image fractions to the respective processors. However, when the volume of the data is very large, especially when we consider satellite image pictures of size $3k \times 3k$, the amount of data that will be processed by a single processor will be significantly large, and transmitting such a large load fraction will incur a nonzero communication delay. These delays are significant when the network traffic is high and naturally affect the overall time performance. In DLT approach, the communication and computation delays are explicitly considered in the modelling, and the size of the proposed optimal load fractions will be decided according to these delay values. In the DLT literature, it has been proven that DLT produces better time and speed-up performance than equal partitioning. We shall verify these theoretical findings in our experiment.

As we observe, with NOWs, the message-passing paradigm is still the most popular choice [20]. As a *de facto* standard in message-passing within parallel and distributed computing, PVM survives as one of the best environments for parallel programming on NOWs [18], especially on heterogeneous distributed environment compared to MPI. In this article, we propose a novel partitioning and scheduling strategy using the DLT. Following the object-oriented programming paradigm, we developed a set of classes for distributed image processing on the basis of our software architecture. We use PVM package for our communication among the processors using message-passing paradigm. Distributed systems on which we conducted experiments are heterogeneous workstation clusters comprising HP, DEC, and SGI machines.

The organization of the article is as follows. Section 2 describes a simple image-processing operation, edge detection using Sobel detectors, as our experiment example. Section 3 presents our program model, the proposed image partitioning and scheduling strategies, and the traditional equal partitioning scheme. Section 4 details the performance analysis based on our experimental results. Section 5 concludes the article.

## 2. Edge Detection Algorithm

Edge detection is one of the main tools for image segmentation [21]. In this section, we only describe a basic edge detection algorithm using gradient operator as our application example. The gradient of an image $f(x, y)$ at location $(x, y)$ is the vector:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{1}$$

In edge detection, a common practice is to approximate the magnitude of this vector as:

$$\nabla f \approx |G_x| + |G_y| \tag{2}$$

A popular method to obtain the derivatives is to use Sobel operators, which have the advantage of providing both a differencing and a smoothing effect [21].

The derivatives based on the Sobel operator kernels are given by:

$$G_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{3}$$

$$G_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{4}$$

In practice, after obtaining the approximate magnitude $\nabla f$, we adjust its value into the range of grey level [0, 255]. The resulting output image is the convolution of the input image and Sobel edge detectors. Sobel edge detector can be easily extended to larger kernel size by outstretching the border of the smaller kernel. Although larger kernels are rarely used, we use a series of kernels ranging from $3 \times 3$ to $11 \times 11$ here only for the performance analysis of our experiment.

## 3. Distributed Image-Processing Strategies

As mentioned in Section 1, image data are the right candidate to exploit data parallelism because they are inherently *divisible* in nature. The only restrictive aspect while processing these data is when the individual processors demand some additional information from other processors to do processing on the data assigned to them [18]. For instance, consider an image of size $N \times N$ and let the kernel size be $K \times K$. Also, let the processor $p_i$ be assigned a set of rows of an image starting from $j$ to $j + r$, for some $j > 0,\ r > 0$ and $j + r \leq N$. Processor $p_i$ may need the $(j - \lfloor K/2 \rfloor)$th to $(j - 1)$th and $(j + r + 1)$th to $(j + r + \lfloor K/2 \rfloor)$th rows for it to perform computation on its assigned rows. To carry out this processing, in the literature it is standard practice to allow the processors to communicate among themselves for the respective data exchange. This data exchange takes place, concurrently, among the respective pairs of processors [17]. However, in the strategies to be proposed here (and adopted in [18]), we assign such additional data required by the individual processors right at the initial communication phase, before the actual load distribution to individual processors is carried out. Thereafter, the processors will perform computations on the respective portions by utilizing the additionally supplied data.

In this section, we first present our system set-up and program model. Then we introduce the traditional equal partitioning scheme (EQS) and our partitioning and scheduling strategy following DLT paradigm (PSSD).

### 3.1 Program Model

The system model we consider is a single-level tree network, shown in Fig. 1. We adopt the *master–slave* program

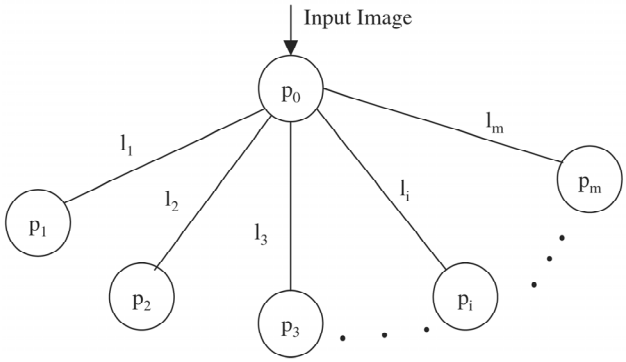
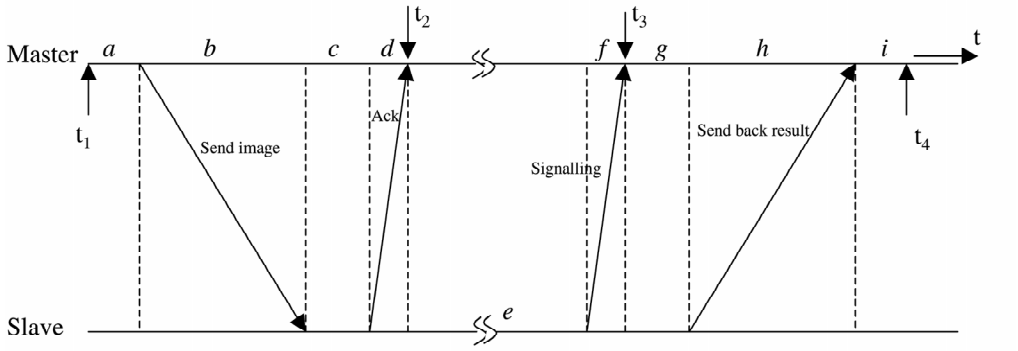



