New Divisible Load Distribution Methods on Tree and Pyramid Networks

KEQIN LI
SUNY

A divisible load can be arbitrarily divided into independent small load fractions which are assigned to processors in a parallel or distributed computing system for simultaneous processing. The theory and techniques of divisible load distribution have a wide range of aerospace applications, including satellite signal and image processing, radar and infrared tracking, target identification and searching, and data reporting and aggregation and processing in wireless sensor networks. We make new progress on divisible load distribution on tree and pyramid networks. We revisit the classic method for divisible load distribution on partitionable static interconnection networks (including complete tree and pyramid networks) and derive a closed-form expression of the parallel time and speedup. We propose two new methods which employ pipelined communication techniques to distribute divisible loads on tree and pyramid networks. We derive closed-form expressions of the parallel time and speedup for both methods and show that the asymptotic speedup of both methods is $b\beta + 1$ for a complete b-ary tree network and $4\beta + 1$ for a pyramid network, where β is the ratio of the time for computing a unit load to the time for communicating a unit load. The technique of pipelined communications leads to improved performance of divisible load distribution on tree and pyramid networks. Compared with the classic method, the asymptotic speedup of our new methods is 100% faster on a complete binary tree network and 33% faster on a pyramid network for large β.

Manuscript received September 30, 2008; revised April 1 and September 22, 2009; released for publication October 16, 2009.

IEEE Log No. T-AES/47/1/XXXXX.

Refereeing of this contribution was handled by T. Robertazzi.

Author's address: Dept. of Computer Science, SUNY, 1 Hawk Drive, New Paltz, NY 12561, E-mail: (lik@newpaltz.edu).

I. INTRODUCTION

A divisible load (job, task) can be arbitrarily divided into independent small load fractions which are assigned to processors in a parallel or distributed computing system for simultaneous processing. Given an arbitrarily divisible load without precedence constraint and a parallel/distributed computing system with communication delays, the problem of divisible load distribution is to find the proportions in which the load is partitioned and distributed among the processors such that the entire load is processed in the shortest possible time [11].

The theory and techniques of divisible load distribution have applications in a wide range of areas such as computer vision [12], large scale data file processing [29], data intensive applications [30], query processing in database systems [31], scientific computing [32], video and multimedia applications [38], numerical computing [47], biomedicine and bioinformatics [51]. Divisible load distribution has applications in filtering for radio communications, encryption for secure communications, and coding for digital communications [27]. Aerospace applications include satellite signal and image processing [13], radar and infrared tracking [25], and data reporting and aggregation and processing in wireless sensor networks [41]. Other applications include finite-element and engineering computations, grid computing, metacomputing, distance learning, real-time computing such as target identification and searching, data collection, and processing in distributed intelligent sensor networks in military surveillance systems [11].

Divisible load distribution, scheduling, sharing, and processing has been a very active and fruitful research field in the last twenty years since the problem was first proposed in 1988 [2, 19]. Extensive investigation has been conducted by numerous researchers for bus systems [6, 45], linear arrays [40, 43], tree networks [4, 6, 20], 2-dimensional meshes [15], 2-dimensional toroidal meshes [17], 3-dimensional meshes [21], k-dimensional meshes and tori [35, 36], hypercubes [14], partitionable networks [33, 34], arbitrary networks [52, 53], clusters [22, 48], grids [50], and networks of workstations [5, 39]. Other studies can be found in [7–9, 16, 18, 19, 23, 26, 28, 29, 42, 46, 49]. The reader is also referred to the web site [1] for more references in this field and Robertazzi's recent article [44] on divisible load theory.

When a divisible load is processed on a multicomputer system with a static interconnection network, there is communication overhead for distributing the load among the processors in the system. The network topology determines the speed at which a divisible load is distributed over a network and has strong impact on performance, i.e., parallel

processing time and speedup. It is well known that due to communication overhead and limited network connectivity, a load distribution method for parallel processing of divisible loads on a static interconnection network with constant node degrees can only achieve asymptotic speedup bounded from above by a quantity independent of the network size. This is consistent with Amdahl's law which states that a small fraction of sequential component sets a constant upper bound for speedup [3].

It has been proven in [34] that on a complete b-ary tree network, as the network size becomes large, an asymptotic speedup of approximately $(b-1)\beta$ for large β can be achieved for processing divisible loads, where β is the ratio of the time for computing a unit load to the time for communicating a unit load. It was also proven in [34] that on a pyramid network, as the network size becomes large, the asymptotic speedup for processing divisible loads is approximately 3β for large β.

The present paper makes new progress in the investigation of divisible load distribution on tree and pyramid networks. Our contributions are summarized as follows.

1) First, we revisit the classic method for divisible load distribution on partitionable static interconnection networks (including complete tree and pyramid networks) developed in [34]. We derive a closed-form expression of the parallel time and speedup of the classic method which is not available before.

2) Second, we propose two methods which employ pipelined communications to distribute divisible loads on tree and pyramid networks [37]. Surprisingly, the analyses of these seemingly more complicated algorithms are easier than that of the classic method. We derive closed-form expressions of the parallel time and speedup for both methods and show that the asymptotic speedup of both methods is $b\beta + 1$ for a complete b-ary tree network and $4\beta + 1$ for a pyramid network.

Compared with the classic method, the asymptotic speedup of our new methods is 100% faster on a complete binary tree network and 33% faster on a pyramid network for large β. We would like to mention that the technique of pipelined communications has been successfully applied to other networks such as k-dimensional meshes [36].

A number of techniques to overlap communication and computation have been proposed and developed in the literature. In cut through switching employed in a tree network [24, 25]; a node does not need to completely receive a load fraction from its parent before the load fraction is divided and forwarded to its descendants. Instead, a node can simultaneously receive a load fraction from its parent and transmit the portion received so far to its children. This communication mechanism certainly needs additional hardware support. In a model described in [27], it is assumed that a processor can start the computation of a load fraction as soon as it starts to receive the load fraction, that is, a processor can initiate data processing while it is still receiving data. However, this model can only be used for certain applications. In multi-installment load distribution on single-level tree networks [10], the load fraction processed by a processor is sent in multiple installments. However, analysis of multi-installment load distribution seems quite sophisticated when processors receive the same number of installments, and it is not clear how this method can be extended to multi-level tree networks.

Our pipelined communication algorithms proposed in this paper still use the traditional easy-to-implement store-and-forward communication model, namely, a load fraction should be received in its entirety before it can be further divided and forwarded. Furthermore, a load fraction cannot be computed or processed when it is still being transmitted. One of our algorithms does share the spirit of multi-installment load distribution. However, processors on different levels of a tree or pyramid network receive different numbers of installments. In particular, processors in lower levels receive fewer installments. As mentioned above, our pipelined communication algorithms for multi-level tree and pyramid networks are easy to analyze.

The rest of the paper is organized as follows. In Section II, we describe our model of divisible load distribution. In Section III, we review the classic method for divisible load distribution on partitionable static interconnection networks. In Sections IV and V, we develop our new methods using pipelined communications on complete binary tree networks. In Sections VI and VII, we extend our methods to complete b-ary tree networks and pyramid networks. In Section VIII, we compare the performance of the two new methods and the classic method. We conclude the paper in Section IX.

II. THE MODEL OF DIVISIBLE LOAD DISTRIBUTION

We consider parallel processing of a divisible load on a multicomputer system with N processors $P_1, P_2, \ldots, P_N$ connected by a static interconnection network. Each processor P_i has n_i neighbors. It is assumed that P_i has n_i separate ports for communication with each of the n_i neighbors. That is, processor P_i can send messages to all its n_i neighbors simultaneously. Once a processor sends a load fraction to a neighbor, it can proceed without waiting with other computation and communication activities. This provides the capability to overlap computation and communication and enhances the system performance. However, a neighbor (receiver) must wait until a load fraction arrives, and then start the processing of the load fraction. It is this waiting time that limits the overall system performance.

Let T_{cm} be the time to transmit a unit load along a link. The time to send a load to a neighbor is proportional to the size of the load, with a negligible communication start-up time. Let T_{cp} be the time to process a unit load on a processor. Again, the computation time is proportional to the size of a load. We use $\beta = T_{cp}/T_{cm}$ to denote the computation granularity, which is a parameter indicating the nature of a parallel computation and a parallel architecture. A large (small) β gives a small (large) communication overhead. A computation intensive load has a large β, and a communication intensive load has a small β. An infinite β implies that the communication cost is negligible.

