Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems

†Veeravalli Bharadwaj, ‡Debasish Ghose, and ⊗Thomas G. Robertazzi

August 11, 2003

†Open Source Software Laboratory,
Dept. of Electrical and Computer Engineering,
The National University of Singapore,
10 Kent Ridge Crescent, Singapore 119 260
E-mail: elebv@nus.edu.sg

‡Dept. of Aerospace Engineering,
Indian Institute of Science, Bangalore 560 012, India
E-mail: dghose@aero.iisc.ernet.in

⊗ Dept. of Electrical and Computer Engineering,
University at Stony Brook,
Stony Brook, N.Y. 11794 USA
Corresponding Author
Tel: (631) 632-8412 or 8400
Fax: (631) 632-8494
E-mail: tom@ece.sunysb.edu

Abstract

Divisible load theory is a methodology involving the linear and continuous modeling of partitionable computation and communication loads for parallel processing. It adequately represents an important class of problems with applications in parallel and distributed system scheduling, various types of data processing, scientific and engineering computation, and sensor networks. Solutions are surprisingly tractable. Research in this area over the past decade is described.

1 Introduction

The interest in network-based computing has grown considerably in recent times. In this environment, a number of workstations or computers are linked through a communication network to form a large loosely coupled distributed computing system. One of the major attributes of such a distributed system, apart from its role in storing information in a distributed manner and allowing the use of shared resources, is the capability that it offers to a user at any single node to exploit the considerable power of the complete network or a subset of it by partitioning and transferring its own processing load to the other processors in the network.

This paradigm of load distribution is basically concerned with a single large load which originates or arrives at one of the nodes in the network. The load is massive and requires an enormous amount of time to process given the computing capability of the node. The processor partitions the load into many fractions, keeps one of the fractions for itself to process and sends the rest to its neighbors (or other nodes in the network) for processing. An important problem here is to decide how to achieve a balance in the load distribution between processors so that the computation is completed in the shortest possible time. This balancing can be done at the beginning or dynamically as the computation progresses and the computational requirements become clearer. This framework of computing is suitable for applications that permit the partitioning of the processing load into smaller fractions to be processed independently so that the partial solutions can be consolidated to construct the complete solution to the problem. Obviously not all processing loads satisfy this requirement. But there are a large class of applications that not only permit this kind of processing, but for which it is essential to do so in order to complete the task in time.

By and large, scheduling problems discussed in the literature do not attempt to formulate scheduling policies based on the type of loads submitted by an user, except perhaps where resource constraints are involved. Usually, the stress has been on designing efficient parallel algorithms in place of conventional sequential algorithms, which requires exploitation of *function parallelism* in the algorithm. However, there is another kind of parallelism that occurs in the data and is called

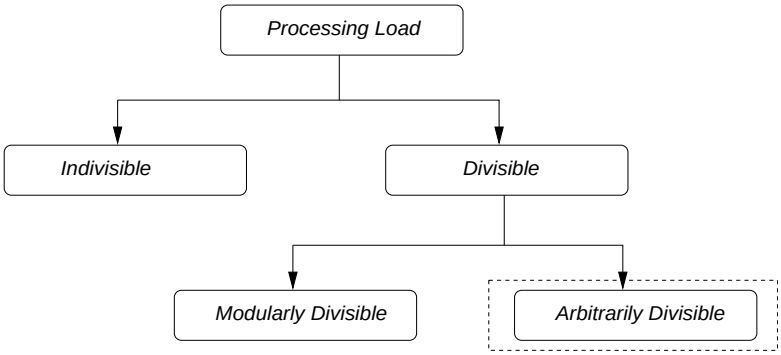

Figure 1: Classification of processing loads

data parallelism. Such loads can be split and assigned to many processors. But, the manner in which this partitioning (or load division) can be done depends on its *divisibility property*, that is, the property which determines whether a load can be decomposed into a set of smaller loads or not (see Figure 1).

Accordingly, loads may be *indivisible* in which case they are independent, of different sizes, and cannot be further subdivided. Thus, they have to be processed in their entirety in a single processor. These loads do not have any precedence relations and, in the context of static/deterministic scheduling, they give rise to *bin-packing* problems that are known to be NP-complete and hence amenable only to heuristic algorithms that yield sub-optimal solutions. In the context of dynamic/stochastic scheduling, these loads arrive at random time instants and have to be assigned to processing nodes based on the state of the system.

Alternatively, a load may be *modularly divisible* in which case it is *a priori* subdivided into smaller modules based on some characteristics of the load or the system. The processing of a load is complete when all its modules are processed. Further, the processing of these modules may be subject to precedence relations. Usually such loads are represented as task interaction graphs whose vertices correspond to the modules, and whose edges represent interaction between these modules and perhaps also the precedence relationships.

Finally, a load may be *arbitrarily divisible* which has the property that all elements in the load demand an identical type of processing. These loads have the characteristic that they can be arbitrarily partitioned into any number of load fractions. These load fractions may or may not have precedence relations. For example, in the case of Kalman filtering applications, the data is arbitrarily divisible but there may exist precedence relation among these data segments or load fractions. On the other hand, if the load fractions do not have precedence relations, then each load fraction can be independently processed. These latter type of loads are the ones which are of interest to us. Applications which satisfy this divisibility property include processing of massive experimental data, image processing applications like feature extraction and edge detection, and signal processing applications like extraction of signals buried in noise from multidimensional data collected over large spans of time, computation of Hough transforms, and matrix computations. Traditionally, the parallelism inherent in these problems was exploited through parallel algorithms. Now with the availability of distributed computing systems it is realized that these parallel algorithms can be mapped on to network based computing. However, this is not a straightforward mapping since many of the special properties and limitations of distributed systems affect the performance of the load distribution algorithms. One such factor is the communication delay which is considerably higher in a distributed network environment (that has distributed memory machines with message-passing architecture) than in a parallel processing environment (which has a shared memory architecture). This is especially true for the processing of divisible loads since there is very little interprocessor communication during the actual computation process.

2 A Historical Perspective

The initial application of divisible load processing was motivated by the objective of integrating communication and computation in distributed sensor network modeling. The basic goal was to understand "intelligent" sensor networks. Nodes in such a network could make measurements of the environment, perform computations on the measurement data and communicate the measurements to other nodes to gain the benefits of parallel processing. What proved most fortuitous was the generic nature of the original load model. The "job" (load) to be processed was envisioned as basically a very large linear data file that could be arbitrarily divided amongst a number of processors. The concern with measurement data (such as radar or sonar returns) led to the "divisible" nature of the load. Linear models were used to represent processing speeds and transmission times, making them dependent on the processor and communication hardware only to the extent of a linear constant. This is an important feature in an era of rapid changes in networking technology.