Figure 1. A single-level tree network.

model to design our software architecture. Furthermore, we *spawn* our slave processes on each workstation (one process per workstation). Thus, one slave process corresponds to one physical workstation. The input image is assumed to arrive at the root node $p_0$. The master process at the root processor is a central scheduler/controller that is in charge of optimal image partitioning and scheduling on the processors. Our scheduling strategy is implemented at the root node. The master process uses a predefined strategy to partition the image into $n$ portions and schedule these portions among the available processors, one to each processor. When the slave process receives the assigned image portion, it conducts the required operations and sends back the processed image to the master. When the master receives all the processed image portions, it combines them into the final image processed. The master processor (root) distributes the respective boundary pixels to each processor before the actual distribution of respective image portions to the respective processors. As we adopt the same method to supply the boundary pixels for two strategies to be described, they introduce the same amount of the extra communication overhead, and hence we do not include this overhead in our calculations.

To illustrate the communication and computation behaviour of the master and the slave processes, we present communication and computation timing mechanisms in Fig. 2.

Because we can neither guarantee the presence of a universal clock nor implement clock synchronization mechanisms, the clock at the master processor serves as a reference clock for all our timing measurements. Thus, the communication time shown in Fig. 2 is $(a+b+c+g+h+i)$. In practice, it seems that we cannot precisely obtain the timing measurement of phase $d$ and phase $f$, because all the timing measurements are carried out with respect to the master clock. In our experiment, we simplify the time epoch $(d+f)$ as the round trip time (RTT). Hence, we can easily obtain the total communication time by subtracting one of the components $d$ or $f$ from $(t_2 - t_1) + (t_4 - t_3)$, shown in Fig. 2. That is, the total communication time equals to $(t_2 - t_1) + (t_4 - t_3) - RTT/2$. We observe that the communication phase $(a, b, c)$ is symmetrical to the solution collection phase $(g, h, i)$. We will use this symmetrical nature in our scheduling strategy. On the other hand, the computation timing measurements are taken with respect




*a*: time to pack image on master; *b*: time to send the image from master to slave;
*c*: time to unpack image on slave; *d*: time to acknowledge the receipt of image on slave;
*e*: time to process image on slave; *f*: time to send the value of *e* from slave to master;
*g*: time to pack processed image on slave; *h*: time to send back the processed image to master;
*i*: time to unpack the processed image on master.

Figure 2. Computation and communication timings on master.

to the clocks in the individual slave processors. The computation time (denoted as $e$ in Fig. 2) has to be communicated to the master. Note that in these measurements, we are concerned with the computation and communication time intervals as opposed to the exact start and finish time instants at these processors.

We assume that the communication and computation time delay is proportional to the image size, which is observed in our experiment and was also adopted in [3, 4, 19, 22, 23]. Communication time delay in PVM consists of three components: latency time, packing/unpacking time, and transmission time [20]. As a common practice, they can be modelled as a linear function of message length [22, 23] as:

$$t_{cm} = CL + \theta \tag{5}$$

where $\theta$ is the latency time, $C$ is the summation of packing/unpacking, and transmission time per byte, and $L$ is the message length in bytes. As the component $\theta$ is negligibly small, we ignore it in our analysis. A study of the effect of overheads can be found in [23, 24].

To measure the processing speed of each node, we define a computation speed parameter $\gamma$ ms/byte for each computation step. The convolution of an $N \times N$ image using a $K \times K$ kernel requires $N^2K^2$ computation steps [18]. Thus, the total computation time is given by:

$$t_{cp} = N^2K^2\gamma \tag{6}$$

We denote $K^2\gamma$ as $E$ for simplifying the notations. Thus, as a computation speed parameter, $E$ denotes the time for $K^2$ computation steps. As shown in Fig. 1, we denote the computation speed of processor $p_i$ as $E_i$ and the communication speed of link $l_i$ as $C_i$, $i = 1, 2, \ldots, m$. In addition, we denote the image fraction assigned to the $i$th processor as $\alpha_i, i = 1, 2, \ldots, m$, where $\sum_1^m \alpha_i = 1$. The total image length is denoted as $L$ bytes, $L = N \times N$ for an $N \times N$ grey-level image. Then the processing time of this image portion at processor $p_i$ is $\alpha_i LE_i$ ms, and the communication time over link $l_i$ is $\alpha_i LC_i$ ms.

Fig. 3 illustrates a sketch of the master and slave processes in pseudo-code. Note that the function *schedule()* in the master process could generate a partition using either EQS or PSSD. In addition, the function *CImage:: Process()* in the slave process could conduct several image-processing operations, such as edge detection, template matching, or other general operations such as FFT or morphology.

### 3.2 Equal Partitioning Scheme

The data-partitioning and distribution strategy of EQS is as follows. Consider an image of size $N \times N$, and let $m$ be the number of processors in the system. In the literature [19], there are several equal partitioning schemes, such as row, column, block, and block-cyclic partition. For instance, in row partition, the $i$th row of the image data is allocated to processor $p_{R(i)}$, where $R(i) = \lfloor i/(N/m) \rfloor$, $i = 0, \ldots, N-1$.