Without loss of generality and for notational convenience, we assume in this paper that a unit load is defined such that $T_{cm} = 1$ and $T_{cp} = \beta$.

We use T_N^A to denote the parallel processing time of one unit of load by using the load distribution algorithm A. Since both computation and communication times are linearly proportional to the amount of load, the time for processing x units of load is xT_N^A for all $x \geq 0$. The speedup S_N^A is defined as the ratio of the sequential processing time to the parallel processing time, namely,

$$S_N^A = \frac{T_1}{T_N^A} = \frac{T_{cp}}{T_N^A} = \frac{\beta}{T_N^A}.$$

We are particularly interested in $T_\infty^A = \lim_{N\to\infty} T_N^A$ and $S_\infty^A = \lim_{N\to\infty} S_N^A$, that is, the ultimate parallel processing time and the asymptotic speedup when the size of a network goes to infinity.

III. THE CLASSIC METHOD

The classic method $\mathbb{C}$ was developed in the context of divisible load distribution on partitionable static interconnection networks [34]. Let $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ be a static network, where $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ is a set of N processors and $\mathcal{E}$ is a set of interprocessor connections. Consider a processor P_i which has n_i neighbors $P_{j_1}, P_{j_2}, \ldots, P_{j_{n_i}}$. We say that $\mathcal{G}$ is partitionable at processor P_i if the following two conditions are satisfied.

1) $\mathcal{P} - \{P_i\}$ can be partitioned into n_i disjoint subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_i}$, such that $P_{j_k} \in \mathcal{P}_k$, for all $1 \leq k \leq n_i$.

2) Furthermore, the subnetworks induced by $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_i}$ have the same topology (or, belong to the same family of networks, with a single processor being the base case) as the original network $\mathcal{G}$, and each $\mathcal{P}_k$ is partitionable at P_{j_k}, $1 \leq k \leq n_i$.

Example partitionable networks are linear arrays, meshes, hypercubes, and completely connected networks at any processor, trees at the roots, pyramids at the apexes, and stars at the centers.

We now consider a general static interconnection network $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ with N processors $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ partitionable at processor P_i which has n_i neighbors $P_{j_1}, P_{j_2}, \ldots, P_{j_{n_i}}$. Since $\mathcal{P} - \{P_i\}$ can be partitioned into n_i disjoint subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_i}$, such that $P_{j_k} \in \mathcal{P}_k$ for all $1 \leq k \leq n_i$, we use N_k to represent the size of $\mathcal{P}_k$, i.e., the number of processors in the subnetwork induced by $\mathcal{P}_k$, such that $N = N_1 + N_2 + \cdots + N_{n_i} + 1$. To process a load x, processor P_i sends a fraction α_k of the load x to the neighbor P_{j_k} for all $1 \leq k \leq n_i$, and continues to process the remaining load $(1 - \alpha_1 - \alpha_2 - \cdots - \alpha_{n_i})x$. Upon receiving the load $\alpha_k x$ by P_{j_k}, the subnetwork induced by $\mathcal{P}_k$ processes the load using the same strategy.

Let $T_N^{\mathbb{C}}$ denote the parallel processing time of a unit load on the above partitionable network with N processors by using the classic algorithm $\mathbb{C}$. For simplicity, we assume that $n_i = d$ and $N_1 = N_2 = \cdots = N_{n_i}$ for all processors. This implies that $\alpha_1 = \alpha_2 = \cdots = \alpha_{n_i} = \alpha$. When $N > 1$, we have

$$T_N^{\mathbb{C}} = (1 - d\alpha)T_{cp} = \alpha(T_{cm} + T_{(N-1)/d}^{\mathbb{C}})$$

which implies that

$$\alpha = \frac{T_{cp}}{T_{(N-1)/d}^{\mathbb{C}} + dT_{cp} + T_{cm}}.$$

Hence, $T_N^{\mathbb{C}}$ satisfies the following recurrence relation,

$$T_1^{\mathbb{C}} = T_{cp}$$

$$T_N^{\mathbb{C}} = \left(\frac{T_{(N-1)/d}^{\mathbb{C}} + T_{cm}}{T_{(N-1)/d}^{\mathbb{C}} + dT_{cp} + T_{cm}}\right) T_{cp}, \qquad N > 1.$$

A closed-form solution for $T_N^{\mathbb{C}}$ is given in the Appendix. To obtain $T_\infty^{\mathbb{C}}$, we take the limit on both sides of the last equation,

$$T_\infty^{\mathbb{C}} = \left(\frac{T_\infty^{\mathbb{C}} + T_{cm}}{T_\infty^{\mathbb{C}} + dT_{cp} + T_{cm}}\right) T_{cp}$$

and get

$$(T_\infty^{\mathbb{C}})^2 + ((d-1)T_{cp} + T_{cm})T_\infty^{\mathbb{C}} - T_{cp}T_{cm} = 0$$

that is,

$$(T_\infty^{\mathbb{C}})^2 + ((d-1)\beta + 1)T_\infty^{\mathbb{C}} - \beta = 0.$$

Solving the above quadratic equation, we obtain the following theorem.

THEOREM 1 *For a partitionable network, we have*

$$T_\infty^{\mathbb{C}} = \frac{1}{2}\left(\sqrt{(d-1)^2\beta^2 + 2(d+1)\beta + 1} - ((d-1)\beta + 1)\right).$$

Mt2 □ job no. 2148 □ ieee □ aerospace and electronic systems □ 2148D50 [3] □ (XXX) □ 11-30-10 □ 05:18 PM

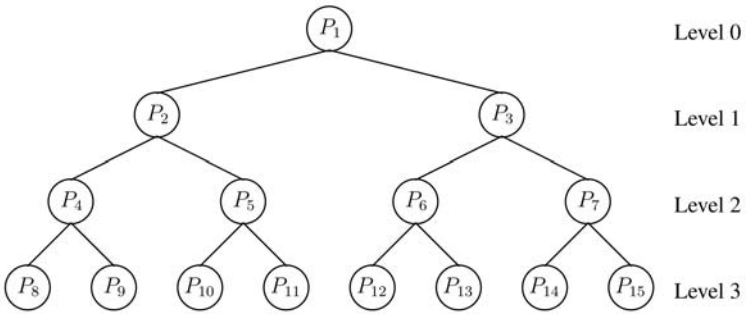

Fig. 1. Complete binary tree with $h = 3$.

Furthermore,

$$S_\infty^{\mathbb{C}} = \frac{\beta}{T_\infty^{\mathbb{C}}} = \frac{\sqrt{(d-1)^2\beta^2 + 2(d+1)\beta + 1} + ((d-1)\beta + 1)}{2} \approx (d-1)\beta$$

as $\beta \to \infty$.

A complete b-ary tree network of height $h > 0$ is partitionable at the root. The b children of the root are roots of b subtrees of height $h-1$. A processor that holds a load in the beginning is called an initial processor. Applying Theorem 1 to a complete b-ary tree network with $d = b$, we have the following result.

THEOREM 2 *For a complete b-ary tree network with the root as the initial processor, we have*

$$T_\infty^{\mathbb{C}} = \frac{1}{2}\left(\sqrt{(b-1)^2\beta^2 + 2(b+1)\beta + 1} - ((b-1)\beta + 1)\right).$$

Furthermore,

$$S_\infty^{\mathbb{C}} = \frac{\beta}{T_\infty^{\mathbb{C}}} = \frac{\sqrt{(b-1)^2\beta^2 + 2(b+1)\beta + 1} + ((b-1)\beta + 1)}{2} \approx (b-1)\beta$$

as $\beta \to \infty$.

A pyramid network is partitionable at the apex. Applying Theorem 1 to a pyramid network with $d = 4$, we have the following result.

THEOREM 3 *For a pyramid network with the apex as the initial processor, we have*

$$T_\infty^{\mathbb{C}} = \frac{1}{2}\left(\sqrt{9\beta^2 + 10\beta + 1} - (3\beta + 1)\right).$$

Furthermore,

$$S_\infty^{\mathbb{C}} = \frac{\beta}{T_\infty^{\mathbb{C}}} = \frac{\sqrt{9\beta^2 + 10\beta + 1} + (3\beta + 1)}{2} \approx 3\beta$$

as $\beta \to \infty$.