A simple parametrization also allowed one to specify the size of the load and whether it was more computation or communication intensive. Moreover, the overall model was purely deterministic, with none of the statistical assumptions that can be the Achille's heel of stochastic models. What was not envisioned originally was that this generic load sharing model would prove so simple to solve. In fact the reason for this simplicity, the linearity of the model, took some time to be fully appreciated. But what it did mean was that many of the analytically powerful concepts of linear theory used in research on electric circuits, queueing models and control theory, could be applied to divisible load analysis.

The generic nature of the divisible load model also meant that it was eventually realized that the theory would apply to parallel and distributed computer system modeling as well as to sensor network modeling. Thus, in the history of the literature on divisible load theory, early publications tended to appear in such journals as the IEEE Transactions on Aerospace and Electronic Systems and later ones often in computer engineering journals. In the past few years the literature on divisible loads has been enriched by many significant contributions by researchers from several countries. A book that covers the early results in this area of research is [1]. Another book that covers several results in the perspective of various network topologies is [2]. A compendium of papers in this area is available in the last author's webpage at <http://www.ee.sunysb.edu/ tom/dlt.html>.

3 Load Distribution Strategies: Design and Analysis

The basic idea underlying the process of scheduling divisible loads to minimize the processing time in distributed networks is in devising efficient load distribution strategies. While a data partitioning algorithm is simple to implement, the non-triviality of scheduling divisible loads lies in designing strategies that efficiently utilize the available network resources in terms of computational power and communication channel bandwidth.

3.1 The Load Distribution Model

Divisible load distribution, in general, goes through the following process. The load to be processed arrives at a node, called the *originator* or the *root* node (in the case of tree networks), depending upon the architecture under consideration. Also, the architecture can be such that the processors can be equipped *with front-ends* or *without front-ends*. In the *with front-end* case, with a network involving m processors, the originator partitions the load into m fractions, starts the computation on its own load fraction and simultaneously starts distributing the other load fractions to other

processors one at a time in a predetermined order. Note that the computation and communication events occur concurrently at the originator, if it is equipped with a front-end (also known as a *communication co-processor*). On the other hand, in the *without front-end* case, the originator first distributes the load fractions to the rest of the processors and then it computes its own load fraction. Of course, when we consider a linear topology for the network, the originator pumps all the data in a pipelined fashion, and every processor that receives the data from its predecessor keeps the portion intended for it and passes the rest to its successor. The problem is then to choose the size of these load fractions in such a way that our objective of minimum processing time is met. It is important to note that we are addressing the problem of load partitioning in a heterogeneous system of processors and links, and hence, dividing the load into equal sized fractions will naturally result in a poor performance.

3.2 Nomenclature

Below we describe the standard notations used in the divisible load scheduling literature.

$\alpha = (\alpha_1, \ldots, \alpha_m)$: Load distribution vector

α_i : Load fraction allocated to processor p_i

$T(\alpha)$: Finish time with load distribution α

w_i : Ratio of the time taken by processor p_i, to compute a given load, to the time taken by a standard processor, to compute the same load

T_{cp} : Time taken to process a unit load by the standard processor

z_i : Ratio of the time taken by link l_i, to communicate a given load, to the time taken by a standard link, to communicate the same load

T_{cm} : Time taken to communicate a unit load on a standard link

$\sigma = \frac{zT_{cm}}{wT_{cp}}$

Then $\alpha_i w_i T_{cp}$ is the time to process the fraction α_i of the entire load on the i-th processor. Note that the units of $\alpha_i w_i T_{cp}$ are [load] $\times$ [sec/load] $\times$ [dimensionless quantity] = [seconds]. Likewise, $\alpha_i z_i T_{cm}$ is the time to transmit the fraction α_i of the entire load over the i-th link. Note that the units of $\alpha_i z_i T_{cm}$ are [load] $\times$ [sec/load] $\times$ [dimensionless quantity] = [seconds].

The *standard* processor or link referred to above is any processor or link which is used as a reference. It could be any processor or link in the network or a conveniently defined fictitious processor or link.

3.3 Linear Networks

The first target architecture to be examined in the study of divisible loads, was that of a (linear) daisy chain [3]. This was done based on a perception that such a reduced case might be tractable. The original application was for sensor networks where the "sensors" were networked computers that shared information. In a linear network the processor p_0 (the *root*) is connected to processor p_1 via link l_1, p_1 is connected to p_2 via l_2 and so on until p_{m-1} is connected to the boundary processor p_m via l_m. If a divisible load originates at one end of a daisy chain of m processors then a set of m linear equations can be set up to solve for the optimal fraction of load to be assigned to each processor in order to minimize the "finish time". Here finish time is the time when all processing has ceased. Other variations to this problem deal with a load that originates at an interior processor and also when the time for processors to report solutions back to the originator is non-negligible.

This optimal assignment of load is done in the context of the schedule of load distribution that the equations are based on. One can have load distribution strategies that involve round robin distribution of load to the processors or strategies that simply distribute load to each processor in turn once. One can also distinguish between a store and forward reception of load and those strategies that are more akin to the virtual cut through switching strategy of networking.

Finally, linear daisy chains that are infinite in extent can be solved to obtain performance bounds. Actually finish time tends to saturate as more processors are added to the network because of the repetitive overhead in communicating load down a chain. Using concepts from algebra, combinatorics, and even electric circuit theory, both the optimal load allocations and finish time of infinite sized daisy chains can be obtained [4, 5]. Such infinite networks represent a performance benchmark that finite chains can be compared against. However, it soon became apparent that other topologies such as tree and buses would yield superior performance.