These types of partitioning of the data yield optimal, if not the best possible, time performance only under the following situations [25]. First, the communication delays incurred are negligible. Second, homogeneous clusters of processors are to be utilized. When the communication delays are nonzero, equal partition will not generate the optimal processing time. To achieve optimal processing time, we deliberately choose the appropriate image fractions to each specific processor in the heterogeneous computing system. This will be demonstrated in our experiment. Further, we in the next section, relax all the above-mentioned assumptions and use a mathematical model that captures a realistic scenario to distribute the data.

Fig. 4 shows the timing behaviour of distributed image-processing using EQS. The processor $p_0$ is the master node. The timing stages on the master process consist of distribution phase, waiting phase, and collection phase. The timing stages on slave processes include image-receiving phase, image-processing phase, and result back-propagation phase. We notice that the time to send back the results of a processed image portion is the same as the transmission time of an original image portion.

```
(a) CImage image;
    image.Load(filename);
    //partition is an array, partition[i] corresponding to the
    //image portion assigned to the i-th processor
    partition=schedule(image);
    for(i=0; i<nhost; i++)
    {
        image.Pack(partition[i]);           //pack image data
        //send the image fraction to each slave
        //IP_EDGE indicates the image processing operation
        image.Send(stid[i], MSG_IMAGE, IP_EDGE);
    }
    for(i=0; i<nhost; i++)
    {
        //receive the processed image from each slave
        image.Recv(stid[i], MSG_RESULT);
        image.UnPack(partition[i]);
    }
    image.Save(processed_image);

(b) CImage image;
    image.Recv(ptid, MSG_IMAGE); //receive the assigned image from master
    image.UnPack();
    image.Process();
    image.Pack();
    image.Send(ptid, MSG_RESULT);
```

Figure 3. Pseudo-code for master and slave processes.

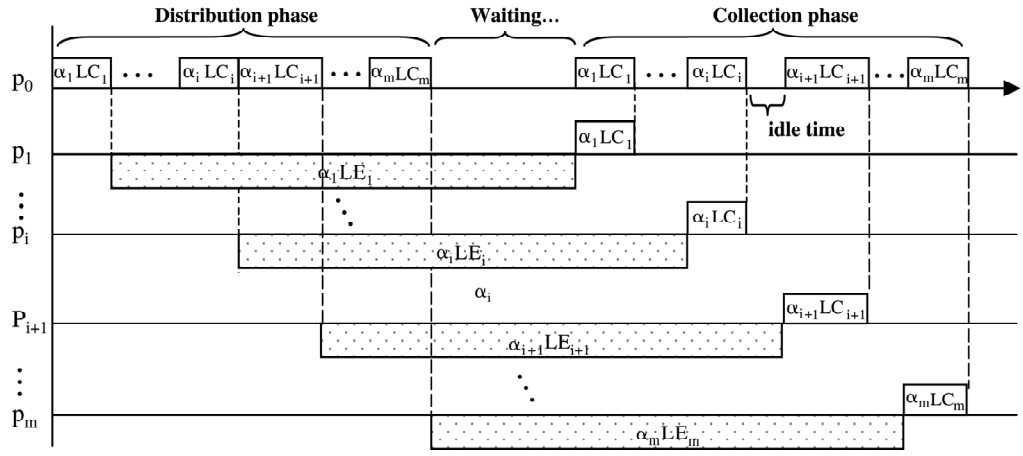



Figure 4. Timing diagram of EQS.

The final processing time using $m$ computers, denoted as $T(\alpha, m)$, is the time interval from the beginning of the entire load distribution to the end of the result collection. From Fig. 4, the final processing time of the total image is given by:

$$T(\alpha, m) = \max_{i=1}^{m} \left( \sum_{j=1}^{m} (\alpha_j L C_j) + \alpha_i L (E_i + C_i) \right) \qquad (7)$$

Using EQS, the total image load $L$ is equally partitioned into $m$ portions; thus $\alpha_i = 1/m$, $i = 1, 2, \ldots, m$. Only when $C_i = C$ and $E_i = E$ (i.e., the system is homogeneous), can the result-sending phase be continuous without idle times between the transmission of the processed image from two adjacent processors. However, in a heterogeneous system, this property will be violated, thus causing processor idling. An example is illustrated in Fig. 4, where processor $p_{i+1}$ has less computation power than $p_i$ (i.e., $E_{i+1} > E_i$), but

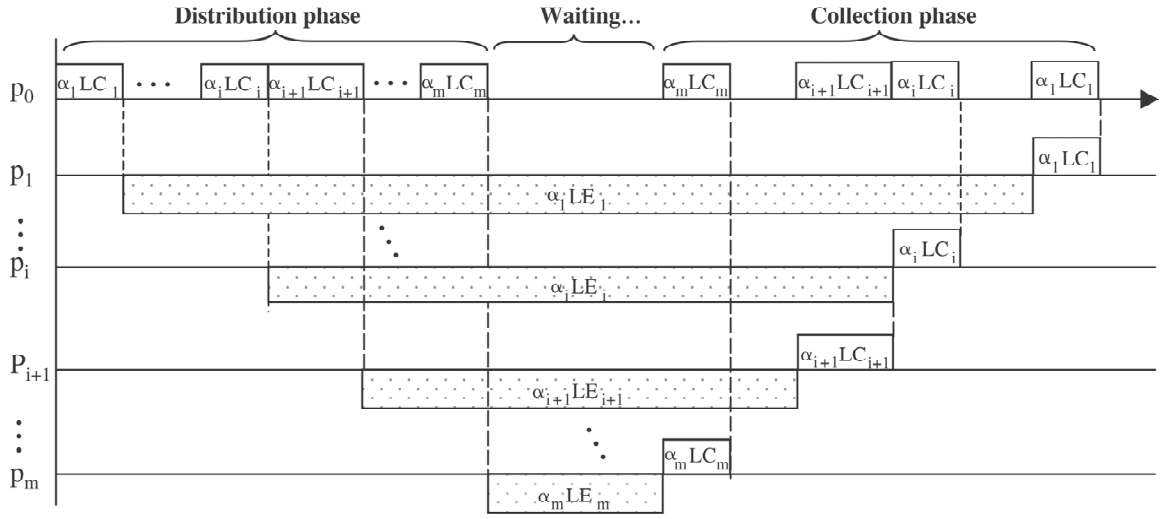



Figure 5. Timing diagram of PSSD.