IV. PIPELINED COMMUNICATIONS

The main source of overhead that limits the performance of algorithm $\mathbb{C}$ is the long waiting time in distributing the load fractions. For instance, a child processor of the root must wait for α time to receive the load fraction α which is the total amount of load processed by a subtree. This motivates the technique of pipelined communications to improve the performance of divisible load distribution.

In this section, we discuss pipelined communications on complete binary tree networks. Let $h \geq 0$ denote the height of a complete binary tree network, which has levels $0, 1, 2, \ldots, h$, and $N = 2^{h+1} - 1$ processors $P_1, P_2, \ldots, P_N$ (see Fig. 1 for an example with $h = 3$ and $N = 15$).

Our first algorithm $\mathbb{P}$ which employs pipelined communications to distribute divisible loads on complete binary tree networks is illustrated in Fig. 2 where $h = 3$. Let x be a load fraction transmitted among the processors. The meaning of x will be clear shortly. Load fractions are distributed from level to level in multiples of x. Whenever a processor on level l receives a load fraction from its parent on level $l-1$, the processor splits the load fraction and sends half of the load fraction to each of its two children on level $l+1$.

At time 0, the initial processor (i.e., the root processor P_1 on level 0) sends a load fraction $4x$ to each of P_2 and P_3 (i.e., the two processors on level 1). At time $T' = 4x$, processors on level 1 receive the load fraction $4x$. At time $T'' = 6x$, processors on level 2 receive the load fraction $2x$ and processors on level 1 receive the second load fraction $2x$. At time $T''' = 7x$, processors on level 3 receive the load fraction x and processors on level 2 receive the second load fraction x and processors on level 1 receive the third load fraction x. Processor P_1 processes $4x/\beta$, $2x/\beta$, and x/β amount of load during each of the time intervals $\tau_1 = [0, T')$, $\tau_2 = [T', T'')$, and $\tau_3 = [T'', T''')$. During the time interval $\tau_4 = [T''', T_{15}^{\mathbb{P}})$, all the processors process x amount of load. Note that the quantities in boldface in Fig. 2 represent the amount of load communicated not computed. The value of x is chosen such that the total amount of load processed by all the N processors is one.

Mt2 □ job no. 2148 □ ieee □ aerospace and electronic systems □ 2148D50 [4] □ (XXX) □ 11-30-10 □ 05:18 PM

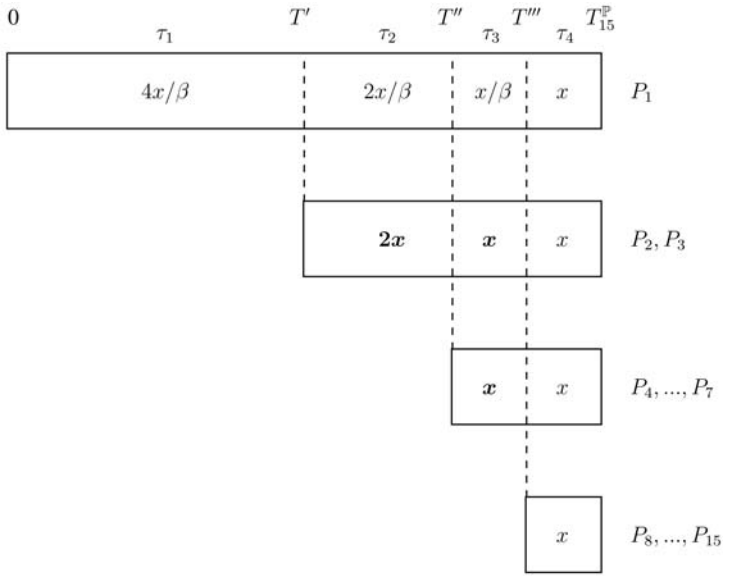

Fig. 2. Illustration of algorithm $\mathbb{P}$ ($h = 3$).

```
the length of τ_k = 2^{h-k}x, for all 1 ≤ k ≤ h
the length of τ_{h+1} = xβ, where x = 2β/((2β+1)N − 1)
for (time interval τ_k, 1 ≤ k ≤ h+1) do
    if (k < h+1)
        for (j = 1; j < 2^{k+1}; j++) do in parallel
            if (j = 1)
                send 2^{h-k}x amount of load to each child
                and compute 2^{h-k}x/β amount of load
            else if (2 ≤ j < 2^k)
                send 2^{h-k}x amount of load to each child
                and receive 2^{h-k}x amount of load from parent
            else if (2^k ≤ j < 2^{k+1})
                receive 2^{h-k}x amount of load from parent
            end if
        end do
    else
        for (j = 1; j ≤ N; j++) do in parallel
            compute x amount of load
        end do
    end if
end do
```

Fig. 3. Algorithm $\mathbb{P}$.

In general, the time interval $[0, T_N^{\mathbb{P}})$ is divided into $h+1$ slots $\tau_1, \tau_2, \ldots, \tau_{h+1}$, where τ_k has length $2^{h-k}x$ for all $1 \le k \le h$ basically used for communicating load fractions, and τ_{h+1} has length $x\beta$ used for computation. During τ_k, where $1 \le k \le h$, the root processor initiates the pipelined communication of a load fraction $2^{h-k}x$ to each of its two children, and the load fraction $2^{h-k}x$ will be split during communication and eventually reaches level $(h+1-k)$ for computation. During τ_{h+1}, all the N processors compute the same amount of load x. Processor P_1 also computes $2^{h-k}x/\beta$ amount of load during τ_k for all $1 \le k \le h$. Algorithm $\mathbb{P}$ is formally described in Fig. 3, where we specify the computations and communications performed by each processor during each time interval.

Let L_l denote the amount of load processed by a processor on level l, where $0 \le l \le h$. Then, we have

$$L_0 = (2^{h-1} + 2^{h-2} + \cdots + 2^0)\frac{x}{\beta} + x = \left(\frac{N-1}{2}\right)\frac{x}{\beta} + x$$

and $L_l = x$ for all $1 \le l \le h$. Since

$$1 = \sum_{l=0}^{h} 2^l L_l = \left(\frac{N-1}{2}\right)\frac{x}{\beta} + Nx = 1$$

we get

$$x = \frac{2\beta}{(2\beta+1)N-1}.$$

Therefore, the parallel processing time is

$$T_N^{\mathbb{P}} = L_0\beta = \left(\frac{N-1}{2} + \beta\right)x = \frac{(N+2\beta-1)\beta}{(2\beta+1)N-1}.$$

The speedup is

$$S_N^{\mathbb{P}} = \frac{T_1}{T_N^{\mathbb{P}}} = \frac{T_{cp}}{T_N^{\mathbb{P}}} = \frac{\beta}{T_N^{\mathbb{P}}} = \frac{(2\beta+1)N-1}{N+2\beta-1}.$$

It is clear that the asymptotic speedup is $S_\infty^{\mathbb{P}} = 2\beta + 1$. The above discussion is summarized as the following theorem.

THEOREM 4 *For a complete binary tree network with the root as the initial processor, we have*

$$T_N^{\mathbb{P}} = \frac{(N+2\beta-1)\beta}{(2\beta+1)N-1}.$$

Furthermore,

$$S_N^{\mathbb{P}} = \frac{\beta}{T_N^{\mathbb{P}}} = \frac{(2\beta+1)N-1}{N+2\beta-1}$$

and $S_\infty^{\mathbb{P}} = 2\beta + 1$.