As an illustration, we consider a linear network with three processors. The equations, however, are easily generalizable to m processors. Each processor is equipped with a front-end. The timing diagram, given in Figure 2, shows communication delay above the time axis and computation time below. The finish times for all processors are assumed to be equal for optimal

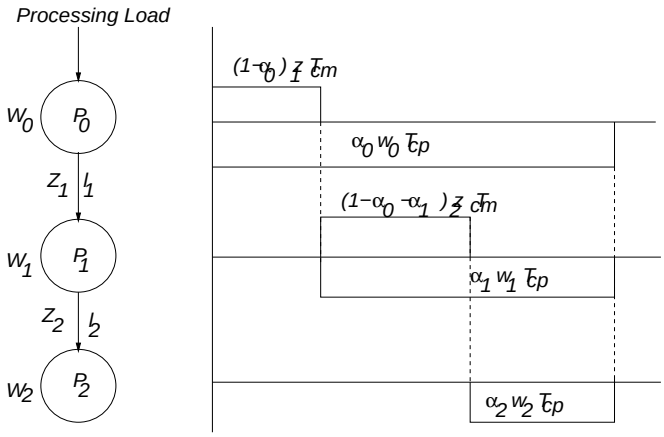

Figure 2: Load distribution in a three-processor linear network

load distribution (see Section 3.5).

$$\begin{aligned} \alpha_0 &+ \alpha_1 + \alpha_2 = 1 \\ \alpha_i w_i T_{cp} &= (1 - \alpha_0 - \cdots - \alpha_i) z_{i+1} T_{cm} + \alpha_{i+1} w_{i+1} T_{cp}, \quad i = 0, 1 \end{aligned}$$

These linear equations can be reduced to a series of product forms that are easy to compute recursively and whose elegance aids analysis. Closed form solutions are available when the network is homogeneous (that is, equal link speeds and processor speeds).

3.4 Tree and Bus Networks

Research during the very early stages of divisible load theory considered these networks [6] in addition to linear networks discussed in the previous section. In these studies, a bus network architecture was conceived as a special case of single-level homogeneous tree networks. This is, of course, expected as our modeling ensures that when all the link speeds are identical in single-level tree network (SLTN) or in star network architectures, the resultant network becomes identical to a bus network. Thus, all the results that are valid for SLTN also hold for bus networks. The treatment to obtain optimal finish time solution follows the same technique as the linear network.

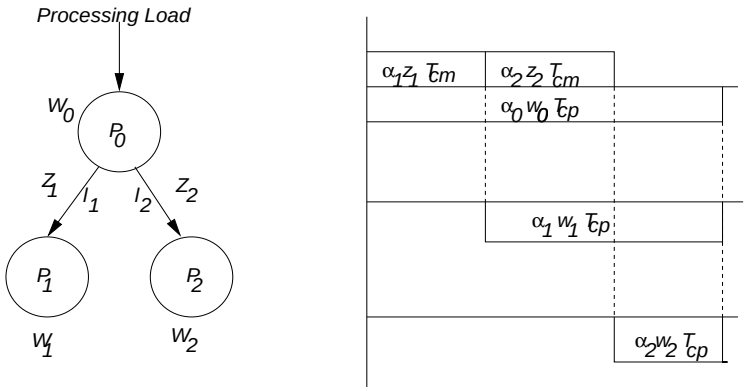

Figure 3: Load distribution in a three-processor tree network

There is a possibility in tree networks of varying the order or sequence of load distribution among the child processors. There are $m!$ such sequences possible with m processors connected to the root. An *optimal sequence* is that in which the load distribution follows the order in which the link speeds decrease. However, from a network designer's perspective, if architectural rearrangement is permissible, then the best way one can arrange the processors and links would be to connect the fastest processor to the fastest link and the next fastest processor to the next fastest link and so on and then follow the optimal sequence of load distribution. Further, when front-ends exist the root must be the fastest of all the processors in the network [7, 8]. The extension of many of these results to multi-level tree networks is also available [9].

In bus networks, three possible configurations are of interest: (i) Bus equipped with a control processor or a bus controller unit (BCU), (ii) Bus not equipped with a BCU, but processors with front-ends, and (iii) Bus not equipped with a BCU, but processors without front-ends. While the analytical treatment and the load distribution process remains identical, the study of divisible load scheduling on bus networks is of interest since the network is simple in nature and allows one to design and study the performance of complex scheduling strategies [9, 10].

As an illustration we consider a tree network with one root processor and two child processors in Figure 3. The corresponding load distribution equations are,

$$\begin{aligned} \alpha_0 &+ \alpha_1 + \alpha_2 = 1 \\ \alpha_i w_i T_{cp} &= \alpha_{i+1} z_{i+1} T_{cm} + \alpha_{i+1} w_{i+1} T_{cp}, \quad i = 0, 1 \end{aligned}$$

These equations too can be solved recursively. Closed-form solutions are also possible.

As a numerical example, consider a single-level heterogeneous tree with 4 processors where $w_0 = 2,\ w_1 = 3,\ w_2 = 1,\ w_3 = 2,\ z_1 = 2,\ z_2 = 0.5,\ z_3 = 5$ and $T_{cm} = T_{cp} = 1$. If we use all the processors then $\alpha = \{0.4321, 0.1728, 0.3457, 0.0494\}$ and $T(\alpha) = 0.8642$. But this distribution is not optimal. Suppose p_2 is given $\alpha_1 + \alpha_2$ and p_1 is not given any load, then the new load distribution is $\alpha' = \{0.4321, 0, 0.5185, 0.0494\}$ and the processing time is $T(\alpha') = 0.8642$. In this case, the processors do not stop at the same time. If we redistribute load to the reduced network consisting of p_0, p_2, and p_3 only so that they all have the same finish times then the new load distribution is α" $= \{0.3962, 0, 0.5283, 0.0755\}$ and $T(\alpha") = 0.7924$, which is indeed the optimal finish time solution to this problem.

3.5 The Optimality Principle

In the above discussions we assumed that to obtain optimal processing time all the participating processors must stop computing at the same instant in time. This was the basic *optimality principle* in the case of divisible load scheduling problems. This assertion is supported by an intuitive observation that when all processors do not stop computing at the same time, it is possible to redistribute some load from processors that stop computing later to those that stop computing earlier. While the above claim seems to have an intuitive validity, it was subsequently shown that, for a single level tree network, the optimal processing time can be achieved by distributing the load only among the "fast" processor-link pairs. An exact expression that distinguishes the "fast" processor-link pairs from the "slow" processor-link pairs has been derived [1]. A *reduced* network can then be obtained after eliminating the slow processor-link pairs and the load is distributed among the remaining processors using the optimality principle.

Based on the above, it is reasonable to say that although the optimality principle remains valid for even an arbitrary network topology, the optimal time performance depends crucially on the selection of a proper subset of the available processors. Thus, using a larger set of nodes may yield an inferior performance compared to an optimal subset of nodes among which the load is distributed according to the optimality principle.

In the case of homogeneous single-level tree networks (and also for the bus networks) all the

processor-link pairs are identical and hence all of them must be used to process the load. this has been shown to be true in [11].