they are assigned the same amount of image portions. We will show the disadvantage of EQS by comparing the experiment results in Section 4.

### 3.3 Partitioning and Scheduling Strategy Using DLT Paradigm

As mentioned in the previous section, the domain of image processing is a good candidate for applying DLT to obtain optimal partitioning of the input image. Our proposed strategy PSSD is as follows. As in the DLT literature [3, 4], it has been proven that for the case without results collection phase, *the optimal processing time is achieved when all the processors involved in the computation of the load stop computing at the same time.* Our strategy PSSD will follow this criterion, and the optimality of the scheme PSSD is proved in the literature [3]. In the case of distributed image processing with result collection phase, PSSD guarantees that there is no idle time gap between communication phases of the result collection, as illustrated in Fig. 5. From Fig. 5, we obtain the recursive equation:

$$\alpha_i L E_i = \alpha_{i+1} L(E_{i+1} + 2C_{i+1}) \quad i = 1, 2, \ldots, m-1 \tag{8}$$

We denote $f_{i+1} = E_i/(E_{i+1} + 2C_{i+1})$, $i = 1, 2, \ldots, m-1$. Expressing each of these load fractions in terms of $\alpha_1$, we obtain:

$$\alpha_i = \alpha_1 \prod_{j=2}^{i} f_j \quad i = 2, 3, \ldots, m \tag{9}$$

Using the normalization equation $\sum_{i=1}^{m} \alpha_i = 1$, we obtain $\alpha_1$ as:

$$\alpha_1 = \frac{1}{1 + \sum_{i=2}^{m} (\prod_{j=2}^{i} f_j)} \tag{10}$$

From Fig. 5, we obtain the closed-form solution of the optimal processing time using PSSD:

$$T(\alpha, m) = \alpha_1 L(2C_1 + E_1) \tag{11}$$

$$= \frac{L(2C_1 + E_1)}{1 + \sum_{i=2}^{m} (\prod_{j=2}^{i} f_j)} \tag{12}$$

Note that the solution of image fractions $\alpha_i L$ *bytes* or $\alpha_i N$ *rows* obtained from the above equations will be approximated into integers, where $L = N \times N$ for an $N \times N$ 256 grey-level image. This integer approximation will be applied to both the EQS and PSSD strategies.

## 4. Experimental Results and Discussions

To see the comparison between PSSD and EQS, we present a numerical example before we discuss the experimental results.

*Example 1.* Consider an input image of size $1,024 \times 1,024$ pixels, 256 grey scales, and kernel size of $3 \times 3$, to be processed on a single-level tree network with one scheduler and $m = 4$ workstations, as shown in Fig. 1. The speed parameters are $C_1 = 1$ ms/kbyte, $C_2 = 2$ ms/kbyte, $C_3 = 3$ ms/kbyte, $C_4 = 4$ ms/kbyte, and $E_1 = 10$ ms/kbyte, $E_2 = 30$ ms/kbyte, $E_3 = 20$ ms/kbyte, $E_4 = 10$ ms/kbyte. We use row-partition and distribute border image data *a priori* as mentioned before. Using EQS, we obtain the schedule as 256, 256, 256, 256 rows, respectively, on each processor, and the final processing time is $T^{(EQS)}(\alpha, m) = 10,752$ ms. Using PSSD, we obtain the schedule as 509, 150, 173, 192 rows, respectively, on each processor, and the final processing time is $T^{(PSSD)}(\alpha, m) = 6,118$ ms.

From Example 1, we see that the time performance using PSSD is far superior to that obtained using EQS. Without distinguishing the individual processor and link speeds, EQS naively distributes the input image equally

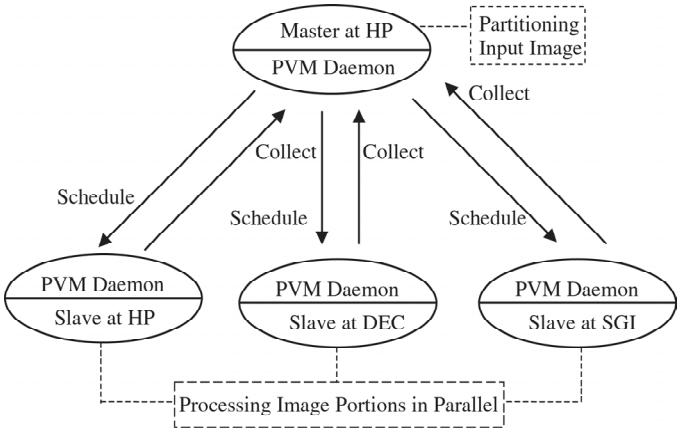



Figure 6. A four-node system using PVM.

among all the processors and results in a larger total processing time. However, in this heterogeneous computing environment, PSSD distributed the amount of the load in accordance with processor and link speed for the fixed distribution sequence. We observe that PSSD significantly improves the time performance, completing processing in about 56.9% of the finish time of the EQS strategy.