Mt2 □ job no. 2148 □ ieee □ aerospace and electronic systems □ 2148D50 [5] □ (XXX) □ 11-30-10 □ 05:18 PM

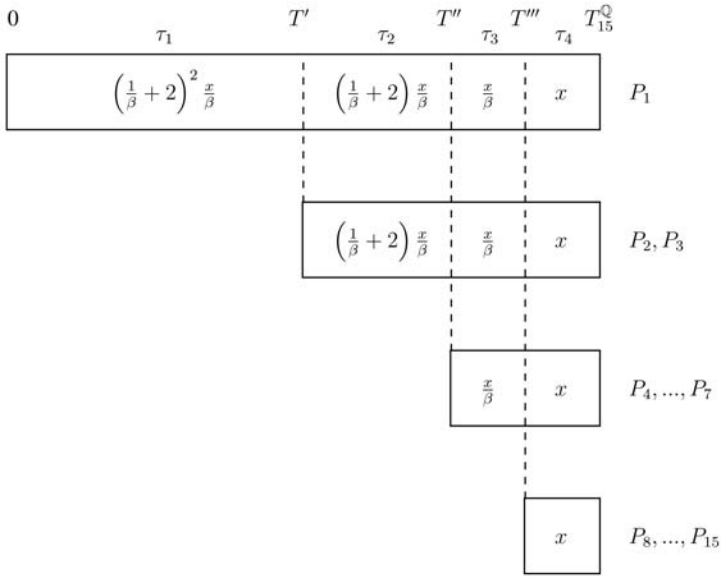

Fig. 4. Illustration of algorithm $\mathbb{Q}$ ($h = 3$).

the length of $\tau_k = (1/\beta + 2)^{h-k}x$, for all $1 \le k \le h$
the length of $\tau_{h+1} = x\beta$, where

$$x = \left(\frac{\beta}{\beta+1}\left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1\right)\right)^{-1}$$

```
for (time interval τ_k, 1 ≤ k ≤ h + 1) do
    if (k < h + 1)
        for (j = 1; j < 2^{k+1}; j++) do in parallel
            if (1 ≤ j < 2^k)
                send (1/β + 2)^{h-k}x amount of load to each child
                and receive (1/β + 2)^{h-k}x amount of load from parent if j > 1
                and compute (1/β)(1/β + 2)^{h-k}x amount of load
            else if (2^k ≤ j < 2^{k+1})
                receive (1/β + 2)^{h-k}x amount of load from parent
            end if
        end do
    else
        for (j = 1; j ≤ N; j++) do in parallel
            compute x amount of load
        end do
    end if
end do
```

Fig. 5. Algorithm $\mathbb{Q}$.

V. OVERLAP OF COMMUNICATIONS AND COMPUTATIONS

In the model of divisible load distribution, a processor can send a load fraction, receive another load fraction, and yet compute a third load fraction at the same time. Our second algorithm $\mathbb{Q}$ that uses pipelined communications and overlaps communications and computations is illustrated in Fig. 4. During the time interval $\tau_1 = [0, T')$, the root processor sends a load fraction $(1/\beta + 2)^2 x$ to each of the processors on level 1. During the time interval $\tau_2 = [T', T'')$, a processor on level 1 does not process the load $(1/\beta + 2)^2 x$ by itself nor sends it to level 2. Instead, it keeps the load fraction $(1/\beta)(1/\beta + 2)x$ for computing and sends the load fraction $(1/\beta + 2)x$ to each of its two children on level 2, and at the same time, receives the load fraction $(1/\beta + 2)x$ from its parent on level 0. During the time interval $\tau_3 = [T'', T''')$, a processor on level 2 keeps the load fraction x/β for computing and sends the load fraction x to each of its two children on level 3, and at the same time, receives the load fraction x from its parent on level 1, which in turn, is receiving the load fraction x from its parent on level 0. During the time interval $\tau_4 = [T''', T_{15}^{\mathbb{Q}})$, every processor computes a load fraction x.

A complete description of algorithm $\mathbb{Q}$ is given in Fig. 5. It is clear that algorithm $\mathbb{Q}$ divides the

time interval $[0, T_N^{\mathbb{Q}})$ into subintervals $\tau_1, \tau_2, \ldots, \tau_{h+1}$, such that the length of τ_k is $(1/\beta+2)^{h-k}x$ for all $1 \le k \le h$, and the length of τ_{h+1} is $x\beta$. During τ_k, $1 \le k \le h$, the amount of load communicated from level l to level $l+1$ is $(1/\beta+2)^{h-k}x$ for all $0 \le l \le k-1$.

Since a processor on level l, where $0 \le l \le h$, computes $(1/\beta)(1/\beta+2)^{h-k}x$ amount of load during τ_k for all $l+1 \le k \le h$, and x amount of load during τ_{h+1}, we have

$$\begin{aligned} L_l &= \left(\left(\left(\frac{1}{\beta}+2\right)^{h-l-1} + \left(\frac{1}{\beta}+2\right)^{h-l-2} \right.\right. \\ &\qquad \left.\left. + \cdots + \left(\frac{1}{\beta}+2\right)^{0} \right) \frac{1}{\beta} + 1 \right) x \\ &= \left(\frac{\left(\frac{1}{\beta}+2\right)^{h-l} - 1}{\left(\frac{1}{\beta}+2\right) - 1} \cdot \frac{1}{\beta} + 1 \right) x \\ &= \left(\frac{1}{\beta+1} \left(\left(\frac{1}{\beta}+2\right)^{h-l} - 1 \right) + 1 \right) x. \end{aligned}$$

Since

$$\begin{aligned} 1 &= \sum_{l=0}^{h} 2^l L_l \\ &= \sum_{l=0}^{h} 2^l \left(\frac{1}{\beta+1} \left(\left(\frac{1}{\beta}+2\right)^{h-l} - 1 \right) + 1 \right) x \\ &= \left(\frac{1}{\beta+1} \sum_{l=0}^{h} 2^l \left(\frac{1}{\beta}+2\right)^{h-l} + \frac{\beta}{\beta+1} \sum_{l=0}^{h} 2^l \right) x \\ &= \left(\frac{1}{\beta+1} \left(\frac{1}{\beta}+2\right)^{h} \sum_{l=0}^{h} \left(\frac{2\beta}{2\beta+1}\right)^{l} + \frac{\beta}{\beta+1} \sum_{l=0}^{h} 2^l \right) x \\ &= \left(\frac{(2\beta+1)^{h+1} - (2\beta)^{h+1}}{(\beta+1)\beta^h} + \left(\frac{\beta}{\beta+1}\right)(2^{h+1}-1) \right) x \\ &= \left(\frac{(2\beta+1)^{h+1} - (2\beta)^{h+1} + \beta^{h+1}(2^{h+1}-1)}{(\beta+1)\beta^h} \right) x \\ &= \left(\frac{(2\beta+1)^{h+1} - \beta^{h+1}}{(\beta+1)\beta^h} \right) x \\ &= \frac{\beta}{\beta+1} \left(\left(\frac{1}{\beta}+2\right)^{h+1} - 1 \right) x \\ &= \frac{\beta}{\beta+1} \left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1 \right) x \end{aligned}$$

we obtain

$$x = \left(\frac{\beta}{\beta+1} \left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1 \right) \right)^{-1}.$$

The parallel processing time is

$$\begin{aligned} T_N^{\mathbb{Q}} &= L_0 \beta \\ &= \frac{\frac{1}{\beta+1}\left(\left(\frac{1}{\beta}+2\right)^{h} - 1 \right) + 1}{\frac{\beta}{\beta+1}\left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1 \right)} \cdot \beta \\ &= \frac{\left(\frac{1}{\beta}+2\right)^{\log(N+1)-1} + \beta}{\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1}. \end{aligned}$$

The speedup is

$$\begin{aligned} S_N^{\mathbb{Q}} &= \frac{T_1}{T_N^{\mathbb{Q}}} \\ &= \frac{T_{cp}}{T_N^{\mathbb{Q}}} \\ &= \frac{\beta}{T_N^{\mathbb{Q}}} \\ &= \frac{\beta\left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1 \right)}{\left(\frac{1}{\beta}+2\right)^{\log(N+1)-1} + \beta} \end{aligned}$$

and the asymptotic speedup is

$$Begineq. S_\infty^{\mathbb{Q}} = \beta\left(\frac{1}{\beta}+2\right) = 2\beta+1.$$

The above discussion is summarized as the following theorem.

THEOREM 5 *For a complete binary tree network with the root as the initial processor, we have*

$$T_N^{\mathbb{Q}} = \frac{\left(\frac{1}{\beta}+2\right)^{\log(N+1)-1} + \beta}{\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1}.$$

Furthermore,

$$S_N^{\mathbb{Q}} = \frac{\beta}{T_N^{\mathbb{Q}}} = \frac{\beta\left(\left(\frac{1}{\beta}+2\right)^{\log(N+1)} - 1 \right)}{\left(\frac{1}{\beta}+2\right)^{\log(N+1)-1} + \beta}$$

and $S_\infty^{\mathbb{Q}} = 2\beta+1$.