3.6 Multi-Installment Strategy

The load distribution model discussed above underwent a fair bit of revision when it incorporated pipelining in the form of a multi-installment strategy, where a processor need not wait till the complete load fraction to its predecessor has been transferred. Exploiting the divisible nature of the load, each fraction was further subdivided and distributed in a repetitive sequence [12, 13]. This strategy reduced the idle time of the processors at the farthest end of the load distribution sequence. In addition to a resulting reduction in processing time, one can also have a control on the finish time by selecting the number of installments. This capability is crucial for real-time processing of certain types of loads.

3.7 Performance Saturation

The presence of communication overheads limits the performance of networks processing divisible loads. Although the speed-up performance and the finish time performance improves with increase in the number of processors m in the network (and the number of installments n in the case of multi-installment strategies) the incremental performance improvements follow the law of diminishing returns. The performance curves are governed by a relation somewhat similar to Amdahl's law as the communication overhead associated with load transfer takes place in a sequential fashion. In Figures 4 [1] we see the speed-up saturation as the number of processors increases in a homogeneous linear and tree network, respectively, equipped with front-ends and using a single installment load distribution strategy. In Figure 5 [1] we see the variation in the optimal finish time $T(m, n)$ with the number of processors m and the number of installments n in a homogeneous tree network equipped with front-ends and using a multi-installment strategy [13, 14]. Asymptotic analyses of various load scheduling policies is quite tractable in the divisible load framework. One way to go about this is to develop expressions for a single processor whose characteristics are equivalent to a network of processors of interest [15]. That this can be done is not too surprising as divisible modeling is generally linear, just as in the case of linear electric circuit theory and its concept of equivalent circuits.

Another contribution in this direction is [16] where the parallel time and speedup is analyzed for a linear array, a mesh, a multi-level n-ary complete tree, and a pyramid. It is shown that the speedup is bounded from above by a quantity independent of network size, but dependent on the

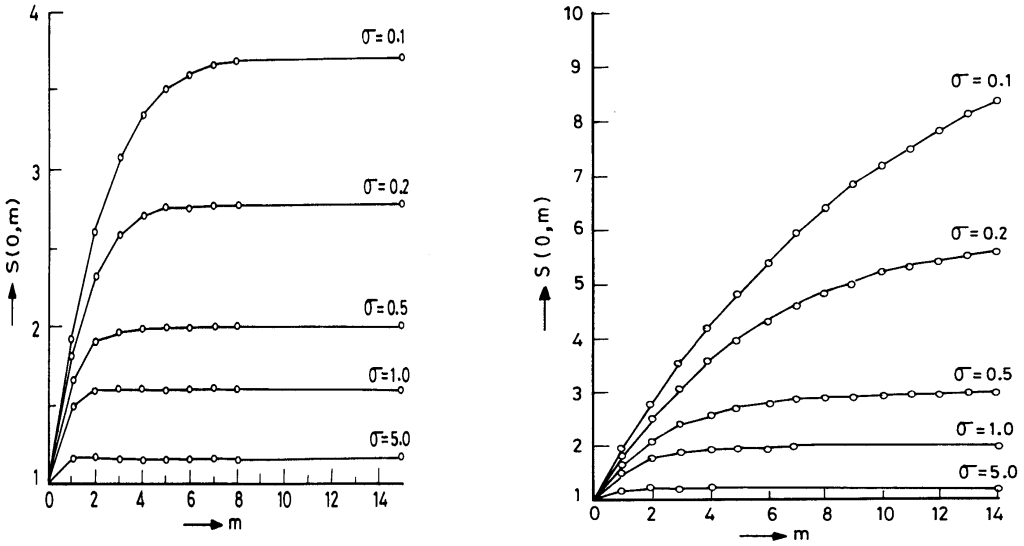

Figure 4: Speed-up curves for (a) Linear network (b) Tree network (©1996 IEEE)

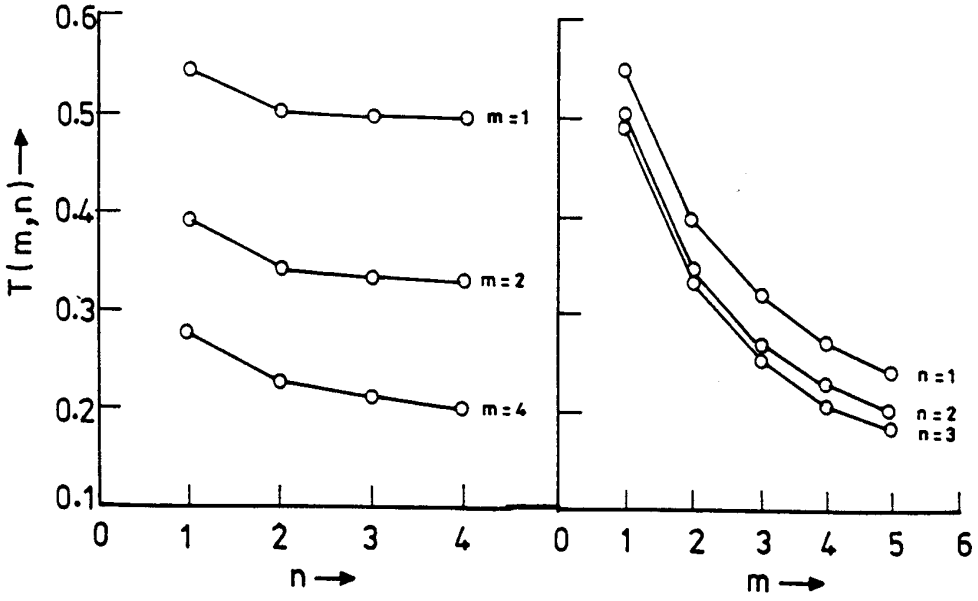

Figure 5: Optimal finish time curves for a tree network using a multi-installment strategy against (a) Number of installments (b) Number of processors (©1996 IEEE)

architecture. An asymptotic performance analysis based on a single processor equivalent of a mesh of processors has been conducted in [27]. In [29] analytical performance evaluation is presented for three dimensional mesh of processors.

4 Applications and Extensions

Several potential applications of divisible load theory has been mentioned earlier. Specifically, an edge detection application in the image processing domain has been considered [1]. In this example, the difference in the finish times when one considers a parallel system (communication delays are negligible) and a distributed system (non-negligible communication delays) are clearly demonstrated with a numerical example. Subsequently, several applications and extensions of the divisible load paradigm have been addressed in the literature. We will discuss some of them below.

