Improved Methods for Divisible Load Distribution on *k*-Dimensional Meshes Using Multi-Installment

Yeim-Kuan Chang, Jia-Hwa Wu, Chi-Yeh Chen, and Chih-Ping Chu

Abstract—In the divisible load distribution, the classic methods on linear arrays divide the computation and communication processes into multiple time intervals in a pipelined fashion. Li [21] has proposed a set of improved algorithms for linear arrays that can be generalized to k-dimensional meshes. In this paper, we first propose the algorithm M (multi-installment) that employs the multi-installment technique to improve the best algorithm Q proposed by Li. Second, we propose the algorithm S (start-up cost), which includes the computation and communication start-up costs in the design. Although the asymptotic speed-ups of our algorithms M and S derived from the closed-form solutions are the same as algorithm Q, our algorithms approach the optimal speed-ups considerably faster than algorithm Q as the number of processors increases. Finally, we combine algorithms M and S and propose the algorithm MS. Although algorithm MS has the same the asymptotic performance as algorithms Q and S, it achieves a better speed-up when the load to be processed is very large and the number of processors is fixed or when the load to be processed is fixed and the number of processors is small.

Index Terms—Divisible load theory, linear array, k-dimensional mesh, multi-installment.

1 Introduction

An efficient load scheduling on the resources of a parallel and distributed system or a multiprocessor system is highly desirable for data-intensive applications. Like other mathematical models, such as queuing theory and electric resistive circuit theory, divisible load theory (DLT) [5], [4], [25] provides a powerful tool for modeling data-intensive applications. A divisible load is a load that can be arbitrarily partitioned in a linear fashion and can be distributed to more than one processor to achieve a faster execution time. Each partitioned portion of the load (called a chunk) can be independently processed on any processor on the network.

DLT started with the architecture of a linear array of processors [11] in 1988. Since then, DLT has been widely studied in the literature. DLT gained much attention because of a landmark book written in 1996 [4] and two introductory surveys [5], [25]. The interconnection topologies, such as bus [7], [14], [18], [26], [28], linear array [6], [7], tree [2], [3], [10], [22], hypercube [7], [23], and mesh [7], [8], [13], [17], [21], have been investigated. Applications of divisible computations include linear algebra [9], [16], image processing [24], multimedia applications [1], database searching [7], [12], and Internet packet scheduling [18].

• *The authors are with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Road, Tainan, Taiwan, ROC.*
E-mail: ykchang@mail.ncku.edu.tw, alpha@mail.stut.edu.tw, raccoonz3@gmail.com, chucp@csie.ncku.edu.tw.

Manuscript received 22 May 2006; revised 24 Oct. 2006; accepted 2 Feb. 2007; published online 23 Apr. 2007.
Recommended for acceptance by T. Davis.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0132-0506.
Digital Object Identifier no. 10.1109/TPDS.2007.1103.

We focus on the mesh networks in this paper. In [13] and [8], circuit-switching-based algorithms for 2D and 3D meshes with start-up costs were studied. In [17], another circuit-switching-based algorithm was also proposed for 3D meshes with start-up costs and a multisweep distribution policy. However, the circuit-switching in which communication times are independent of the distances among processors is different from the store-and-forward routing used by us and most researchers in this field. In store-and-forward routing, the communication time of a transmission is linearly proportional to both load size and the distance covered. In [20], Li proposed a divisible load algorithm based on store-and-forward routing for k-dimensional meshes. In [21], Li also proposed improved algorithms by employing pipelined communication. Both [20] and [21] do not consider the start-up costs.

One technique that can minimize the parallel computation time is the multiround policy, or multi-installment policy. In one-round algorithms, each processor receives one load share for computation. However, in multiround algorithms, at least one processor receives two or more load shares, and the load distribution exhibits some kind of repetition or periodicity [2]. Using multiple rounds improves the overlap of computation with communication and, thus, overall performance. If the start-up cost is not considered, an infinite number of rounds would lead to an optimal schedule. Many multiround divisible load algorithms for chains, star, and trees can be found in [2], [3], [12], [15], and [28].

In this paper, we develop a technique also called multi-installment to minimize the parallel processing time. However, the proposed multi-installment technique is different from the existing multiround or multi-installment methods. Our multi-installment algorithms are new designs

improved over the divisible load algorithm proposed in [21], which is a multiround algorithm. Our multi-installment algorithms divide the load distribution process into two different intervals: regular intervals and installment intervals. The regular intervals are executed first and followed by the installment intervals. Also, the message routing method is assumed to be store-and-forward routing. The contributions of this paper are summarized as follows:

- The computation and communication start-up costs are included in the analysis, which were not considered in [21] and [6].
- We derive the closed-form solutions for the parallel execution time and speed-up of a linear array and a k-dimensional mesh.
- We show that the proposed algorithms perform better than those in [21].

The rest of this paper is organized as follows: In Section 2, we present the network model of divisible load algorithms. In Section 3, we review the classic method proposed in [21] and derive its closed-form solutions of the parallel execution time and speed-up by including the start-up costs. In Section 4, we extend the classic algorithms with a multi-installment processing technique and propose a set of improved algorithms. In Section 5, we extend the proposed algorithms for linear arrays to k-dimensional meshes. In Section 6, we compare the performance of the classic method with proposed algorithms. We conclude the paper in Section 7.