Mt2 □ job no. 2148 □ ieee □ aerospace and electronic systems □ 2148D50 [7] □ (XXX) □ 11-30-10 □ 05:18 PM

VI. COMPLETE TREE NETWORKS

The analysis of algorithm $\mathbb{P}$ for divisible load distribution on a general complete b-ary tree network with height h is a straightforward extension of that in Section IV. We notice that the length of the time interval τ_k is $b^{h-k}x$ for all $1 \le k \le h$. During τ_k, where $1 \le k \le h$, the root processor initiates the pipelined communication of a load fraction $b^{h-k}x$ to each of its b children, and the load fraction $b^{h-k}x$ will be split during communication and eventually reaches level $(h+1-k)$ for computation. Hence,

$$L_0 = (b^{h-1} + b^{h-2} + \cdots + b^0)\frac{x}{\beta} + x$$
$$= \left(\frac{N-1}{b}\right)\frac{x}{\beta} + x$$

and $L_l = x$ for all $1 \le l \le h$. This leads to

$$x = \frac{b\beta}{(b\beta+1)N-1}$$

and the following theorem.

THEOREM 6 *For a complete b-ary tree network with the root as the initial processor, we have*

$$T_N^{\mathbb{P}} = \frac{(N+b\beta-1)\beta}{(b\beta+1)N-1}.$$

Furthermore,

$$S_N^{\mathbb{P}} = \frac{\beta}{T_N^{\mathbb{P}}} = \frac{(b\beta+1)N-1}{N+b\beta-1},$$

and $S_\infty^{\mathbb{P}} = b\beta + 1$.

For algorithm $\mathbb{Q}$, the length of the time interval τ_k is $(1/\beta+b)^{h-k}x$ for all $1 \le k \le h$, such that upon receiving the load fraction $(1/\beta+b)^{h-k}x$ by a processor, $(1/\beta)(1/\beta+b)^{h-k-1}x$ amount of load is kept for computation, and $(1/\beta+b)^{h-k-1}x$ amount of load is sent to each of its b children. Therefore, for $0 \le l \le h$, we get

$$L_l = \left(\left(\left(\frac{1}{\beta}+b\right)^{h-l-1} + \left(\frac{1}{\beta}+b\right)^{h-l-2} + \cdots + \left(\frac{1}{\beta}+b\right)^{0}\right)\frac{1}{\beta}+1\right)x$$
$$= \left(\frac{1}{(b-1)\beta+1}\left(\left(\frac{1}{\beta}+b\right)^{h-l}-1\right)+1\right)x.$$

The last equation gives rise to

$$x = \left(\frac{\beta}{(b-1)\beta+1}\left(\left(\frac{1}{\beta}+b\right)^{\log_b((b-1)N+1)}-1\right)\right)^{-1}$$

and the following theorem.

THEOREM 7 *For a complete b-ary tree network with the root as the initial processor, we have*

$$T_N^{\mathbb{Q}} = \frac{\left(\frac{1}{\beta}+b\right)^{\log_b((b-1)N+1)-1} + (b-1)\beta}{\left(\frac{1}{\beta}+b\right)^{\log_b((b-1)N+1)} - 1}.$$

Furthermore,

$$S_N^{\mathbb{Q}} = \frac{\beta}{T_N^{\mathbb{Q}}} = \frac{\beta\left(\left(\frac{1}{\beta}+b\right)^{\log_b((b-1)N+1)}-1\right)}{\left(\frac{1}{\beta}+b\right)^{\log_b((b-1)N+1)-1} + (b-1)\beta}$$

and $S_\infty^{\mathbb{Q}} = b\beta + 1$.

VII. PYRAMID NETWORKS

A pyramid network contains a complete 4-ary tree as a subnetwork. Applying Theorems 6 and 7 to a pyramid network with $b = 4$, we have the following results.

THEOREM 8 *For a pyramid network with the apex as the initial processor, we have*

$$T_N^{\mathbb{P}} = \frac{(N+4\beta-1)\beta}{(4\beta+1)N-1}.$$

Furthermore,

$$S_N^{\mathbb{P}} = \frac{\beta}{T_N^{\mathbb{P}}} = \frac{(4\beta+1)N-1}{N+4\beta-1}$$

and $S_\infty^{\mathbb{P}} = 4\beta + 1$.

THEOREM 9 *For a pyramid network with the apex as the initial processor, we have*

$$T_N^{\mathbb{Q}} = \frac{\left(\frac{1}{\beta}+4\right)^{\log_4(3N+1)-1} + 3\beta}{\left(\frac{1}{\beta}+4\right)^{\log_4(3N+1)} - 1}.$$

Furthermore,

$$S_N^{\mathbb{Q}} = \frac{\beta}{T_N^{\mathbb{Q}}} = \frac{\beta\left(\left(\frac{1}{\beta}+4\right)^{\log_4(3N+1)}-1\right)}{\left(\frac{1}{\beta}+4\right)^{\log_4(3N+1)-1} + 3\beta}$$

and $S_\infty^{\mathbb{Q}} = 4\beta + 1$.

VIII. PERFORMANCE COMPARISON

Finally, we prove the following result.

THEOREM 10 *For a complete b-ary tree network with the root as the initial processor and a pyramid network with the apex as the initial processor, we have $T_N^{\mathbb{C}} > T_N^{\mathbb{P}} > T_N^{\mathbb{Q}}$ and $S_N^{\mathbb{C}} < S_N^{\mathbb{P}} < S_N^{\mathbb{Q}}$ for all $b \ge 2$ and $h \ge 2$.*

First, we prove by induction on $h \geq 2$ that $T_N^{\mathbb{C}} > T_N^{\mathbb{P}}$, where $T_N^{\mathbb{C}}$ is given by

$$T_1^{\mathbb{C}} = \beta$$

$$T_N^{\mathbb{C}} = \left(\frac{T_{(N-1)/b}^{\mathbb{C}} + 1}{T_{(N-1)/b}^{\mathbb{C}} + b\beta + 1} \right) \beta, \qquad N > 1$$

and

$$T_N^{\mathbb{P}} = \frac{(b\beta + N - 1)\beta}{bN\beta + N - 1}.$$

When $h = 2$ and $N = b^2 + b + 1$, it is easy to verify that

$$T_N^{\mathbb{C}} = \left(\frac{\beta^2 + (b+2)\beta + 1}{(b^2+b+1)\beta^2 + 2(b+1)\beta + 1} \right) \beta$$

and

$$T_N^{\mathbb{P}} = \left(\frac{\beta + (b+1)}{(b^2+b+1)\beta + (b+1)} \right) \beta$$

and $T_N^{\mathbb{C}} > T_N^{\mathbb{P}}$. When $h > 2$, we have

$$\begin{aligned} T_N^{\mathbb{C}} &= \left(\frac{T_{(N-1)/b}^{\mathbb{C}} + 1}{T_{(N-1)/b}^{\mathbb{C}} + b\beta + 1} \right) \beta \\ &> \left(\frac{T_{(N-1)/b}^{\mathbb{P}} + 1}{T_{(N-1)/b}^{\mathbb{P}} + b\beta + 1} \right) \beta \\ &\qquad \text{(by the induction hypothesis)} \\ &= \left(\frac{\frac{(b\beta + (N-1)/b - 1)\beta}{(N-1)\beta + (N-1)/b - 1} + 1}{\frac{(b\beta + (N-1)/b - 1)\beta}{(N-1)\beta + (N-1)/b - 1} + b\beta + 1} \right) \beta \\ &= \left(\frac{b^2\beta^2 + (bN + N - 2b - 1)\beta + (N - b - 1)}{b^2N\beta^2 + (2bN + N - b^2 - 3b - 1)\beta + (N - b - 1)} \right) \beta \\ &> \left(\frac{b\beta + N - 1}{bN\beta + N - 1} \right) \beta \\ &= T_N^{\mathbb{P}}. \end{aligned}$$

Next, we show that $S_N^{\mathbb{P}} < S_N^{\mathbb{Q}}$, namely,

$$\frac{(b\beta + 1)N - 1}{N + b\beta - 1} < \frac{\beta(X-1)}{\frac{X}{\frac{1}{\beta} + b} + (b-1)\beta}$$

where

$$X = \left(\frac{1}{\beta} + b \right)^{h+1}$$

and

$$N = \frac{b^{h+1} - 1}{b - 1}.$$

By straightforward algebraic manipulations, the above inequality can be rewritten as

$$((b-1)\beta + 1)N + \beta - 1 < \left(\frac{b\beta^2}{b\beta + 1} \right) X.$$

Substituting X and N, we get

$$\begin{aligned} &((b-1)\beta + 1)\left(\frac{b^{h+1} - 1}{b - 1} \right) + \beta - 1 \\ &\qquad < \left(\frac{b\beta^2}{b\beta + 1} \right) \left(\frac{b\beta + 1}{\beta} \right)^{h+1} \end{aligned}$$

that is,

$$((b-1)\beta + 1)\left(\frac{b^{h+1} - 1}{b - 1} \right) + \beta - 1 < \frac{b}{\beta^{h-1}}(b\beta + 1)^h$$

which can be simplified as

$$(b-1)(b\beta)^h + (b^h - 1)\beta^{h-1} < (b-1)(b\beta + 1)^h.$$

Since

$$(b\beta + 1)^h = (b\beta)^h + h(b\beta)^{h-1} + \cdots$$

it suffices to show

$$(b-1)(b\beta)^h + (b^h - 1)\beta^{h-1} < (b-1)((b\beta)^h + h(b\beta)^{h-1})$$

that is,

$$b^h - 1 < (b-1)hb^{h-1}$$

or

$$b^{h-1} + b^{h-2} + \cdots + 1 < hb^{h-1}.$$

The last inequality is obvious for all $b \geq 2$ and $h \geq 2$.