To validate our claim that PSSD scheme is better than EQS for distributed image processing, we used edge-detection algorithm as a case study. All the experiments were carried out using PVM communication library on an Ethernet network of HP, SGI, and DEC workstations. In the network of workstations, each workstation is a working node and the whole network performs as a distributed-memory MIMD computer. We run our master program on an HP workstation as the controller and spawn one slave process on each other workstation (HP or SGI). As shown in Fig. 6, the master process residing in the dedicated HP workstation works as the scheduler. One can also adopt a scheme that allows master and slave processes to reside in the same computer. Our cluster is constituted by 15 heterogeneous workstations (six HP-C200 running HPUX 10.2, four SGI O2 running IRIX 6.5, and five DEC AlphaStation500/500 running Digital UNIX V4.0D). Our master and slave programs are coded in C++ following the object-oriented programming (OOP) paradigm. Fig. 6 illustrates a small system set-up using PVM.

As mentioned in [26], in our testing we measure the speed parameters using the minimum time standard. In our heterogeneous distributed computing system, after a number of repetitive tests (typically 200 runs), the communication parameter $C_i$ ranges from $0.2\,\mu$s/byte to $0.5\,\mu$s/byte, and the computation parameter $E_i$ ranges from $10\,\mu$s/byte to $150\,\mu$s/byte, depending on image size, kernel size and specific operations.

To evaluate the performance of the above two strategies, we set a standard for comparison. As a common practice, we adopt the speed-up to compare the efficiency of EQS and PSSD strategies. The speed-up of our distributed image processing is defined as [27]:

$$S = \frac{\min_{i=1}^{m}(T_i^{(1)})}{T(\alpha, m)} \tag{13}$$

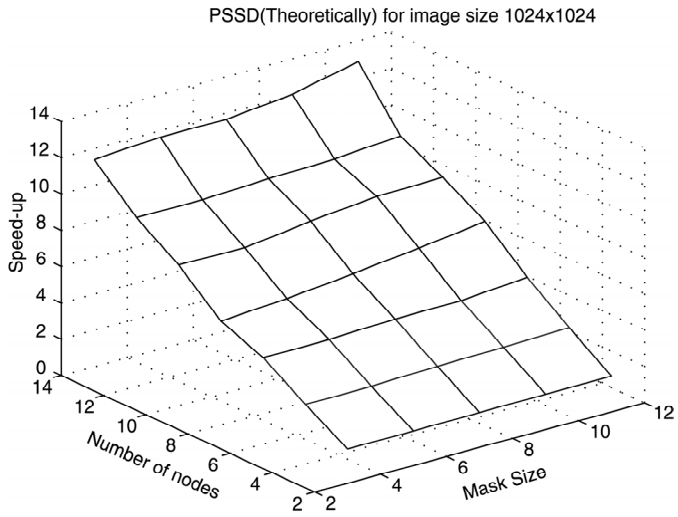



Figure 7. The theoretical speed-up w.r.t. number of workstations and kernel size using PSSD.

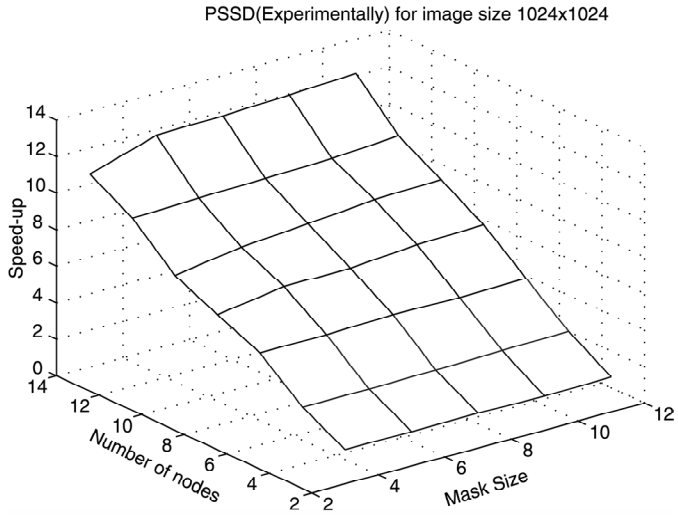



Figure 8. The experimental speed-up w.r.t. number of workstations and kernel size using PSSD.

where $T_i^{(1)}$ is the execution time of a sequential image-processing program on a single workstation $p_i, i = 1, 2, \ldots, m$; and $T(\alpha, m)$ is the final processing time of a distributed image-processing program on $m$ workstations as given by (7) for EQS or (12) for PSSD.

Using PVM communication primitives and timing mechanism as illustrated in Fig. 2, we experimentally determine the physical speed parameters $C_i$ and $E_i$ of the system. Following the derivations (9) and (10), we obtain the optimal schedules for each system configuration. Using (12) and (13), we obtain the behaviour of speed-up with respect to the number of workstations $m$ and the kernel size $k$ (theoretically) as illustrated in Fig. 7. Using our earlier analytical study on optimal schedule for PSSD strategy, in our experiments we partition and distribute the image among $m$ workstations and obtain a realistic time performance of the system. The experimental time performance behaviour is demonstrated in Fig. 8.