TABLE 1
Notations and Terminology

N	The number of processors in a multicomputer system
L	The total load to be processed
T_{cm}	The time to transmit a unit of load along a link.
T_{cp}, $T_{cp,k+1}$	T_{cp} and $T_{cp,k+1}$ are the times to process a unit of load on a processor and a k-dimensional mesh, respectively.
β, β_k	The computation-to-communication ratio of a node in the system. This ratio is $\beta=T_{cp}/T_{cm}$ when a node is a single processor. This ratio is β_k when a node is an k-dimensional mesh.
θ_{cm}	The communication start-up cost in terms of delay time
θ_{cp}, θ_k	θ_{cp} and θ_k are the computation start-up costs in terms of delay time when a node is a single processor and k-dimensional mesh, respectively. (Refer to Table 3)
τ_j	Interval j or the time duration of interval j
x	It is a temporary variable used in each algorithm. The value of x is computed based on L and it is different in all the algorithms studied in this paper.
m	m is the number of multi-installment intervals in the proposed algorithms *M* and *MS*.
k	The number of dimensions of the mesh
ρ, Δ, Δ_k, ρ_k	$\rho=\beta/(N-1)$, $\Delta=(\theta_{cp}-\theta_{cm})/T_{cm}$, $\Delta_k=\frac{\theta_k-\theta_{cm}}{T_{cm}}$ and $\rho_k=\frac{\beta_k}{N_k-1}$.
$h(i)$	$h(0)=x+\Delta$ and $h(i)=\rho\times h(i-1)+\Delta=\rho^i x+\Delta\sum_{j=0}^{i}\rho^j$ for $i=1$ to $m-1$, used in algorithm *MS* running on a linear array.
L_j^A	The amount of the load processed by processor P_j for algorithm A. $\sum_{j=1}^{N} L_j^A = L$.
$T_N^{A,L}$, $T_{N,m}^{A,L}$, $S_N^{A,L}$, $S_{N,m}^{A,L}$	The parallel execution time ($T_N^{A,L}$) and speedup ($S_N^{A,L}=(LT_{cp}+\theta_{cp})/T_N^{A,L}$) of algorithm A for processing L units of load in a system of N processors. We use T_N^A and S_N^A when asymptotic performance is considered or when omitting L or m (number of multi-installments) does not incur confusion. For algorithms with multi-installment, we use $T_{N,m}^{A,L}$ and $S_{N,m}^{A,L}$.
C_i^N	$C_i^N=N(N-1)\times\cdots\times(N-i+1)/i!$ which is the number of combinations of i components selected from a set of N components.
>>	$E >> F$ means E is much larger than F.

2 The Model

We consider a homogeneous mesh network for analysis. A k-dimensional mesh $\mathrm{Mesh_k}$ consists of $N = N_1 \times \ldots \times N_k$ processors, where k is a positive integer that is greater than or equal to one. In $\mathrm{Mesh_k}$, an interior processor P_i for $i = i_1 \times \ldots \times i_k$ is connected to $2k$ neighbors P_j, where

$$j = i_1 \pm 1 \times i_2 \times \ldots \times i_k, i_1 \times i_2 \pm 1 \times \ldots \times i_k, \ldots, i_1 \times i_2 \times \ldots \times i_k \pm 1.$$

In this paper, we assume that the *corner* processor P_N is the initial processor that transmits load fractions to other processors for processing. It is known that we can improve the overall speed-up by using an interior processor as the initial processor [21]. Since using an interior processor is a straightforward extension, the details are omitted in this paper. All the links have the same communication speed and bandwidth. All the processors have the same processing capability. Each interior processor P_i has $2k$ separate ports to communicate with all its neighbors. In other words, processor P_i can send/receive messages to/from all its neighbors simultaneously. A processor sends a load fraction to a neighbor and then it can proceed with other computation and communication activities without waiting for the completion of the load transmission process. However, a processor can only start to perform the computation after the entire load fraction assigned to it is received from its predecessor. To enhance the system performance, our load distribution algorithms allow simultaneous transmission of all processors, rather than sequential load distribution. We list the notations and terminology used in this paper in Table 1.

3 The Existing Method on Linear Arrays

The classic divisible load distribution methods running on a linear array of N processors divide the computation and communication processes into N time *intervals* or *stages* in a pipelined fashion. We assume that the leftmost processor P_N is the initial processor that commences the computation and communication. In the first interval, P_N computes a load fraction and transmits another load fraction to processor P_{N-1} simultaneously. In the second interval, both processors P_N and P_{N-1} compute a load fraction and transmit another load fraction to their successors (that is, P_{N-1} and P_{N-2}) simultaneously. The same computation and communication processes repeat until P_1 receives a load fraction in the $(\mathrm{N}-1)\mathrm{th}$ interval and processes the received load fraction in the last interval. Notice that P_1 only receives a load fraction once. A better load distribution method determines the sizes of the load fractions computed and transmitted by a processor in such a way that the accumulated length of all time intervals is minimized. We assume that the processor does not start its computation and communication processes until it receives the entire load fraction assigned to it. One design principle for a good load distribution method is as follows:

Load balance rule between computation and communication. All processors finish their computation and communication processes at the end of each interval simultaneously. The best load distribution method is that no computation and communication resource is idle in any interval.

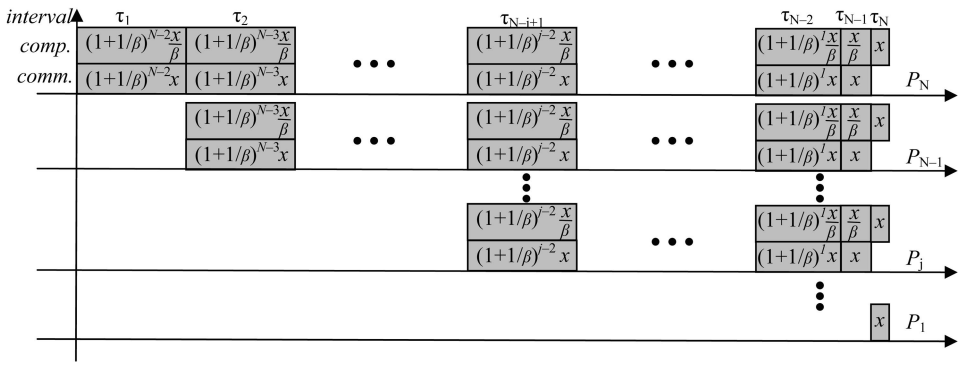


Fig. 1. Load distribution diagram of algorithm Q.

Theoretically, an infinite number of processors could lead to optimal performance for the divisible load distribution when the computation and communication start-up costs (θ_{cp} and θ_{cm}) are ignored. However, we cannot ignore the start-up costs in real-world cases. The performance may degrade as the number of processors exceeds a certain value. In this section, we analyze the algorithm Q proposed in [21] by including the computation and communication start-up costs that are originally neglected. As a result, the analysis results are the same as the ones in [21] when the start-up costs are set to zero.