In Fig. 6, we demonstrate the numerical values of the speedup of the algorithms considered in this paper for a complete binary tree network and a pyramid network. It is assumed that $\beta = 100$. $S_N^{\mathbb{Q}}$ is not displayed since its difference from $S_N^{\mathbb{P}}$ is not noticeable in the figure.

In Tables I and II, we demonstrate numerical values of parallel processing times of the three algorithms considered in this paper for a complete binary tree network ($b = 2$) and a pyramid network ($b = 4$). It is assumed that $\beta = 100$. Compared with the classic algorithm $\mathbb{C}$, algorithm $\mathbb{P}$ is 97% faster on a complete binary tree network and 33% faster on a pyramid network when $h = 15$. Such performance improvement is due to the technique of pipelined communications. These numerical data are consistent with our analytical results. Algorithm $\mathbb{Q}$ is a little faster than algorithm $\mathbb{P}$. This is due to the technique of overlap of communications and computations and consistent with our analytical result in Theorem 10.

IX. CONCLUSIONS

We have proposed two new methods which employ pipelined communications to distribute divisible loads on tree and pyramid networks. We derived closed-form expressions of the parallel time and speedup for both methods and showed that the asymptotic speedup of both methods is $b\beta + 1$ for a complete b-ary tree network and $4\beta + 1$ for a pyramid network. The technique of pipelined communications leads to improved performance of divisible load

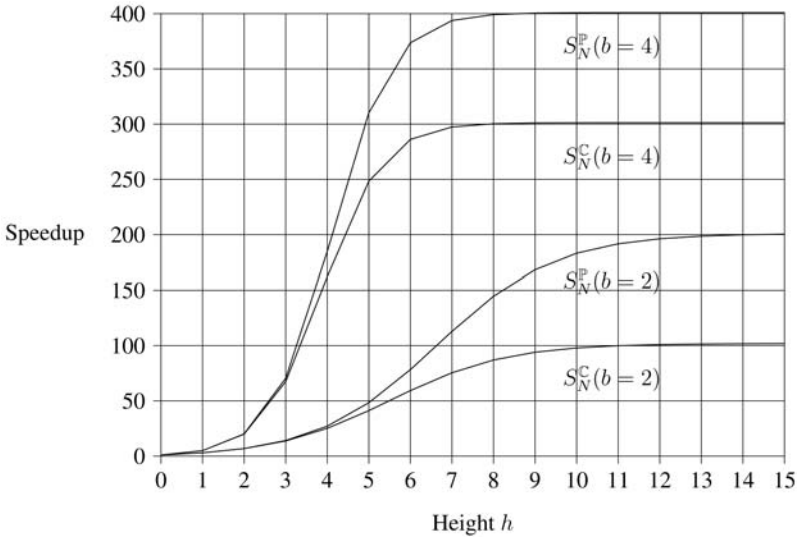

Fig. 6. Speedup of various algorithms ($\beta = 100$).

TABLE I
Parallel Processing Times ($b = 2$, $\beta = 100$)

h	Algorithm $\mathbb{C}$	Algorithm $\mathbb{P}$	Algorithm $\mathbb{Q}$
0	100.00000	100.00000	100.00000
1	33.55482	33.55482	33.55482
2	14.73209	14.65149	14.61114
3	7.29242	7.10020	7.05637
4	3.98114	3.69181	3.65701
5	2.43005	2.06918	2.04496
6	1.68611	1.27713	1.26147
7	1.32525	0.88578	0.87615
8	1.14927	0.69127	0.68553
9	1.06321	0.59429	0.59097
10	1.02107	0.54588	0.54399
11	1.00043	0.52169	0.52063
12	0.99031	0.50960	0.50901
13	0.98535	0.50356	0.50323
14	0.98292	0.50053	0.50036
15	0.98173	0.49902	0.49893

TABLE II
Parallel Processing Times ($b = 4$, $\beta = 100$)

h	Algorithm $\mathbb{C}$	Algorithm $\mathbb{P}$	Algorithm $\mathbb{Q}$
0	100.00000	100.00000	100.00000
1	20.15968	20.15968	20.15968
2	5.02415	4.98812	4.97911
3	1.48369	1.42002	1.41508
4	0.61709	0.54117	0.53923
5	0.40265	0.32227	0.32161
6	0.34944	0.26760	0.26739
7	0.33622	0.25393	0.25387
8	0.33294	0.25052	0.25050
9	0.33213	0.24966	0.24966
10	0.33193	0.24945	0.24945
11	0.33188	0.24939	0.24939
12	0.33186	0.24938	0.24938
13	0.33186	0.24938	0.24938
14	0.33186	0.24938	0.24938
15	0.33186	0.24938	0.24938

distribution on tree and pyramid networks. Compared with the classic method, the asymptotic speedup of our new methods is 100% faster on a complete binary tree network and 33% faster on a pyramid network for large β.

APPENDIX. A CLOSED-FORM SOLUTION FOR $T_N^{\mathbb{C}}$

We now derive a closed-form solution to the parallel time of the classic method for processing divisible loads on partitionable networks. Our discussion in Section III implies that a partitionable static interconnection network contains a d-ary complete tree as a subnetwork which is used for load distribution. We divide such a network into $m+1$ layers, where layer m contains the initial processor (i.e., the root, or level 0, of the d-ary complete tree), and layer $m-1$ contains neighbors of the initial processor (i.e., level 1 of the d-ary complete tree), and layer 0 contains the leaves of the d-ary complete tree. It is clear that the number of processors is

$$N = 1 + d + d^2 + \cdots + d^m = \frac{d^{m+1}}{d-1}.$$

Let $T_m^{\mathbb{C}}$ denote the parallel processing time of a unit load on the above partitionable network with N processors and layers $0,1,2,\ldots,m$, by using the classic algorithm $\mathbb{C}$. Clearly, we have $T_N^{\mathbb{C}} = T_m^{\mathbb{C}}$ and we will find a closed-form solution to $T_m^{\mathbb{C}}$.

Assume that a unit load is processed during the time interval $[0, T_m^{\mathbb{C}})$. Let L_j denote the fraction of this unit load processed by a processor in layer j, where $0 \le j \le m$. By the definition of L_j, we have

$$d^m L_0 + d^{m-1} L_1 + d^{m-2} L_2 + \cdots + d L_{m-1} + L_m = 1.$$

We illustrate algorithm $\mathbb{C}$ in Fig. 7, where $m = 3$. The quantities in the figure are load amounts processed by the processors during various time intervals.

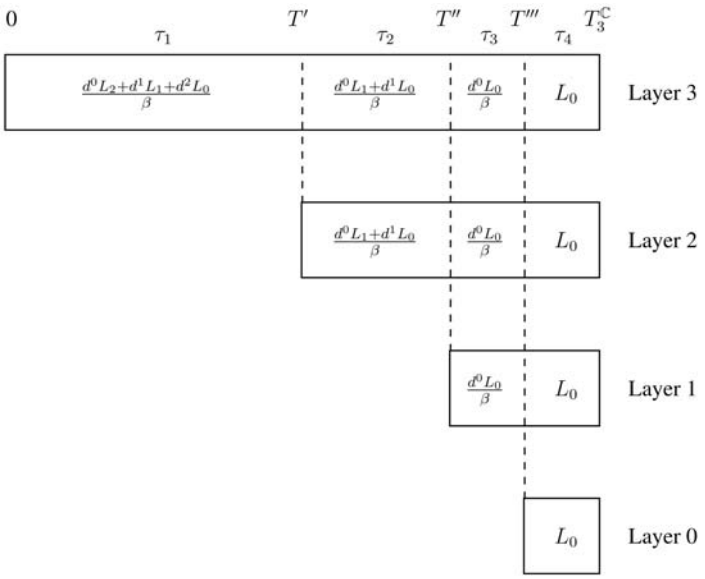

Fig. 7. Illustration of algorithm $\mathbb{C}$ ($m = 3$).