We see that the behaviours of speed-up under theoretical study correspond closely to the experimental results,

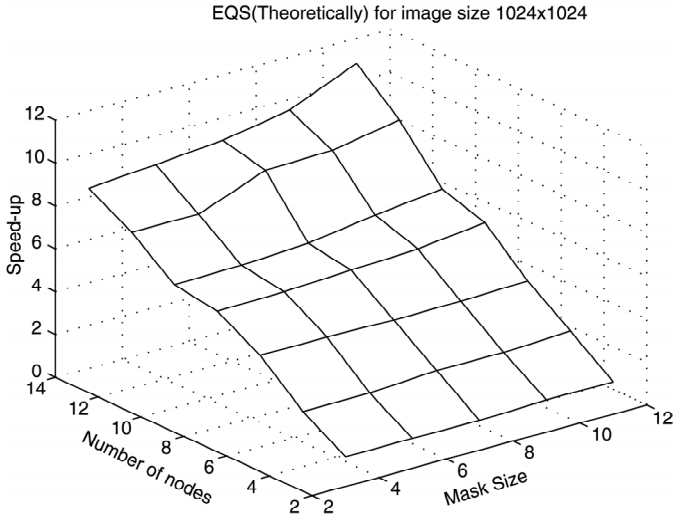



Figure 9. The theoretical speed-up w.r.t. number of workstations and kernel size using EQS.

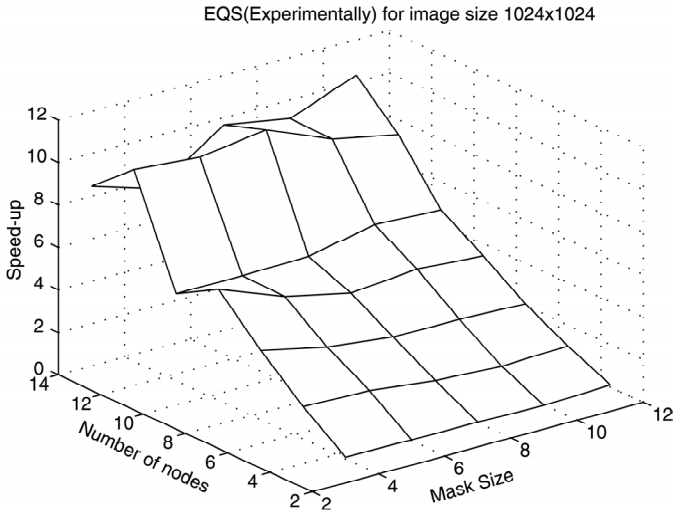



Figure 10. The experimental speed-up w.r.t. number of workstations and kernel size using EQS.

and their speed-up values are very close to each other for the same $m$ and $k$ values. From Figs. 7 and 8 we see that the speed-up factor increases greatly when we use more computing nodes. However, we observe that the speed-up is influenced more by increasing the computing nodes than just increasing kernel size. This phenomenon may result from the system latency in communication and computation aspects.

Figs. 9 and 10 demonstrate the behaviour of speed-up with respect to the number of workstations $m$ and the kernel size $k$ using EQS scheme. Here, too, we observe that the time performance using EQS scheme is inferior to that using PSSD. The theoretical analysis produces smooth results as in Fig. 9. However, the experimental figure has several abrupt jumps due to the fact that EQS is more sensitive to heterogeneous system architecture.

The comparison between the time performance for EQS and PSSD is more clearly demonstrated in Fig. 11. Fig. 11(a) shows the speed-up versus kernel size from $3 \times 3$ to $11 \times 11$ for a network configuration consisting

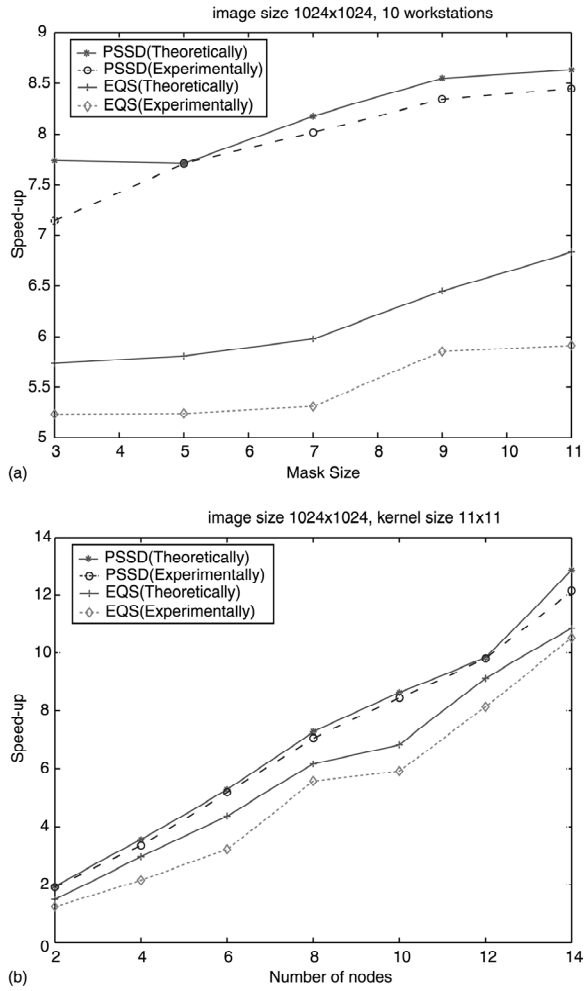



Figure 11. Comparison of speed-up between PSSD and EQS. (a) Speed-up versus kernel size; (b) spped-up versus number of nodes.

of 10 workstations to process a $1,024 \times 1,024$ image, and Fig. 11(b) shows the speed-up versus the number of nodes in processing a $1,024 \times 1,024$ image, with $11 \times 11$ kernel size. From Fig. 11 we see that the theoretical results agree quite well with the experimental results. An interesting observation drawn from this figure is that the speed-up increases as a function of kernel size. This is due to the increasing nature of the computational demand by the load, as the ratio of computation to communication time increases as kernel size increases.