Algorithm Q on a linear array of N processors divides the parallel execution process into N time intervals. In each interval, some of the processors can compute and communicate a load fraction simultaneously. Allocating load to a processor is based on the load balance rule described above. Thus, in an interval, if the load allocated to a processor for computation is w, then the load allocated to the same processor for transmission is βw, where β is the computation-to-communication ratio of a processor defined in Table 1. As a result, the computation and communication processes are finished at the end of each interval simultaneously. The load distribution diagram of algorithm Q is illustrated in Fig. 1. The initial processor is P_N. In the first time interval τ_1, P_N processes $(1/\beta+1)^{N-2}x/\beta$ units of load and transmits $(1/\beta+1)^{N-2}x$ units of load to P_{N-1} simultaneously. At the end of interval τ_1, P_{N-1} completely receives $(1/\beta+1)^{N-2}x$ units of load for processing. P_{N-1} splits the received load into two parts, $(1/\beta+1)^{N-3}x/\beta$ and $(1/\beta+1)^{N-3}x$, in interval τ_2. The former remains in P_{N-1} for processing and the latter is transmitted to processor P_{N-2}. Notice that the term $(1/\beta+1)^{N-2}x$, which is the amount of load transmitted from the initial processor P_N, comes from an elaborate calculation such that all processors receive x units of load for computation at the last interval. In general, in interval τ_{N-j+1}, all processors P_k for $k=N$ to j process $(1/\beta+1)^{j-2}x/\beta$ units of load and transmits $(1/\beta+1)^{j-2}x$ units of load to P_{k-1} simultaneously.

We have to recalculate the parallel execution time and the speed-up of algorithm Q because the computation and communication start-up costs are to be included in the analysis. Since the load fraction processed by P_j is $L_j^Q=(1/\beta+1)^{j-1}x$ for all $1\le j\le N$ and the total load processed by all processors is L (that is, $\sum_{j=1}^{N}L_j^Q=L$), we obtain $x=\frac{L/\beta}{(1/\beta+1)^N-1}$. In algorithm Q, a processor is not able to finish its computation and communication processes at the end of an interval simultaneously when the computation and communication start-up costs θ_{cp} and θ_{cm} are not the same. This is opposite of the load balance rule because either the computation or communication resource may be idle for some time in each time interval. The duration of an interval is the maximum of the computation time and communication time in that interval. Specifically, we have $\tau_j=(1+1/\beta)^{j-1}(x/\beta)T_{cp}+\max\{\theta_{cp},\theta_{cm}\}$ for $j=1$ to $N-1$ and $\tau_N=(1+1/\beta)^{N-1}(x/\beta)T_{cp}+\theta_{cp}$ since no communication start-up cost is involved in the last interval. Thus, $T_N^{Q,L}=L_N^QT_{cp}+(N-1)\max\{\theta_{cp},\theta_{cm}\}+\theta_{cp}$, as shown in Table 2. We give the following result without a proof because the proof is straightforward:

Theorem 1. *The speed-up of algorithm Q is* $S_N^{Q,L}=\frac{(LT_{cp}+\theta_{cp})}{T_N^{Q,L}}$ *and the asymptotic speed-up is* $S_N^Q=1+\beta$ *when* N *and* L *tend to infinity and* $L>>N$.

4 The Proposed Algorithms

In this section, we first propose an improved algorithm M by using the multi-installment processing technique that will be compared with the original algorithm Q

TABLE 2
Performance Summary

Algorithm	Parallel execution time	Speedup
Q	$T_N^{Q,L}=\frac{LT_{cm}(1+1/\beta)^{N-1}}{(1+1/\beta)^N-1}+(N-1)\max\{\theta_{cp},\theta_{cm}\}+\theta_{cp}$	$S_N^{Q,L}=\frac{LT_{cp}+\theta_{cp}}{T_N^{Q,L}}$
M	$T_{N,m}^{M,L}=\frac{LT_{cm}\left((1+1/\beta)^{N-1}+\sum_{i=1}^{m-1}\rho^i\right)}{(1+1/\beta)^N+\frac{N}{\beta}\sum_{i=1}^{m-1}\rho^i-1}$	$S_{N,m}^{M,L}=\frac{LT_{cp}}{T_{N,m}^{M,L}}$
S	$T_N^{S,L}=\frac{T_{cm}(L-N\Delta)(1+1/\beta)^{N-1}}{(1+1/\beta)^N-1}+\Delta T_{cp}+N\theta_{cp}$	$S_N^{S,L}=\frac{LT_{cp}+\theta_{cp}}{T_N^{S,L}}$
MS	$T_{N,m}^{MS,L}=\frac{T_{cm}\left(L-N\Delta\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i\right)\left((1+1/\beta)^{N-1}+\sum_{k=1}^{m-1}\rho^k\right)}{(1+1/\beta)^N+\frac{N}{\beta}\sum_{k=1}^{m-1}\rho^k-1}+\Delta T_{cp}\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i+(N-1)m\theta_{cp}+\theta_{cp}$	$S_{N,m}^{MS,L}=\frac{LT_{cp}+\theta_{cp}}{T_{N,m}^{MS,L}}$

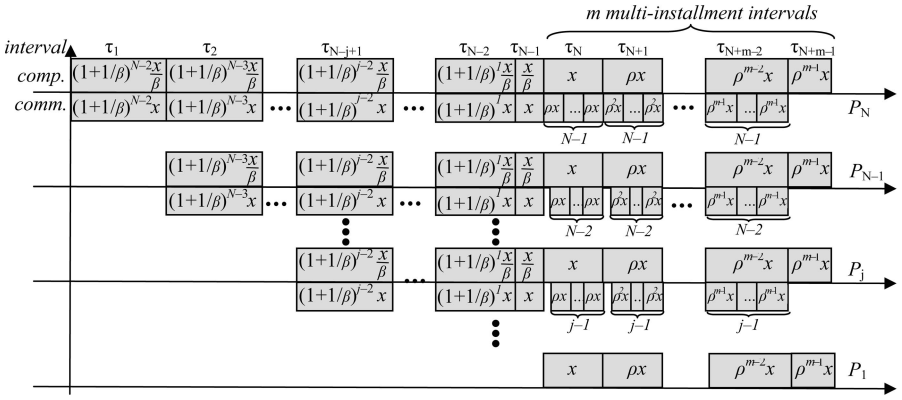


Fig. 2. Load distribution diagram of algorithm M.

proposed in [21]. Both algorithms Q and M do not consider start-up costs. Second, we consider the computation and communication start-up costs and propose an improved algorithm S over algorithm Q. Third, we combine algorithms M and S and develop the best algorithm MS. We first summarize the performance results in Table 2 to give an easy reference for readers to understand the straightforward but tedious equation derivations needed in the proposed algorithms.