A processor in layer 2 receives the load fraction sent by the initial processor at time $T' = d^0L_2 + d^1L_1 + d^2L_0$. A processor in layer 1 receives the load fraction sent by a processor in layer 2 at time $T'' = T' + d^0L_1 + d^1L_0$. A processor in layer 0 receives the load fraction sent by a processor in layer 1 at time $T''' = T'' + L_0$. During the time interval $[0,T')$, the initial processor computes $(d^0L_2 + d^1L_1 + d^2L_0)/\beta$ amount of load, since during the same time interval, a load fraction $d^0L_2 + d^1L_1 + d^2L_0$ is being sent from the initial processor to a processor in layer 2. During the time interval $[T',T'')$, each processor in layers 3 and 2 computes $(d^0L_1 + d^1L_0)/\beta$ amount of load, since during the same time interval, a load fraction $d^0L_1 + d^1L_0$ is being sent from a processor in layer 2 to a processor in layer 1. During the time interval $[T'',T''')$, each processor in layers 3, 2, and 1 computes L_0/β amount of load, since during the same time interval, a load fraction L_0 is being sent from a processor in layer 1 to a processor in layer 0. During the time interval $[T''',T_3^{\mathbb{C}})$, each processor in layers 3, 2, 1, and 0 computes L_0 amount of load.

It is clear that a processor in layer m sends $d^0L_{m-1} + d^1L_{m-2} + \cdots + d^{m-1}L_0$ amount of load to a processor in layer $m-1$. Before a processor in layer $m-1$ receives the above load, a processor in layer m can process

$$\frac{d^0L_{m-1} + d^1L_{m-2} + \cdots + d^{m-1}L_0}{\beta}$$

amount of load. After the load fraction $d^0L_{m-1} + d^1L_{m-2} + \cdots + d^{m-1}L_0$ is received by a processor in layer $m-1$, each processor in both layers m and $m-1$ processes L_{m-1} amount of load in the remaining computation. Assume that $L_0 = x$, where x is a value be defined later. The above discussion suggests the following recurrence relation,

$$L_0 = x$$

$$L_m = \frac{d^0L_{m-1} + d^1L_{m-2} + \cdots + d^{m-1}L_0}{\beta} + L_{m-1}, \qquad m \geq 1$$

where x is chosen such that $d^mL_0 + d^{m-1}L_1 + d^{m-2}L_2 + \cdots + dL_{m-1} + L_m = 1$.

To solve the above recurrence relation, we notice that

$$L_{m-1} = \frac{d^0L_{m-2} + d^1L_{m-3} + \cdots + d^{m-2}L_0}{\beta} + L_{m-2}$$

and

$$dL_{m-1} = \frac{d^1L_{m-2} + d^2L_{m-3} + \cdots + d^{m-1}L_0}{\beta} + dL_{m-2}.$$

Therefore, we get

$$L_m - dL_{m-1} = \frac{L_{m-1}}{\beta} + L_{m-1} - dL_{m-2}.$$

Hence, we obtain the following homogeneous linear recurrence relation,

$$L_0 = x$$

$$L_1 = \left(\frac{1}{\beta} + 1\right)x$$

$$L_m = \left(\frac{1}{\beta} + d + 1\right)L_{m-1} - dL_{m-2}, \qquad m \geq 2.$$

The characteristic equation of the above linear recurrence relation is

$$r^2 - \left(\frac{1}{\beta} + d + 1\right)r + d = 0$$

or

$$\beta r^2 - ((d+1)\beta + 1)r + d\beta = 0$$

with roots

$$r_1 = \frac{((d+1)\beta+1) + \sqrt{((d+1)\beta+1)^2 - 4d\beta^2}}{2\beta}$$

and

$$r_2 = \frac{((d+1)\beta+1) - \sqrt{((d+1)\beta+1)^2 - 4d\beta^2}}{2\beta}.$$

Consequently, L_m can be represented as

$$L_m = pr_1^m + qr_2^m$$

where p and q satisfy

$$p + q = L_0 = x$$

and

$$pr_1 + qr_2 = L_1 = \left(\frac{1}{\beta} + 1\right)x.$$

Solving the above equations, we get

$$p = \left(\frac{\beta + 1 - \beta r_2}{\beta(r_1 - r_2)}\right)x$$

and

$$q = \left(\frac{\beta r_1 - \beta - 1}{\beta(r_1 - r_2)}\right)x.$$

Based on the condition

$$\sum_{j=0}^{m} d^{m-j} L_j = 1$$

we have

$$\sum_{j=0}^{m} d^{m-j}(pr_1^j + qr_2^j) = 1$$

that is,

$$\sum_{j=0}^{m}\left(p\left(\frac{r_1}{d}\right)^j + q\left(\frac{r_2}{d}\right)^j\right) = \frac{1}{d^m}$$

or

$$\left(\left(\frac{\beta + 1 - \beta r_2}{\beta(r_1 - r_2)}\right)\left(\frac{(r_1/d)^{m+1} - 1}{r_1/d - 1}\right) + \left(\frac{\beta r_1 - \beta - 1}{\beta(r_1 - r_2)}\right)\left(\frac{(r_2/d)^{m+1} - 1}{r_2/d - 1}\right)\right)x = \frac{1}{d^m}$$

which implies that

$$x = \left(\left(\left(\frac{\beta + 1 - \beta r_2}{\beta(r_1 - r_2)}\right)\left(\frac{(r_1/d)^{m+1} - 1}{r_1/d - 1}\right) + \left(\frac{\beta r_1 - \beta - 1}{\beta(r_1 - r_2)}\right)\left(\frac{(r_2/d)^{m+1} - 1}{r_2/d - 1}\right)\right)d^m\right)^{-1}.$$

Finally, the parallel processing time of a unit load on a partitionable network with N processors is $T_N^{\mathbb{C}} = T_m^{\mathbb{C}} = L_m T_{cp} = L_m \beta$, that is,

$$T_N^{\mathbb{C}} = \left(\left(\frac{\beta + 1 - \beta r_2}{\beta(r_1 - r_2)}\right)r_1^m + \left(\frac{\beta r_1 - \beta - 1}{\beta(r_1 - r_2)}\right)r_2^m\right)x\beta$$

where $m = \log_d N(d-1) - 1$.

REFERENCES

[1] http://www.ece.sunysb.edu/∼tom/dlt.html.

[2] Agrawal, R. and Jagadish, H. V.
Partitioning techniques for large grained parallelism.
IEEE Transactions on Computers, **37**, 12 (1988), 1627–1634.

[3] Amdahl, G. M.
Validity of the single processor approach to achieving large scale computing capabilities.
In *Proceedings of the AFIPS Spring Joint Computer Conference*, vol. 30, 1967, 483–485.

[4] Barlas, G. D.
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.

[5] Bataineh, S. M.
Towards analytical solution to task allocation, processor assignment, and performance evaluation of network of processors.
Journal of Parallel and Distributed Computing, **65** (2005), 29–47.

[6] Bataineh, S., Hsiung, T-Y., and Robertazzi, T. G.
Closed form solutions for bus and tree networks of processors load sharing a divisible job.
IEEE Transactions on Computers, **43**, 10 (1994), 1184–1196.

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

[8] Bataineh, S. and Robertazzi, T. G.
Performance limits for processor networks with divisible jobs.
IEEE Transactions on Aerospace and Electronic Systems, **33**, 4 (1997), 1189–1198.

[9] Bharadwaj, V., Ghose, D., and Mani, V.
An efficient load distribution strategy for a distributed linear network of processors with communication delays.
Computers and Mathematics with Applications, **29**, 9 (1995), 95–112.

[10] Bharadwaj, V., Ghose, D., and Mani, V.
Multi-installment load distribution in tree networks with delays.
IEEE Transactions on Aerospace and Electronic Systems, **31**, 2 (1995), 555–567.