### 4.1 Extensions to the Case of Finite-Size Buffers

As a final step, we extend our experimental study to consider the case when each processor in the system has a finite buffer as opposed to the assumption of infinite-size buffer in the literature. This problem is studied rigorously in [14], where an *incremental balancing strategy* (IBS) is presented. In practice, each processor may have a finite buffer capacity to hold the input load. Also, in a distributed system each processor may be engaged in its local processing and at the same time may accept the global tasks. Because of multitasks running at each processor, each 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 an input load under finite-size buffer constraints. The significant impact on the load-scheduling strategies due to limited buffer sizes has been investigated, and in our experiment we follow the IBS strategy proposed in [14]. With the inclusion of the solution collection phase, we tune the IBS algorithm to fit in our distributed image-processing scenario. It is worth mentioning that IBS algorithm also follows the underlying principle of DLT as PSSD. Now, we present an example in our experiment to demonstrate the impact of finite-size buffer constraint and the time performance of our strategies. The presentation of IBS algorithm and the detailed analysis of its properties can be found in [14].

*Example 2.* Consider an input image of size $1,024 \times 1,024$ pixels and 256 grey scales with a kernel size $11 \times 11$, to be processed on a single-level tree network with one scheduler and $m = 5$ workstations, as shown in Fig. 6. The speed parameters are $C_1 = 294.76\,\mu$s/kbyte, $C_2 = 328.97\,\mu$s/kbyte, $C_3 = 299.74\,\mu$s/kbyte, $C_4 = 326.97\,\mu$s/kbyte, $C_5 = 297.27\,\mu$s/kbyte, and $E_1 = 17,257.16\,\mu$s/kbyte, $E_2 = 37,702.04\,\mu$s/kbyte, $E_3 = 17,311.03\,\mu$s/kbyte, $E_4 = 32,109.57\,\mu$s/kbyte, $E_5 = 17,519.92\,\mu$s/kbyte. In addition, the respective buffer capacities are $B_1 = 200$ kbyte, $B_2 = 180$ kbyte, $B_3 = 300$ kbyte, $B_4 = 150$ kbyte, $B_5 = 300$ kbyte. Note that the above mentioned buffer capacities refer to the space needed to store the image data, and we store the kernel image data *a priori*. We use row-partition and distribute additional image data *a priori* as mentioned before. Following the IBS algorithm [14] with buffer constraints, we obtain an optimal portions of the image as 200, 133, 280, 148, 263 rows for processors $p_1$ to $p_5$, respectively. Using (12), we obtain a final processing time of $T^{(PSSD)}(\alpha, m) = 5.23415$ s. The time performance with our experiment, for this case, is observed to be within 10% of the theoretical value shown above.

Note that the resulting schedule following EQS violates the finite-size buffer constraint. The processor $p_4$ has only a 150 kbyte buffer; thus, it cannot accommodate the assigned load 205 kbyte and processor $p_3$ wastes its extra buffer resource. Further, for this case, without any buffer constraints, we obtain 271, 122, 256, 135, 240 as the optimal load distribution, and the final processing time is given by 4.83968 s.

## 5. Conclusion

The problem of divisible load scheduling for an image-processing application (edge detection) was theoretically and experimentally studied in this work, which is one of the first efforts to obtain experimental results in this area. A testbed consisting of high-speed workstations was used assuming a single-level tree topology. As per the theoretical suggestions, the speed parameters were fairly accurately measured and the load distribution was carried out. We have successfully verified all the theoretical findings, and this research highlights some interesting issues between the experimental and theoretical findings. In fact, this study allows researchers to decide on the possible assumptions in their theoretical framework that the experiment can offer and tolerate, such as system-related internal delays. With our experiments, we have conclusively shown the benefits of using the divisible load paradigm in a distributed computing network with high-speed machines and a link. With further use of a high-speed link, the delays experienced due to the data transmission become even small.

Our attempt in this article shows the plausibility of using DLT, and also demonstrates its ability to tap available computational power from computing resources in a clever manner. Thus, DLT serves as a natural choice even for other applications wherein the requirements for processing are computationally intensive. What remains to be decided lies in the choice of DLT for a particular application, even when an application facilitates a complete data parallelism approach. This is essentially due to a fundamental aspect of parallelism: the tie between the scalability of the problem and the size of the network (in terms of its available computing power) [16, 27]. Thus, the size of the load to be processed must be such that the available computational resources are not underutilized. In fact, the speed-up curves will naturally reflect this aspect in terms of a quick saturation with even fewer processors, and any attempt to utilize more processors will in reality result in only an incremental, if any, gain.

The conclusions drawn from this experimental investigation serve as a useful reference and provide a good basis for making an apt choice concerning the applicability of the results of DLT. We strongly believe that the impact of DLT will be significant for applications that demand CPU-intensive computations, and is not just restricted to the image-processing domain.

## References


[1] R. Buyya (Ed.), *High performance cluster computing*, vols. 1, 2 (Prentice Hall, 1999).

[2] M.M. Eshaghian (Ed.), *Heterogeneous computing* (Artech House, 1996).

[3] V. Bharadwaj, D. Ghose, V. Mani, & T.G. Robertazzi, *Scheduling divisible loads in parallel and distributed systems* (Los Almitos, CA: IEEE Computer Society Press, 1996).

[4] M. Drozdowski, *Selected problems of scheduling tasks in multiprocessor computer systems*, Monograph no. 321 (Poznan: Wydawnictwa Politechniki Poznanskiej, 1997).