4.1 Algorithm M with Multiple Installments

By carefully inspecting the load distribution in Fig. 1, it can be seen that the waiting time for a processor to receive a load fraction to compute is the main delay that limits the performance of algorithm Q. This waiting time is proportional to x. A larger x results in a longer waiting time for a processor to receive a fraction of load to compute. Also, the last processor P_1 in the linear array only receives a fraction of load for computation once. The longer P_1 waits for a fraction of load to compute, the longer is the total parallel execution time. Therefore, we propose an improved algorithm M that uses the multi-installment technique to shrink x. By multiple installments, we mean that all of the processors (including P_1) receive a fraction of load multiple times. The load distribution diagram of algorithm M is shown in Fig. 2. The original interval τ_N is replaced with m intervals called *multi-installment intervals*. The load distribution in the first $N-1$ intervals is exactly the same as that in algorithm Q. Each multi-installment interval except the last one (τ_N to τ_{N+m-2}) is divided into $N-1$ subintervals. In each subinterval of τ_N, every processor computes $x/(N-1)$ units of load at the same time. In the first subinterval of τ_N, P_N transmits $x\beta/(N-1) = \rho x$ units of load to its successor (P_{N-1}). In the second subinterval of τ_N, P_N and P_{N-1} transmit ρx units of load to their successors. Generally, in the kth subinterval of τ_N, processors P_i for $i = N$ to $N-k+1$ transmit ρx units of load to their successors. Other multi-installment intervals $\tau_{N+1} \ldots \tau_{N+m-2}$ are divided into $N-1$ subintervals in the same way as τ_N. In the last interval τ_{N+m-1}, every processor computes $\rho^{m-1}x$ units of load. It can be easily shown that the load balance rule is fulfilled, that is, the computation process and communication process finish simultaneously at the end of each interval τ_{N+i} for $i = 0$ to $m-2$.

We give a complete pseudocode of algorithm M in Fig. 3. Since the load fraction processed by P_j is $L_j^M = ((1/\beta+1)^{j-1} + \sum_{i=1}^{m-1}\rho^i)x$ for all $1 \le j \le N$ and the total load processed by all processors is L (that is, $\sum_{j=1}^{N} L_j^M = L$), we have

$$x = \frac{L/\beta}{(1+1/\beta)^N + \frac{N}{\beta}\sum_{i=1}^{m-1}\rho^i - 1}.$$

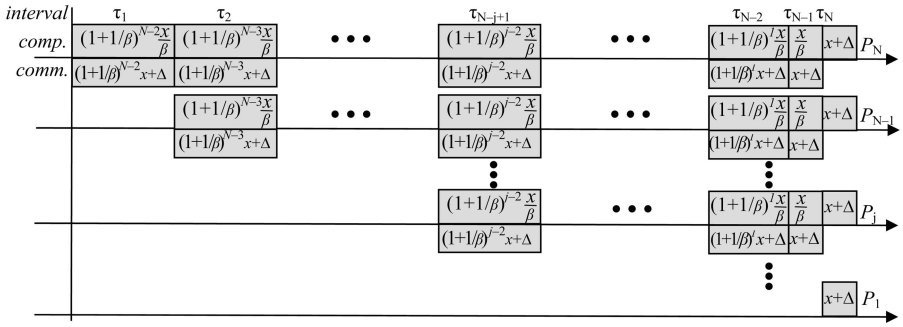
Consequently, the parallel execution time is $T_{N,m}^{M,L} = T_{cp}L_N^M$, and $T_{N,m}^{M,L}$ is linearly proportional to L.

Theorem 2. *The speed-up of algorithm M is $S_{N,m}^M = \frac{T_{cp}L}{T_{N,m}^M}$ and its asymptotic speed-up is 1) $S_{N,m}^M = \beta+1$ if $N=\infty$, 2) $S_{N,m}^M = \beta+1$ if $m=\infty$ and $0<\rho<1$, or 3) $S_{N,m}^M = N$ if $m=\infty$ and $\rho \ge 1$.*

Proof. Refer to Table 2 for $S_{N,m}^M$. If $N=\infty$, $(1+1/\beta)^N$ and $(1+1/\beta)^{N-1}$ are much larger than $\frac{N}{\beta}\sum_{i=1}^{m-1}\rho^i$ and $\sum_{i=1}^{m-1}\rho^i$.

```
Algorithm M
{
01  for (i = 1; i < N + m; i++) { //time interval τi
02      if (i < N)
03          for (j = N – i; j <= N; j++) do in parallel
04              if (j > N – i)
05                  send (1/β+1)^(N–i–1)x amount of load to Pj–1;
06                  receive (1/β+1)^(N–i–1)x amount of load from Pj+1 if j < N;
07                  compute (1/β+1)^(N–i–1)x/β amount of load;
08              else
09                  receive (1/β+1)^(N–i–1)x amount of load from PN–i+1 if j < N;
10              end if
11          end do
12      else if (i < N + m – 1)
13          for (k = 1; k < N; k++) do
14              for (j = 1; j <= N; j++) do in parallel
15                  if (j > N – k)
16                      send ρ^(i–N+1)x amount of load to Pj–1;
17                      receive ρ^(i–N+1)x amount of load from Pj+1 if j < N;
18                      compute ρ^(i–N+1)x/β amount of load;
19                  else if (j = N – k)
20                      receive ρ^(i–N+1)x amount of load from PN–k+1;
21                      compute ρ^(i–N+1)x/β amount of load;
22                  else
23                      compute ρ^(i–N+1)x/β amount of load;
24                  end if
25              end do
26          end do
27      else
28          for (j = 1; j <= N; j++) do in parallel
29              compute ρ^(i–N)x amount of load;
30          end do
31      end if
32  end do
}
```

Fig. 3. Algorithm M.



(a)

```
Algorithm S
{
01  for (i = 1; i < N + m; i++) { //time interval τi
02      if (i < N)
03          for (j = N – i; j <= N; j++) do in parallel
04              if (j > N – i)
05                  send (1/β+1)^(N–i–1)x + Δ amount of load to P(j–1);
06                  receive (1/β+1)^(N–i–1)x + Δ amount of load from P(j+1) if j < N;
07                  compute (1/β+1)^(N–i–1)x/β amount of load;
08              else
09                  receive (1/β+1)^(N–i–1)x + Δ amount of load from P(N–i+1) if j < N;
10              end if
11          end do
12      else
13          for (j = 1; j <= N; j++) do in parallel
14              compute x + Δ amount of load;
15          end do
16      end if
17  end do
}
```

(b)

Fig. 4. Algorithm S. (a) Load distribution diagram of algorithm S. (b) The pseudocode of algorithm S.