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

[12] Bharadwaj, V., Li, X., and Ko, C. C.
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.

[13] Bharadwaj, V. and Ranganath, S.
Theoretical and experimental study of large size image processing applications using divisible load paradigm on distributed bus networks.
Image and Vision Computing, **20**, 13–14 (2002), 917–936.

[14] Błażaewicz, J. and Drozdowski, M.
Scheduling divisible jobs on hypercubes.
Parallel Computing, **21** (1995), 1945–1956.

[15] Błażewicz, J. and Drozdowski, M.
The performance limits of a two-dimensional network of load sharing processors.
Foundations of Computing and Decision Sciences, **21**, 1 (1996), 3–15.

[16] Błażewicz, J. and Drozdowski, M.
Distributed processing of divisible jobs with communication startup costs.
Discrete Applied Mathematics, **76**, 1–3 (1997), 21–41.

[17] Błażewicz, J., Drozdowski, M., Guinard, F., and Trystram, D.
Scheduling a divisible task in a two-dimensional toroidal mesh.
Discrete Applied Mathematics, **94**, 1–3 (1999), 35–50.

[18] Błażewicz, J., Drozdowski, M., and Markiewicz, M.
Divisible task scheduling—Concept and verification.
Parallel Computing, **25**, 1 (1999), 87–98.

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

[20] Cheng, Y. C. and Robertazzi, T. G.
Distributed computation for a tree network with communication delays.
IEEE Transactions on Aerospace and Electronic Systems, **26**, 3 (1990), 511–516.

[21] Drozdowski, M. and Głazek, W.
Scheduling divisible loads in a three-dimensional mesh of processors.
Parallel Computing, **25**, 4 (1999), 381–404.

[22] Drozdowski, M. and Wolniewicz, P.
Experiments with scheduling divisible tasks in clusters of workstations.
Lecture Notes in Computer Science, **1900** (2000), 311–319.

[23] Ghose, D. and Mani, V.
Distributed computation with communication delays: Asymptotic performance analysis.
Journal of Parallel and Distributed Computing, **23**, 3 (1994), 293–305.

[24] Hung, J. T. and Robertazzi, T. G.
Scalable scheduling for clusters and grids using cut through switching.
International Journal of Computers and Applications, **26**, 3 (2004), 147–156.

[25] Hung, J. T. and Robertazzi, T. G.
Divisible load cut through switching in sequential tree networks.
IEEE Transactions on Aerospace and Electronic Systems, **40**, 3 (2004), 968–982.

[26] Hung, J. T. and Robertazzi, T. G.
Scheduling nonlinear computational loads.
IEEE Transactions on Aerospace and Electronic Systems, **44**, 3 (2008), 1169–1182.

[27] Kim, H-J.
A novel optimal load distribution algorithm for divisible loads.
Cluster Computing, **6** (2003), 41–46.

[28] Kim, H. J., Jee, G-I., and Lee, J. G.
Optimal load distribution for tree network processors.
IEEE Transactions on Aerospace and Electronic Systems, **32**, 2 (1996), 607–612.

[29] Ko, K.
Scheduling data intensive parallel processing in distributed and networked environments.
Ph.D. dissertation, Dept. of Electrical and Computer Engineering, State University of New York, Stony Brook, NY, 2000.

[30] Ko, K. and Robertazzi, T. G.
Equal allocation scheduling for data intensive applications.
IEEE Transactions on Aerospace and Electronic Systems, **40**, 2 (2004), 695–705.

[31] Ko, K. and Robertazzi, T. G.
Signature search time evaluation in flat file databases.
IEEE Transactions on Aerospace and Electronic Systems, **44**, 2 (2008), 493–502.

[32] Kong, C. S., Bharadwaj, V., and Ghose, D.
Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: Performance and simulation.
Computers and Mathematics in Simulation, **58** (2001), 71–92.

[33] Li, K.
Managing divisible load on partitionable networks.
In J. Schaeffer (Ed.), *High Performance Computing Systems and Applications*, Boston, MA: Kluwer Academic Publishers, 1998, 217–228.

[34] Li, K.
Parallel processing of divisible loads on partitionable static interconnection networks.
Cluster Computing, **6**, 1 (2003), 47–55.

[35] Li, K.
Speedup of parallel processing of divisible loads on k-dimensional meshes and tori.
The Computer Journal, **46**, 6 (2003), 625–631.

[36] Li, K.
Improved methods for divisible load distribution on k-dimensional meshes using pipelined communications.
IEEE Transactions on Parallel and Distributed Systems, **14**, 12 (2003), 1250–1261.

[37] Li, K.
Accelerating divisible load distribution on tree and pyramid networks using pipelined communications.
In *Proceedings of the International Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications*, Santa Fe, NM, Apr. 30, 2004.

[38] Li, P., Veeravalli, B., and Kassim, A. A.
Design and implementation of parallel video encoding strategies using divisible load analysis.
IEEE Transactions on Circuits and Systems for Video Technology, **15**, 9 (2005), 1098–1112.

[39] Li, X., Veeravalli, B., and Ko, C. C.
Distributed image processing on a network of workstations.
International Journal of Computers and Applications, **25**, 2 (2003), 1–10.

[40] Mani, V. and Ghose, D.
Distributed computation in linear networks: Closed-form solutions.
IEEE Transactions on Aerospace and Electronic Systems, **30**, 2 (1994), 471–483.

[41] Moges, M. and Robertazzi, T. G.
Wireless sensor networks: Scheduling for measurement and data reporting.
IEEE Transactions on Aerospace and Electronic Systems, **42**, 1 (2006), 327–340.

[42] Piriyakumar, D. A. L. and Murthy, C. S. R.
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, **28**, 2 (1998), 245–251.

Mt2 □ job no. 2148 □ ieee □ aerospace and electronic systems □ 2148D50 [13] □ (XXX) □ 11-30-10 □ 05:18 PM

[43] Robertazzi, T. G.
Processor equivalence for daisy chain load sharing processors.
IEEE Transactions on Aerospace and Electronic Systems, **29**, 4 (1993), 1216–1221.

[44] Robertazzi, T. G.
Ten reasons to use divisible load theory.
IEEE Computer, **36**, 5 (2003), 63–68.

[45] Sohn, J. and Robertazzi, T. G.
Optimal divisible job load sharing for bus networks.
IEEE Transactions on Aerospace and Electronic Systems, **32**, 1 (1996), 34–40.

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

[47] Suresh, S., Omkar, S. N., and Mani, V.
Parallel implementation of back-propagation algorithm in network of workstations.
IEEE Transactions on Parallel and Distributed Systems, **16**, 1 (2005), 23–34.

[48] Veeravalli, B.
Design and performance analysis of heuristic load balancing strategies for processing divisible loads on Ethernet clusters.
International Journal of Computers and Applications, **27**, 2 (2005), 97–107.

[49] Veeravalli, B., Li, X., and Ko, C. C.
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.

[50] Viswanathan, S., Veeravalli, B., and Robertazzi, T. G.
Resource aware distributed scheduling strategies for large-scale computational cluster/grid systems.
IEEE Transactions on Parallel and Distributed Systems, **18**, 10 (2007), 1450–1461.

[51] Wong, H. M. and Veeravalli, B.
Aligning biological sequences on distributed bus networks: A divisible load scheduling approach.
IEEE Transactions on Information Technology in BioMedicine, **9**, 4 (2005), 489–501.

[52] Yao, J. and Veeravalli, B.
Design and performance analysis of divisible load scheduling strategies on arbitrary graphs.
Cluster Computing, **7**, 2 (2004), 841–865.

[53] Zeng, Z. and Veeravalli, B.
Distributed scheduling strategy for divisible loads on arbitrarily configured distributed networks using load balancing via virtual routing.
Journal of Parallel and Distributed Computing, **66** (2006), 1404–1418.

Keqin Li is a SUNY Distinguished Professor of computer science. His research interests are mainly in the areas of design and analysis of algorithms, parallel and distributed computing, and computer networking. He has contributed extensively to approximation algorithms, parallel algorithms, job scheduling, task dispatching, load balancing, performance evaluation, dynamic tree embedding, scalability analysis, parallel computing using optical interconnects, wireless networks, and optical networks. His current research interests include power-aware computing, location management in wireless communication networks, lifetime maximization in sensor networks, and file sharing in peer-to-peer systems.

Dr. Li has published over 215 journal articles, book chapters, and research papers in refereed international conference proceedings.