4.1 Large Matrix-Vector Products

Large matrix-vector products are necessary in several applications involving iterative solutions of matrix equations. For example, in the design and simulation of microwave ovens, the finite element approximation of a wave equation in a cavity filled with non-homogeneous material gives rise to large systems of linear equations with near about 300,000 unknowns. Solutions of such equations via iterative methods require the computation of large-sized matrix-vector products. Such cases also arise during the design of complex control systems with large number of state variables.

Due to the large size of the matrices and the homogeneity of the computational process it is possible to treat these problems as divisible jobs. For example, consider the computation of a matrix-vector product $b = Ax$, where b is an $m \times 1$ vector, A is an $m \times n$ matrix, and x is an $n \times 1$ vector. The total number of multiplications and additions needed to compute b is mn and $m(n-1)$, respectively. The problem is decomposed using a row striping strategy as follows: Let

$$\begin{aligned} A &= (A_1, A_2, \ldots, A_m)^T \\ x &= (x_1, x_2, \ldots, x_n)^T \\ b &= (b_1, b_2, \ldots, b_m)^T \end{aligned}$$

where, A_i is the i-th row of the matrix A, x_i is the i-th element of the vector x, and b_i is the i-th element of the vector b. Then, b_i can be computed as

$$b_i = A_i x, \quad i = 1, \ldots, m$$

where, $A_i x$ is the inner product of the vectors A_i and x. The load processed by a processor P_i is given in terms of the number of rows m_i of the matrix A that P_i processes. That is, P_i computes m_i elements of the vector b. Thus,

$$m_0 + m_1 + \cdots + m_p = m$$

Thus, in the matrix-vector problem defined above, the $(p+1)$-tuple $(m_0, m_1, \ldots, m_p)$ defines a load distribution.

It is also possible to solve this problem using a column-striping strategy where, if a processor gets columns j to k then it is also assigned x_j to x_k elements of the vector x. Each processor computes m partial sums of the vector b. The consolidation of the results at the root constitutes p vector additions to obtain the elements of b. Thus, a total of pm additions have to be performed by the root after the main computations are over. This accumulation of partial sums in the child processors increases the computational burden on the root while the child processors remain idle. Since the broad objective of load partitioning is to distribute the arithmetic operations (additions and multiplications) in a way that would minimize the processing time, row striping (in which consolidation implies only collecting the elements of b) provides a more efficient distribution. On the other hand, column striping gives rise to the possibility of multicasting the vector x too. But this advantage is not very significant when $m \approx n$ or $m \gg n$ since A has $O(m)$ more elements than x. Consequently, since the communication time is usually much less than the time for performing arithmetic operations, the computation time at the root turns out to be far in excess of the communication time saved by multicasting the vector x [17].

4.2 Experimental Verifications

While the above paper emphasizes the development of a theoretical framework, in the work in [18], experimental evidence to validate the time performance of algorithms based on the divisible load paradigm on Ethernet based bus networks is reported. In this experiment, a network of Pentium III series computers is used as a test-bed and a matrix of size $200 \times 100,000$ is considered to implement the matrix-vector multiplication algorithm. A software architecture based on C++ Microsoft foundation class is developed. All the theoretical findings reported in [17] were tested and the time performance is demonstrated by setting different speed parameters.

Other experimental verifications were reported in [19], where the authors implemented applications such as *pattern search*, *file compression*, *joining operation in relational databases*, *graph coloring,* and *genetic search* using the divisible load paradigm. In the case of pattern search ap-

plication, a given string of characters are searched for a known substring. In file compression applications, parts of the file, in chunks of 10KB each, are compressed independently using LZW compression algorithm and the resulting compressed files are sent back to the originator. In the join operation two lists, extracted from relational databases, are joined to produce a final list. One of the lists, say A, was transmitted to all the processors and the second list, say B, is partitioned into m parts and each of the processors are allowed to calculate the *the join on A and* B_i $i = 1, ..., m$ and return the results to the originator. In the last application, the problem of determining a chromatic number is a hard problem and hence, genetic search was used to solve this problem approximately. The population derived from a pool of possible genes are distributed among the processors and each processor generates a fixed number of new populations and these are transmitted back to the originator. The results from the theoretical model and the experimental values were found to be close. This quite convincingly shows the applicability of divisible load paradigm to several important applications.

4.3 Monetary Cost Optimization

It is interesting to consider scheduling models where the monetary cost of scheduling is a consideration. One reason is that for a "computer utility" that charges customers for access to distributed and networked computer resources an important question is the management of computing and communication to provide profitable and attractive service. A second reason is that an important question in current computer architecture and microelectronics is whether it is better to design a system using one very fast but expensive processor or to use many cheap but slow processors. One can use the linear scheduling theory of divisible loads to address such issues. The initial work [20] on such cost modeling considered bus or single level tree interconnected networks of front-end equipped processors. A contribution that applies these ideas to parallel processor configuration design is [21].

There are really two optimization criteria, that are sometimes conflicting, in such situations: finish time and cost. While there are a variety of ways to trade cost against finish time, a natural way is to minimize cost such that for a given sequence of load distribution (i.e., the order in which processors receive load) finish time is minimized. In a bus network if only computation costs are considered a simple ordering based on the ratio of computing cost and computing speed is optimal. In a single level tree network, if link transmission costs are also considered then there is no simple optimality condition but rather a set of moderately complex algebraic conditions to determine when an interchange of processors in the load distribution order is beneficial. Such conditions, or

even a direct cost evaluation, can be used as the basis of a greedy algorithm to create efficient (though not always optimal) load distribution orders. Modern heuristic techniques such as tabu search or simulated annealing can be applied to such problems effectively.

There are many possible variations to such monetary cost optimization problems. One may seek to develop algorithms to minimize cost subject to a bound on finish time or to minimize finish time with a bound on cost. One may optimize, in a tree network for instance, over logical sequences of load distribution from parent node to children or one may optimize over different physical arrangements of processors and links. This flexibility in modeling is ideal for addressing important problems in cost optimization.