Thus, $S^M_{N,m} = \beta + 1$. If $m = \infty$ and $0 < \rho < 1$ $(N \neq \infty)$, we have $\sum_{i=1}^{m-1} \rho^i = \frac{\rho}{1-\rho} = \frac{\beta}{N-1-\beta}$ and, thus,

$$S^M_{N,m} = \frac{(1+1/\beta)^N + \frac{N}{\beta}\frac{\beta}{N-1-\beta} - 1}{\frac{1}{\beta}\left((1/\beta+1)^{N-1} + \frac{\beta}{N-1-\beta}\right)},$$

which can be simplified as $S^M_{N,m} = \beta + 1$. However, if $m = \infty$ and $\rho \geq 1$, $\sum_{i=1}^{m-1} \rho^i$ is much larger than $(1 + 1/\beta)^N$ and $(1 + 1/\beta)^{N-1}$. Thus, $S^M_{N,m} = N$. □

Theorem 3. *For algorithms Q and M without start-up costs, $T^Q_N \geq T^M_{N,m}$ or $S^Q_N \leq S^M_{N,m}$ for $N \geq 1$.*

Proof. We prove that

$$\frac{T^Q_N - T^M_{N,m}}{LT_{cm}} = \frac{(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} - \frac{(1+1/\beta)^{N-1} + \sum_{i=1}^{m-1}\rho^i}{(1+1/\beta)^N - 1 + \frac{N}{\beta}\sum_{i=1}^{m-1}\rho^i} \geq 0,$$

which can be simplified as $f(N, \beta) \geq 0$, where $f(N, \beta) = \frac{N}{\beta}(1/\beta + 1)^{N-1} - (1/\beta + 1)^N + 1$. By expanding $(1 + 1/\beta)^{N-1}$ into $1 + \sum_{i=1}^{N-1} \frac{C^{N-1}_i}{\beta^i}$ and $(1 + 1/\beta)^N$ into $1 + \frac{N}{\beta} + \sum_{i=2}^{N} \frac{C^N_i}{\beta^i}$, we obtain

$$f(N,\beta) = \frac{N}{\beta}\sum\nolimits_{i=1}^{N-1}\frac{C^{N-1}_i}{\beta^i} - \sum\nolimits_{i=2}^{N}\frac{C^N_i}{\beta^i} = \sum\nolimits_{i=2}^{N}(i-1)\frac{C^N_i}{\beta^i}.$$

Since $(i-1)\frac{C^N_i}{\beta^i} \geq 0$ for $i = 2$ to N, we must have $f(N, \beta) \geq 0$. Thus, the theorem follows. □

4.2 Algorithm S with Start-Up Costs

To avoid either computation or communication resource lying idle in a time interval, we propose an improved algorithm S over algorithm Q to distribute the divisible load in a linear array of processors with start-up costs.

Consider applying algorithm Q in the situation where $\theta_{cp} > \theta_{cm}$. A processor must finish its communication process before the computation process. Thus, as the load balance rule states, we have to balance the time delays between the computation and communication processes. Let w_{cp} and w_{cm} be the loads assigned to processor P_N for processing and transmitting in a certain interval, respectively. To make the processes of computing load w_{cp} and transmitting load w_{cm} finish at the same time, we have the equation $w_{cp}T_{cp} + \theta_{cp} = w_{cm}T_{cm} + \theta_{cm}$, which yields $w_{cm} = \beta w_{cp} + (\theta_{cp} - \theta_{cm})/T_{cm}$. Therefore, one way (called *addition method*) to achieve the load balance between computation and communication processes is to add an additional amount of load $\Delta = (\theta_{cp} - \theta_{cm})/T_{cm}$ to the communication process as illustrated in the load distribution diagram in Fig. 4a. Specifically, we add an additional load Δ to the communication process in all the intervals except the final interval. Also, each processor gets an extra computation load Δ in its final interval. Another way, called the *subtraction method*, is to subtract the Δ

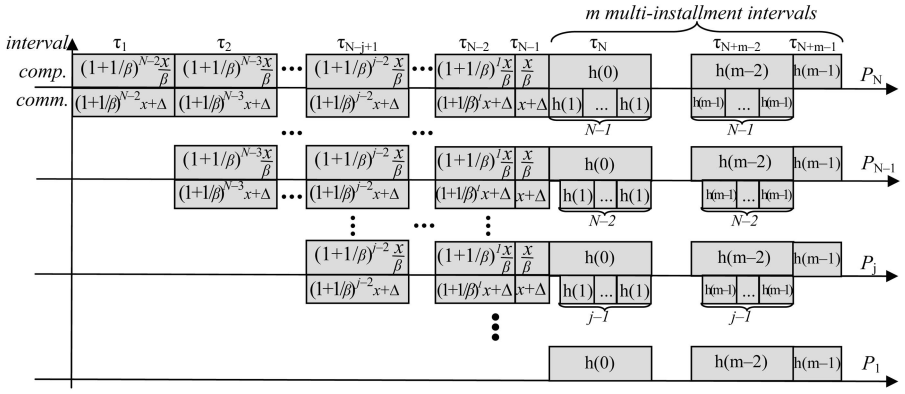


Fig. 5. Load distribution diagram of algorithm MS.

amount of load from the load assigned to the computation process. The parallel execution time of the subtraction method is the same as the addition method. The reason is that the variable x is able to adapt to these two methods. Consequently, both methods work when Δ is negative, that is, $\theta_{cp} < \theta_{cm}$. We use the addition method in this paper. We give a complete pseudocode of algorithm S in Fig. 4b. Since the load processed by processor P_j is $L_j^S = (1+1/\beta)^{j-1}x + \Delta$ for all $1 \le j \le N$ and $\sum_{j=1}^{N} L_j^S = L$, we obtain

$$x = \frac{L - N\Delta}{\beta(1+1/\beta)^N - \beta}$$

and the parallel execution time $T_N^{S,L} = T_{cp}L_N^S + N\theta_{cp}$ (please see Table 2). N is upper bounded by the inequality $L \ge N\Delta$ because, otherwise, variable x becomes negative.

Theorem 4. *The speed-up of algorithm S is* $S_N^{S,L} = \frac{(LT_{cp}+\theta_{cp})}{T_N^{S,L}}$ *and the asymptotic speed-up is* $S_N^S = 1+\beta$ *when* N *and* L *tend to infinity and* $L >> N$.

Proof. The proof is straightforward. □

Theorem 5. $S_N^{S,L} \ge S_N^{Q,L}$ *and* $S_N^{Q,L} \cong S_N^{S,L}$ *if* $L >> N$.