[5] V. Bharadwaj, D. Ghose, & V. Mani, Optimal sequencing and arrangement in distributed single-level networks with communication delays, *IEEE Transactions on Parallel and Distributed Systems*, *5*, 1994, 968–976.

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

[7] T.G. Robertazzi, Processor equivalence for a linear daisy chain of load sharing processors, *IEEE Transactions on Aerospace and Electronic Systems*, *29*, 1993, 1216–1221.

[8] J. Sohn & T.G. Robertazzi, Optimal divisible job load sharing on bus networks, *IEEE Transactions on Aerospace and Electronic Systems*, *1*, 1996.

[9] S. Bataineh, T. Hsiung, & T.G. Robertazzi, Closed form solutions for bus and tree networks of processors load sharing a divisible job, *IEEE Transactions on Computers*, *43*(10), 1994.

[10] J. Blazewicz, M. Drozdowski, & M. Markiewicz, Divisible task scheduling—Concept and verification, *Parallel computing*, vol. 25 (Elsevier Science, 1999).

[11] G.D. Barlas, 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), 1998, 429–441.
[12] D. Ghose & H.-J. Kim, Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays, *Journal of Parallel and Distributed Computing*, *54*, 1998.
[13] K. Li, Managing divisible loads in partitionable networks, in J. Schaeffer & R. Unrau (Eds.), *High performance computing systems and applications* (Kluwer, 1998).
[14] X. Li, V. Bharadwaj, & C.C. Ko, Scheduling divisible tasks on heterogeneous single-level tree networks with finite-size buffers, *IEEE Transactions on Aerospace and Electronic Systems*, *36*, 2000, 1298–1308.
[15] V. Bharadwaj & G.D. Barlas, Access time minimization for distributed multimedia applications, *Special Issue in Multimedia Tools and Applications*, *12*(2/3), 2000, 235–256.
[16] K. Hwang & Z. Xu, *Scalable parallel computing: Technology, architecture, programming* (McGraw-Hill, 1998).
[17] H.J. Siegel, J.B. Armstrong, & D.W. Watson, Mapping computer-vision-related tasks onto reconfigurable parallel-processing systems, *IEEE Computer*, *25*(2), 1992, 54–63.
[18] C.K. Lee & M. Hamdi, Parallel image processing applications on a network of workstations, *Parallel Computing*, *21*, 1995, 137–160.
[19] C. Lee, Y.-F. Wang, & T. Yang, *Global optimization for mapping parallel image processing tasks on distributed-memory multiprocessors*, Report TRCS 96-28, University of California, Santa Barbara, CA, 1996.
[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, & V. Sunderam, *PVM: Parallel virtual machine: A users' guide and tutorial for networked parallel computing* (Cambridge: MIT Press, 1994).
[21] R.C. Gonzalez & R.E. Woods, *Digital image processing* (Addison-Wesley, 1992).
[22] A. Clematis & A. Corana, Performance analysis of task-based algorithms on heterogeneous systems with message passing, in *Lecture notes in computer science*, vol. 1497 (Springer-Verlag, 1998).
[23] B.K. Schmidt & V.S. Sunderam, Empirical analysis of overheads in cluster environments, *Concurrency, Practice and Experience*, *6*, 1994.
[24] V. Bharadwaj, X. Li, & C.C. Ko, On the influence of start-up costs in scheduling divisible loads on bus networks, *IEEE Transactions on Parallel and Distributed Systems*, *11*(12), 2000, 1288–1305.
[25] V. Bharadwaj, X. Li, & C.C. Ko, Efficient partitioning and scheduling of computer vision and image processing data on bus networks using divisible load analysis, *Image and Vision Computing*, *18*(11), 2000, 919–938.
[26] W. Gropp & E. Lusk, Reproducible measurements of MPI performance characteristics, *Proc. of the 6th European PVM/MPI Users' Group Meeting*, Spain, 1999.
[27] S. Sahni & V. Thanvantri, Performance metrics: Keeping the focus on runtime, *IEEE Parallel and Distributed Technology*, Spring, 1996, 43–56.

## Biographies

*Xiaolin Li* received his B.Eng. from Qingdao University, PRC, in 1995 and his M.Eng. from Zhejiang University, PRC, in 1998. At present, he is working towards his Ph.D. degree in the Department of Electrical and Computer Engineering, the National University of Singapore. He is currently working for the Center for Wireless Communications (CWC) at NUS as a member of the technical staff. His research interests include heterogeneous computing, load scheduling and balancing, image processing, and intelligent CAD. He is a member of IEEE Computer Society.

*Bharadwaj Veeravalli* received his B.Sc. in physics from Madurai-Kamaraj University, India, in 1987; his master's in electrical communication engineering from the Indian Institute of Science, Bangalore, India in 1991; and his Ph.D. from the Department of Aerospace Engineering, Indian Institute of Science, in 1994. He was a post-doctoral research fellow with the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently Assistant Professor at the Department of Electrical and Computer Engineering, in the Computer Engineering Division, at the National University of Singapore. 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. He is the principal author of the book *Scheduling Divisible Loads in Parallel and Distributed Systems* (Los Alamitos, CA: IEEE Computer Society, 1996) and is a member of IEEE.

*Ko Chi Chung* received the B.Sc. (1st Class Honours) and Ph.D. degrees in electrical engineering from Loughborough University of Technology, UK. In 1982 he joined the Department of Electrical and Computer Engineering, the 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. He is a senior member of IEEE, has written over 100 technical publications in his areas of interest, and is an Associate Editor of *IEEE Transactions on Signal Processing.*