4.4 Efficient Movie Retrieval for Network-based Multimedia Systems

As opposed to the idea of data striping in organizing the data onto a set of disks (for example, RAID technology), divisible load paradigm proposes a totally different way of retrieving a long duration movie to service a client request on a network. In the conventional video/movie-on-demand systems (V/M-oD), a single server serves a pool of requests and also attempts to maximize the number of clients that it can serve (admission control algorithms). Whereas the divisible load paradigm allows an elegant solution in which a pool of servers are considered for retrieving a movie to the client site. Thus, in this case, even low-bandwidth servers can be effectively utilized in the retrieval process while, at the same time, every server can maximize the number of clients that it can support. This is due to the fact that the server is used only for a small amount of time. Typically, the following equations govern the retrieval process.

$$m_{i+1} bw_{i+1} \leq m_i bw_i + m_i R_p, \quad i = 0, 1, ..., N-2. \tag{1}$$

where, m_i is the portion of the movie retrieved from server S_i, bw_i is the inverse of the bandwidth, and R_p is the playback rate of the movie at the client site. The inequality captures an important relationship referred to as the *continuity constraint*, which has to be satisfied so as to have a continuous presentation at the client site. Thus, solving these set of recursive equations (with equality conditions) determine the minimum amount of movie portion to be retrieved from each server maintaining the continuity relationship, and yields minimum time to access the movie (minimum time that a client has to wait before the start of the presentation). The idea of multiple installments can also be directly applied and the work in [22] presents a complete analysis as well as simulation results of these schemes. The above idea was extended to the case of a generic scenario in which multiple movies are requested by multiple clients from different sites in an arbitrary network topology, in [23]. Here, a scheduler is designed to carry out a careful allocation of the

available server bandwidth among client requests. The minimum size of buffers needed at the client site and the size of movie portions to be downloaded from each server to the respective clients are also derived.

4.5 Collection-Aware Query/Image Processing

A study that extends the concept of optimal sequencing to general tree networks, when one considers both the load distribution phase and the results collection phase, is given in [24]. The problem is solved for two different classes of applications, namely *query processing* and *image processing*. Query processing applications execute a search through the database and the result of the query is a fixed size data. Image processing applications, on the other hand, extract data homogeneously from the database and so the size of the result is proportional to the size of the data. The paper extends the concept of utilizing an optimal subset of nodes to general tree networks. The set of nodes that do not participate in the computation process are referred to as *conductor nodes*. These nodes just communicate loads to their descendants. Using the concept of equivalent processors, an algorithm is proposed to construct an optimal load distribution sequence. Also, several *greedy* strategies are proposed to yield the optimal subset of active nodes. A second work on query processing [25] develops solutions for expected record search time in flat file distributed over either a linear daisy chain or a single level tree network.

4.6 Other Network Topologies

Apart from linear arrays and tree networks, several other network topologies, such as mesh and hypercubes, have been considered by researchers [26]-[32]. These topologies have been used as interconnection architectures for dedicated parallel processors. Important results such as speed-up and processor utilization were obtained. As one would naturally expect, the speed-up and the utilization were found to be dependent on the dimension of the respective networks. Simulation results reported reflect this behavior and also highlight the possibility of achieving a much desirable linear speed-up.

4.7 Release Times, Multi-Tasking, and Fault-Tolerance

In a general distributed network, there is no guarantee that all processors will be available for use at the same time instant. In a study by Bharadwaj et al. [33], availability of processors was modeled as a constraint in the problem formulation and various possible *release time* distributions

were considered in scheduling a divisible load. For an arbitrary processor release time distribution, a load sharing condition was found that identifies processors that are required at every stage of iteration. For some specific release time distributions, the load distribution adopted a multi-installment strategy. In fact, such a mix of strategies was found to achieve optimal processing time.

Multi-tasking capability has been considered in bus networks to schedule divisible loads [1]. A numerical algorithm to find optimal fraction of the entire work load for minimal processing time is presented by a deterministic analysis when the background job's arrival and departure process is known exactly, and by stochastic analysis when these processes are not known. In the deterministic analysis, when there are k background jobs running in a processor, each job receives $1/k$ of the total computational power of the CPU. When a job leaves the system, the remaining jobs receive an increased amount of the CPU time. In the case of stochastic analysis, the arrival and the departure process of the jobs are modeled as a M/M/1 queueing network and also as a M/G/1 queueing model.

Fault-tolerance was considered by Bataineh et al. [34] in the context of a bus network. They consider an instance when a processor becomes faulty after the computation process has begun and its load has to be redistributed among the healthy processors. This requires interprocessor communication. Several methods have been considered in this study to redistribute the load held by a processor. A probabilistic analysis yields the average unfinished task in the system and the corresponding average finish time.

4.8 Start-Up Costs and Time-Varying Loads

The process of communication or computation demands an initial start-up time. More detailed modeling of communication and computation loads is possible using the divisible load paradigm. For instance, the process of communication and computation may include a significant fixed start-up delay. These costs (really delays, not monetary cost) were explicitly considered by Blazewicz et al. [35] and Bharadwaj et al. [36]. Recursive equations were developed and a condition to determine the maximal set of processors that can be assigned the load fractions was derived. This process of obtaining the maximal set is found to have the order of complexity $O(\log\ m)$, using binary search procedures. Expressions for the speed-up and utilization were also derived. While [35] considers the start-up costs in communication time of the load fractions alone, in [36], the start-up costs were included in both the communication and computation times of the loads. These overhead factors are considered as additive components, thus defining the computation and communication models

as $\alpha_i w_i T_{cp} + \theta_{cp}$ and $\alpha_i z T_{cm} + \theta_{cm}$, respectively. Using this model, a necessary and sufficient condition on the existence of an optimal solution (employing the optimality principle) using m processors in the system was derived. The concept of sequencing the load distribution was also considered with these overhead, and it was shown that, when the overhead parameters are included in the analysis, the processing time depends on the sequence of load distribution. Thus, in [36], the influence of these overhead factors on the sequencing is analyzed and an optimal sequence is obtained. This paper [36] also presents a greedy algorithm that produces a sub-optimal solution, when an optimal sequence fails to exist with a m-processor sequence.

The effect of overheads was also analyzed for the case of linear networks and necessary and sufficient conditions for utilizing all the processors in the system were derived [37]. Both the cases when the processors are equipped with and without front-ends are considered.

In [38] the problem of partitioning a very large image data is considered on a bus network of processors with start-up costs. Different possible data partitioning schemes were discussed. Using row-wise partitioning scheme, closed-form solution for optimal number of pixels to be distributed to the processors and the optimal processing time are derived by assuming the data to be arbitrarily divisible.

A second feature of loads, their time-varying nature, is examined in [39]. Here processor and/or link speed is modeled as time-varying because of the presence of background jobs/transmissions. In this environment, if the arrival/departure times of background loads are known in advance, it is possible to optimally distribute a divisible job among a number of processors. A first study at a stochastic analysis with no future knowledge, also appears in this work.