Proof. $S_N^{S,L} \ge S_N^{Q,L}$ means that $T_N^{S,L} \le T_N^{Q,L}$. Therefore, we have to prove that

$$\frac{T_{cm}(L-N\Delta)(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} + \Delta T_{cp} + N\theta_{cp} \le \frac{T_{cm}L(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} + (N-1)\max\{\theta_{cp}, \theta_{cm}\} + \theta_{cp}.$$

Since $N\theta_{cp}$ must be smaller than or equal to $(N-1)\max\{\theta_{cp}, \theta_{cm}\} + \theta_{cp}$, we only need to prove that

$$\Delta\beta - N\Delta\frac{(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} \le 0 \ \text{ or } \ N\frac{(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} \ge \beta.$$

By expanding $(1+1/\beta)^{N-1}$ and $(1+1/\beta)^N$, we have

$$N\frac{(1+1/\beta)^{N-1}}{(1+1/\beta)^N - 1} = \beta\frac{\sum_{i=0}^{N-1}\frac{NC_i^{N-1}}{\beta^i}}{\sum_{i=1}^{N}\frac{C_i^N}{\beta^{i-1}}},$$

which is larger than or equal to β because the $\frac{NC_j^{N-1}}{\beta^j}$ of the numerator must be larger than or equal to $\frac{C_{j+1}^N}{\beta^j}$ of the denominator for $j = 0$ to $N-1$. Therefore, $T_N^{S,L} \le T_N^{Q,L}$. Similarly, if $L >> N$, $T_N^{Q,L} \cong T_N^{S,L}$. Thus, the theorem follows. □

4.3 Algorithm MS with Start-Up Costs and Multiple Installments

In this section, we propose an algorithm MS that is better than algorithm S by utilizing the multi-installment technique as in algorithm M. The load distribution diagram is illustrated in Fig. 5. The complete pseudocode of algorithm MS is also given in Fig. 6. As in algorithm M, the original interval τ_N in algorithm MS is replaced with m multi-installment intervals, and each multi-installment interval is divided into $N-1$ subintervals. In general, in the kth subinterval of τ_{N+j-1} for $j = 1$ to $m-1$, all processors P_i for $i = N$ to $N-k+1$ compute $h(j-1)$ units of load and transmit $h(j)$ units of load to their successors, where $h(j)$ is defined in Table 1.

With $h(i) = \rho \times h(i-1) + \Delta$, it can be easily shown that the load balance rule between the computation and communication processes is satisfied. Since the load processed by processor P_j is

```
Algorithm MS
{
01  for (i = 1; i < N + m; i++) { //time interval τi
02      if (i < N)
03-11       the same as lines 03-11 of Algorithm S in Figure 4(b)
12      else if (i < N + m – 1)
13          for (k = 1; k < N; k++) do
14              for (j = 1; j <= N; j++) do in parallel
15                  if (j > N – k)
16                      send h(i – N) amount of load to Pj–1;
17                      receive h(i – N) amount of load from Pj+1 if j < N;
18                      compute h(i – N)/β amount of load;
19                  else if (j = N – k)
20                      receive h(i – N) amount of load from PN–k+1;
21                      compute h(i – N)/β amount of load;
22                  else
23                      compute h(i – N)/β amount of load;
24                  end if
25              end do
26          end do
27      else
28          for (j = 1; j <= N; j++) do in parallel
29              compute h(m – 1) amount of load;
30          end do
31      end if
32  end do
}
```

Fig. 6. Algorithm MS.

$$L_j^{MS} = x(1/\beta+1)^{j-1} - x + \sum_{i=0}^{m-1} h(i)$$
$$= x(1/\beta+1)^{j-1} + x\sum_{k=1}^{m-1}\rho^k + \Delta\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i$$

for all $1 \le j \le N$ and $\sum_{j=1}^{N} L_j^{MS} = L$, we have

$$x = \frac{\frac{1}{\beta}\left(L - N\Delta\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i\right)}{(1+1/\beta)^N + \frac{N}{\beta}\sum_{k=1}^{m-1}\rho^k - 1}.$$

N is upper bounded by the equation $L \ge N\Delta\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i$ because, otherwise, variable x becomes negative. Also, the parallel execution time to process L units of load is $T_{N,m}^{MS,L} = T_{cp}L_N^{MS} + (N-1)m\theta_{cp} + \theta_{cp}$ (see Table 2). Notice that algorithm S is the same as algorithm MS when the number of installments $m = 1$, and algorithm M is the same as algorithm MS when $\Delta = 0$. We summarize our results in the following theorems:

Theorem 6. *The speed-up of algorithm MS is $S_{N,m}^{MS,L} = \frac{LT_{cp}+\theta_{cp}}{T_{N,m}^{MS,L}}$, and the asymptotic speed-up is $S_N^{MS,L} = 1+\beta$ when N and L tend to infinity and $L >> N$.*

Proof. When N tends to infinity, we obtain $\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i = m$, $\frac{N}{\beta}\sum_{k=1}^{m-1}\rho^k \approx 1$, and $\sum_{k=1}^{m-1}\rho^k \approx 0$ since $\rho = \frac{\beta}{N-1}$. Thus, the theorem follows. □

Theorem 7. *$S_N^{S,L} \ge S_{N,m}^{MS,L}$ when L is constant and N tends to infinity.*

Proof. If N tends to infinity, $\sum_{k=0}^{m-1}\sum_{i=0}^{k}\rho^i \approx m$, $\frac{N}{\beta}\sum_{k=1}^{m-1}\rho^k \approx 1$, and $\sum_{k=1}^{m-1}\rho^k \approx 0$. Thus, we have $T_{N,m}^{MS,L} \cong \frac{LT_{cp}}{1+\beta} + \frac{mN}{1+\beta}(\beta\theta_{cm} + \theta_{cp})$. Since $\beta\theta_{cm} + \theta_{cp} \ge 0$, the larger m is, the longer $T_{N,m}^{MS,L}$ is. As a result, $T_{N,1}^{MS,L} = T_N^{S,L}$ is shorter than $T_{N,m\ge 2}^{MS}$. Thus, the theorem follows. □

Theorem 8. *$S_{N,m}^{MS,L} \ge S_N^{S,L}$ if N is a constant and L tends to infinity.*

Proof. It is sufficient to prove that $T_N^{S,L} - T_{N,m}^{MS,L} \ge 0$. Let $\mathrm{EQ} = T_N^{S,L} - T_{N,m}^{MS,L}$. Based on whether or not the terms in EQ depend on L (refer to Table 2), EQ can be simplified as

$$LT_{cm}S\Big((1+1/\beta)^{N-1}(N-1-\beta)/\beta + 1\Big) + g(N, T_{cp}, T_{cm}, \beta, \theta_{cp}, \theta_{cm}),$$

where $S = \sum_{k=1}^{m-1}\rho^k$ and $g(N, T_{cp}, T_{cm}, \beta, \theta_{cp}, \theta_{cm})$ contains all the terms in EQ that do not depend on L. Since all the parameters N, T_{cp}, T_{cm}, β, θ_{cp}, and θ_{cm} are constants, $g(N, T_{cp}, T_{cm}, \beta, \theta_{cp}, \theta_{cm})$ is a constant. As a result, we only have to prove that $(1+1/\beta)^{N-1}(N-1-\beta)/\beta + 1 > 0$ because $T_{cm}S$ is positive and L can be very large. The same proof has been given in Theorem 3. Thus, the theorem follows. □

4.4 Maximum Number of Installments or Processors

By setting $\frac{d}{dN}T_N^{Q,L} = 0$, we are able to obtain the maximum number of processors $N_{\max}^Q$ such that, if the number of processors exceeds N_{max}, then the speed-up of algorithm Q decreases. The detailed analysis to obtain $N_{\max}^Q$ is given in Appendix A. Since the number of processors is an integer that must be larger than or equal to one, extra computations are needed to determine if $\lfloor N_{\max}^Q \rfloor$ or $\lceil N_{\max}^Q \rceil$ produces the best speed-up.

Subsequently, we only analyze algorithm MS since algorithms M and S are its special cases. By setting $\frac{d}{dm}T_{N,m}^{MS,L} = 0$, we are able to obtain the number of installments m_{max} such that, if the number of installments exceeds m_{max}, then the speed-up decreases. We divide algorithm MS into two cases: $\rho = 1$ and $\rho \ne 1$. We present the result for the case of $\rho = 1$ here, whereas the result for the case of $\rho \ne 1$ is given in Appendix B.

When $\rho = 1$, we have the following parallel execution time:

$$T_{N,m}^{MS,L} = \frac{T_{cm}\left(L - \frac{Nm(m+1)\Delta}{2}\right)\left(\left(\frac{1}{\beta}+1\right)^{N-1} + m - 1\right)}{\left(\frac{1}{\beta}+1\right)^{N} - 1 + \frac{N(m-1)}{\beta}} + \frac{T_{cp}m(m+1)\Delta}{2} + m(N-1)\theta_{cp} + \theta_{cp}.$$

$T_{N,m}^{MS,L}$ can be simplified as $\frac{Am^2+Bm+C}{\frac{N}{\beta}m+\left(\frac{1}{\beta}+1\right)^N - 1 - \frac{N}{\beta}}$, where

$$C = T_{cm}L\left(\left(\frac{1}{\beta}+1\right)^{N-1} - 1\right),$$

$$A = T_{cm}\left(\frac{-N + T_{cp} + 1}{2}\Delta\right)\left(\frac{1}{\beta}+1\right)^{N-1} - \frac{T_{cp}\Delta}{2} + \frac{N(N-1)\theta_{cp}}{\beta},$$

and

$$B = T_{cm}L + \left(\frac{-T_{cm}N + T_{cp} + T_{cm}}{2}\Delta + (N-1)\theta_{cp}\left(\frac{1}{\beta}+1\right)\right)\left(\frac{1}{\beta}+1\right)^{N-1} - \frac{T_{cp}\Delta}{2} - (N-1)\theta_{cp} - \frac{N(N-1)\theta_{cp}}{\beta}.$$

To obtain m_{max}, we perform $\frac{dT_{N,m}^{MS,L}}{dm} = 0$, which leads to

$$\frac{AN}{\beta}m^2 + 2Am\left(\left(\frac{1}{\beta}+1\right)^N - 1 - \frac{N}{\beta}\right) + B\left(\left(\frac{1}{\beta}+1\right)^N - 1 - \frac{N}{\beta}\right) - C\frac{N}{\beta} = 0.$$

As a result, obtaining m_{max} for a minimum $T_{N,m}^{MS,L}$ becomes solving the above quadratic equation in one variable. Since the number of installments is an integer and must be larger than or equal to one, extra computations must be performed to determine if $\lfloor m_{max} \rfloor$ or $\lceil m_{max} \rceil$ produces the best speed-up.

5 Extension to *k*-Dimensional Meshes

To extend the results for linear arrays to k-dimensional meshes, a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is treated as a linear array $Mesh_1$ of N_k nodes, where each node P_j, $1 \le j \le N_k$, is a $(k-1)$-dimensional submesh $Mesh_{k-1}$ of size $N_1 \times N_2 \times \ldots \times N_{k-1}$. Notice that P_j is not a single processor when $k > 1$. We denote the

algorithm A operated on a k-dimensional mesh $Mesh_k$ by A_k. In A_k, the computation time taken by a node to process L units of load is the time (that is, $T^{A,L}_{N_1\times\cdots\times N_{k-1}}$) taken for processing load L in a $(k-1)$-dimensional submesh $Mesh_{k-1}$ using algorithm A, which may not be the same as $L \times T^{A,1}_{N_1\times\cdots\times N_{k-1}}$, where $T^{A,1}_{N_1\times\cdots\times N_{k-1}}$ is the time taken for processing a unit of load in a $(k-1)$-dimensional submesh. In other words, $T^{A,L}_{N_1\times\cdots\times N_{k-1}}$ may not be linearly proportional to L. Depending on if $T^{A,L}_{N_1\times\cdots\times N_{k-1}}$ is linearly proportional to L, we divide the algorithms developed in this paper into two categories based on whether or not the computation and communication start-up costs are included in the analysis.