A paper that uses a combination of queueing and divisible load theory to determine bounds on the arrival rate of loads as a function of system and load parameters is [40].

4.9 Granularity

A specific problem of divisible loads is the *granularity* of the loads assigned to the processors. A divisible load can be *coarse-* or *fine-* grained depending on the application and processing requirements. In most of the literature on divisible load theory, all loads are considered to be fine grained to the extent that they are assumed to be infinitely divisible. However, for most current day application requirements, this assumption imposes severe constraints.

In Bharadwaj et al. [36], for bus networks with m processors, integer approximation is carried out to yield integer load fractions. Thus, in this case, we obtain only a sub-optimal

solution (as the processors will not stop at the same instant in time) and it was shown that the solution obtained using the integer approximation technique is within a radius of $(wT_{cp} + zT_{cm})$ from the optimal solution, for a homogeneous system of processors in a bus network. Further, in [41], this issue has been addressed and a restricted partitioning of the loads (at most k partitions are allowed) is considered. A minimum size, referred to as *divisibility factor*, is considered in the analysis of divisible load scheduling. Thus, the load fractions assigned to the processors can no longer be of arbitrary size, but they must be integral multiples of this fundamental divisibility factor. Using this restricted partitioning constraint together with the granularity constraint, two algorithms, referred to as *PIA* (*Processor based Integer Approximation*)and *IIA*(*Installment based Integer Approximation*) are proposed. In the case of PIA strategy, while carrying out the integer approximation, the surplus (deficit) load is "pushed" to ("pulled" from) its successor by a processor, while in IIA strategy, the surplus (deficit) is "pushed" to ("pulled" from) the next round of the same processor. Using these algorithms, the performance bounds (the bounds on the sub-optimal solutions generated) is found to be within a radius of $(mzT_{cm})/2 + w_{cp}$ and $zT_{cm} + (kwT_{cp})/2$, from the optimal solution, respectively.

4.10 Scheduling Hybrid Loads

In general, loads arriving at a computer system may not be strictly of one particular type. For instance, jobs can be indivisible, modularly divisible, or arbitrarily divisible. Also, it is possible that different parts of the load may be of different types. An algorithm to schedule both indivisible and divisible loads was proposed by Bataineh et al. [42] with an objective to obtain a schedule that gives the minimum finish time. The algorithm was designed for a Mach operating system. The central idea of the algorithm is to allow all the processors to serve the divisible loads first and then the indivisible loads.

However, in some cases, it was found that this policy may not lead to an optimum solution. In these cases, part of the processors are assigned indivisible loads and the rest are assigned divisible loads. The assignment depends on the number of indivisible loads, their finish times, and the total finish time of the divisible loads. In this study the communication delay is included in the execution time of the task, as it is assumed that the communication overhead for each task is available a priori. Although this assumption is somewhat strong, in the case of parallel processing systems, where the communication overheads are almost insignificant, the algorithm seems to be relevant.

A recent paper presents an algorithm for scheduling divisible and indivisible tasks in multiprocessor systems [43]. In addition to applying both paradigms, the algorithm is proved to be

better than all existing ones and its results are very close to optimal values.

4.11 Scheduling with Finite-size Buffer Constraints

In processing divisible loads, an important practical issue is the available buffer size capacity at the nodes for processing. In fact, this limits the amount of data that a single node can accept for processing. Very recently, the issue of finite-size buffers are considered and an algorithm referred to as *incremental balancing strategy*(IBS) is proposed in the literature [44]. This algorithm is demonstrated to work in an incremental fashion by checking the available buffer at each iteration before supplying the load. This algorithm is an off-line algorithm and determines the amount of load that can be supplied to each node in every iteration. The algorithm guarantees that there will be no load that is left unprocessed when it terminates and the order of complexity is proven to be $O(m)$. Several important properties leading to the proof of optimal solution is presented in [44]. The IBS algorithm was also applied to a hypercube cluster in [45]. An n-dimensional hypercube is considered and the load is assumed to originate at one of the corner processors. The processors were assumed to have a limited buffer capacity. The load is partitioned and assigned to the processors as recommended by the IBS algorithm and time performance is measured. Also, during the load distribution process, the number of layers of processors on the hypercube that can be assigned the load is determined based on granularity constraints.

5 Conclusions

In this paper we have surveyed the recent but fairly extensive literature on divisible load theory with special emphasis on recent work done in this area. The wide spread of applications of divisible load theory to practical problems is apparent from the diverse fields in which this theory has contributed. The early literature on divisible load theory primarily concentrated on developing theoretical tools to analyze the various algorithms that arise from the basic framework. However, the more recent work focuses on the applications and extensions of this theory. In fact, some of the recent efforts have been in the area of experimental verifications where encouraging results have been obtained. We hope that this survey paper will help to compile at one place the recent work on divisible load theory and encourage researchers to take up challenging problems in this area of research.

References

[1] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos CA, 1996.

[2] M. Drozdowski, *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*, Politechnika Poznanska, Book No. 321, Poznan, Poland, 1997. <http://www.cs.put.poznan.pl/txt/h.ps>.

[3] Y.C. Cheng and T.G. Robertazzi, Distributed computation with communication delays, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 24, No. 6, November 1988, pp. 700-712.

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

[5] T.G. Robertazzi, Processor equivalence for a linear daisy chain of load sharing processors, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 29, No. 4, October 1993, pp. 1216-1221.

[6] Y.C. Cheng and T.G. Robertazzi, Distributed computation for a tree network with communication delays, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 26, No. 3, May 1990, pp. 511-516.

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

[8] H.J. Kim, G.-I. Jee, and J.G. Lee, Optimal load distribution for tree network processors, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 32, No. 2, April 1996, pp. 607-612.

[9] S. Bataineh, T. Hsiung, and T.G. Robertazzi, Closed form solutions for bus and tree networks of processors load sharing a divisible job, *IEEE Transactions on Computers*, Vol. 43, No. 10, October 1994, pp. 1184-1196.

[10] S. Bataineh and T.G. Robertazzi, Bus oriented load sharing for a network of sensor driven processors, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No.5, September 1991, pp. 1202-1205.

[11] J. Sohn and T.G. Robertazzi, Optimal load sharing for a divisible job on a bus network, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 32, No. 1, January 1996, pp. 34-40.

[12] V. Bharadwaj, D. Ghose, and V. Mani, An efficient load distribution strategy for a distributed linear network of processors with communication delays, *Computers and Mathematics with Applications*, Vol. 29, No. 9, May 1995, pp. 95-112.

[13] V. Bharadwaj, D. Ghose, and V. Mani, Multi-installment load distribution in tree networks with delays, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 31, No. 2, April 1995, pp. 555-567.

[14] D. Ghose and V. Mani, Distributed computation with communication delays: Asymptotic performance analysis, *Journal of Parallel and Distributed Computing*, Vol. 23, No. 3, December 1994, pp. 293-305.

[15] S. Bataineh and T.G. Robertazzi, Performance limits for processor networks with divisible jobs, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 33, No. , April 1997, pp. .

[16] K. Li, Managing divisible load on a partitionable network, *High Performance Computing Systems and Applications*, J. Schaeffer, editor, Kluwer Academic Publishers, 1998, pp. 217-228.

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

[18] S.K. Chan, V. Bharadwaj, and D. Ghose, Experimental study on large size matrix-vector product computations using divisible load paradigm on distributed bus networks, Technical Report No. TR-OSSL/BV-DLT/01, Open Source Software Laboratory, Department of Electrical and Computer Engineering, The National University of Singapore, Singapore, November 2000.

[19] M. Drozdowski and P. Wolniewicz, Experiments with scheduling divisible tasks in clusters of workstations, Euro-Par 2000, LNCS 1900, Springer-Verlag, pp. 311-319, 2000.

[20] J. Sohn, T.G. Robertazzi, and S. Luryi, Optimizing computing costs using divisible load analysis, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 3, March 1998, pp. 225-234.

[21] S. Charcranoon, T.G. Robertazzi, and S. Luryi, Parallel processor configuration design with processing/transmission costs, *IEEE Transactions on Computers*, Vol. 49, No. 9, September 2000, pp. 987-991.

[22] V. Bharadwaj and G. Barlas, Access time minimization for distributed multimedia applications, *Multimedia Tools and Applications*, Vol. 12, No. 2/3, pp. 235-256, November 2000.

[23] L.G. Dong, V. Bharadwaj, and C.C.Ko, The design of a multiple server strategy for distributed multimedia video-on-demand services, Technical Report No. TR-OSSL/BV-MM/01, Open Source Software Laboratory, Department of Electrical Engineering, National University of Singapore, Singapore, June 2000.

[24] 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*, Vol. 9, No. 5, May 1998, pp. 429-441.

[25] K. Ko and T.G. Robertazzi, Record search time evaluation, *Proceedings of the Conference on Information Sciences and Systems*, Princeton University, Princeton, NJ, March 2000, p. WP2-24.

[26] J. Blazewicz and M. Drozdowski, Scheduling divisible jobs on hypercubes, *Parallel Computing*, Vol. 21, No. 12, December 1995, pp. 1945-1956.

[27] J. Blazewicz and M. Drozdowski, The performance limits of a two-dimensional network of load sharing processors, *Foundations of Computing and Information Sciences*, Vol. 21, No. 1, 1996, pp. 3-15.

[28] J. Blazewicz, M. Drozdowski, and M. Markiewicz, Divisible task scheduling – concept and verification, *Parallel Computing*, Vol. 25, No. 1, January 1999, pp. 87-98.

[29] M. Drozdowski, and W. Glazek, Scheduling divisible loads in a three-dimensional mesh of processors, *Parallel Computing*, Vol. 25, No. 4, April 1999, pp. 381-404.

[30] J. Blazewicz, M. Drozdowski, F. Guinand, and D. Trystram, Scheduling a divisible task in a 2-dimensional mesh, *Discrete Applied Mathematics*, Vol. 94, No. 1-3, June 1999, pp. 35-50.

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

[32] W. Glazek, Distributed computation in a three dimensional mesh with communication delays, *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Computing*, Madrid, Spain, January 1998, pp. 38-42.

[33] V. Bharadwaj, H.F. Li, and T. Radhakrishnan, Scheduling divisible loads in bus networks with arbitrary processor release times, *Computers and Mathematics with Applications*, Vol. 32, No. 7, 1996, pp. 57-77.

[34] S. Bataineh and M. Al-Ibrahim, Effect of fault-tolerance and communication delay on response time in a multiprocessor system with a bus topology, *Computer Communications*, Vol. 17, 1994, pp. 843-851.

[35] J. Blazewicz and M. Drozdowski, Distributed processing of divisible jobs with communication startup costs, *Discrete Applied Mathematics*, Vol. 76, No. 1-3, June 1997, pp. 21-41.

[36] V. Bharadwaj, Xiaolin Li, and, Ko Chi Chung, On the influence of start-up costs in scheduling divisible loads on bus networks, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 12, December 2000, pp. 1288-1305.

[37] V. Bharadwaj, X. Li, and K.C. Chung, Design and analysis of load distribution strategies with start-up costs in scheduling divisible loads on distributed networks, *Mathematical and Computer Modelling*, Vol. 32, No. 7-8, October 2000, pp. 901-932.

[38] V. Bharadwaj, X. Li, and 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*, Vol. 18, No. 11, August 2000, pp. 919-938.

[39] J. Sohn, and T.G. Robertazzi, Optimal time-varying load sharing for divisible loads, IEEE Transactions on Aerospace and Electronic Systems, Vol. 34, No. 3, July 1998, pp. 907-924.

[40] S. Bataineh and M. Al-Ibrahim, Load management in loosely coupled multiprocessor systems, *Dynamics and Control*, Vol. 8, No. 1, January 1998, pp. 107-116.

[41] V. Bharadwaj and N. Viswanadham, Sub-optimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks, *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, Vol. 30, No. 6, November 2000, pp. 680-691.

[42] S. Bataineh and B. Al-Asir, An efficient scheduling algorithm for divisible and indivisible tasks in loosely coupled multiprocessor systems, *Software Engineering Journal*, Vol. 9, 1994, pp. 13-18.

[43] A.M. Al-Saideen and S. Bataineh, A heuristic algorithm for scheduling multiclasses of tasks in multiprocessor systems, *International Journal of Computers and Applications*, Jan. 1999.

[44] X. Li, V. Bharadwaj, and C.C. Ko, Optimal divisible task scheduling on single-level tree networks with finite size buffers, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36, No. 4, October 2000, pp. 1298-1308.

[45] X. Li, V. Bharadwaj, and C.C.Ko, Divisible load scheduling on a hypercube cluster with finite-size buffers and granularity constraints, To appear in *Proceedings of the ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, May 2001, Brisbane, Australia.