5.1 Algorithms without Start-Up Costs

Both M_k and Q_k belong to this category. We only consider algorithm M_k because Q_k has been analyzed in [21]. The analysis in Section 4.1 shows that the parallel execution time of a linear array (that is, a one-dimensional mesh) using algorithm M is linearly proportional to L. Thus, the time denoted by $T_{cp,2}$ to compute a unit of load on a linear array of N_1 processors is simply $T^{M,1}_{N_1}$. In general, the time to compute a unit of load on a k-dimensional mesh $Mesh_k$ (denoted by $T_{cp,k+1}$) is simply $T^{M,1}_{N_1\times\cdots\times N_k}$. Therefore, the parallel execution time and the speed-up of algorithm M running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ to process L units of load can be expressed recursively as follows: $T^{M,L}_{N_1\times\cdots\times N_k} = LT_{cp,k+1}$ and $S^{M,L}_{N_1\times\cdots\times N_k} = \frac{LT_{cp}}{T^{M,L}_{N_1\times\cdots\times N_k}} = \frac{T_{cp}}{T_{cp,k+1}}$, where $\beta_k = \frac{T_{cp,k}}{T_{cm}}$, $\rho_k = \frac{\beta_k}{N_k-1}$, and

$$T_{cp,k+1} = \frac{T_{cm}\left(\left(1+\frac{1}{\beta_k}\right)^{N_k-1} + \sum_{i=1}^{m_k-1}(\rho_k)^i\right)}{\left(1+\frac{1}{\beta_k}\right)^{N_k} + \frac{N_k}{\beta_k}\sum_{i=1}^{m_k-1}(\rho_k)^i - 1}$$

for $k \geq 1$, and m_k is the number of installment intervals in the kth dimension. The base conditions are $T_{cp,1} = T_{cp}$ and $\theta_{cp,1} = \theta_{cp}$ and, thus, $\beta_1 = \beta$.

Lemma 1. *The asymptotic β_{k+1} value of algorithm M running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is 1) $\frac{\beta}{k\beta+1}$ if $N_i = \infty$ for all $i = 1$ to k, 2) $\frac{\beta}{k\beta+1}$ if $m_i = \infty$ and $0 < \rho_i < 1$ for all $i = 1$ to k, or 3) $\frac{\beta}{N_1\times\cdots\times N_k}$ if $m_i = \infty$ and $\rho_i \geq 1$ for all $i = 1$ to k.*

Proof. The lemma is a direct extension of Theorem 2. We prove the lemma by induction. We prove case 1 only since cases 2 and 3 can be proved by the same induction analysis. The base condition is $\beta_1 = \beta$. Assume that $\beta_k = \frac{\beta}{(k-1)\beta+1}$, and $N_i = \infty$ for all $i = 1$ to $k-1$. Since $N_k = \infty$, $(1+1/\beta_k)^{N_k}$ and $(1+1/\beta_k)^{N_k-1}$ are much larger than $\frac{N_k}{\beta_k}\sum_{i=1}^{m_k-1}\rho_k^i$ and $\sum_{i=1}^{m_k-1}\rho_k^i$, respectively. Thus, $\beta_{k+1} = \frac{\beta_k}{1+\beta_k} = \frac{\beta}{k\beta+1}$. □

Theorem 9. *The asymptotic speed-up of algorithm M running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is 1) $S^M_{N_1\times\cdots\times N_k} = k\beta + 1$ if $N_i = \infty$ for $i = 1$ to k, 2) $S^M_{N_1\times\cdots\times N_k} = k\beta + 1$ if $m_i = \infty$ and $0 < \rho_i \leq 1$ for $i = 1$ to k, or 3) $S^M_{N_1\times\cdots\times N_k} = N_1 \times \cdots \times N_k$ if $m_i = \infty$ and $\rho_i > 1$ for $i = 1$ to k.*

Proof. The proof can be shown directly from Lemma 1. □

Theorem 10. *For algorithms Q and M without start-up costs, we have $T^{Q,L}_{N_1\times\cdots\times N_k} \geq T^{M,L}_{N_1\times\cdots\times N_k}$ or $S^{Q,L}_{N_1\times\cdots\times N_k} \leq S^{M,L}_{N_1\times\cdots\times N_k}$ if $N_i \geq 1$ for $i = 1$ to k.*

Proof. We can prove the theorem by induction and by using the same analysis as in Theorem 3. □

5.2 Algorithms with Start-Up Costs

Consider algorithms Q, S, or MS running on a k-dimensional mesh $Mesh_k$ that is treated as a linear array $Mesh_1$ of N_k nodes. If the computation and communication start-up costs are considered, the parallel execution time of processing L units of load in each node is not linearly proportional to L. In other words, the computation-to-communication ratio of each node in the linear array $Mesh_1$ is not $T^{A,1}_{N_1\times\cdots\times N_{k-1}}$, where A is Q, S, or MS.

Based on the results shown in Table 2, the parallel execution time of a linear array of N_1 processors is

$$T^{S,L}_{N_1} = \frac{T_{cm}(L - N\Delta)(1+1/\beta)^{N_1-1}}{(1+1/\beta)^{N_1} - 1} + \Delta T_{cp} + N_1\theta_{cp}$$

for algorithm S. By a simple term rearrangement, we have $T^{S,L}_{N_1} = LT_{cp,2} + \theta_{cp,2}$, where

$$\theta_{cp,2} = \Delta T_{cp} + N_1\theta_{cp} - \frac{N_1\Delta T_{cm}(1+1/\beta)^{N_1-1}}{(1+1/\beta)^{N_1} - 1}$$

and

$$T_{cp,2} = \frac{T_{cm}(1+1/\beta)^{N_1-1}}{(1+1/\beta)^{N_1} - 1}.$$

$T^{S,L}_{N_1} = LT_{cp,2} + \theta_{cp,2}$ can be interpreted as "the time to compute a unit of load and the computation start-up cost on a linear array of N_1 processors are $T_{cp,2}$ and $\theta_{cp,2}$, respectively." Consequently, the parallel execution time of algorithm S running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ to process L units of load can be expressed recursively as follows: $T^{S,L}_{N_1\times\cdots\times N_k} = LT_{cp,k+1} + \theta_{cp,k+1}$, where $\beta_k = \frac{T_{cp,k}}{T_{cm}}$,

$$T_{cp,k+1} = \frac{T_{cm}(1+1/\beta_k)^{N_k-1}}{(1+1/\beta_k)^{N_k} - 1}, \quad \Delta_k = \frac{(\theta_{cp,k} - \theta_{cm})}{T_{cm}},$$

and

$$\theta_{cp,k+1} = N_k\theta_{cp,k} + \Delta_k T_{cp,k} - \frac{N_k\Delta_k T_{cm}(1+1/\beta_k)^{N_k-1}}{(1+1/\beta_k)^{N_k} - 1}.$$

The base conditions are $T_{cp,1} = T_{cp}$ and $\theta_{cp,1} = \theta_{cp}$. Therefore, the speed-up of algorithm S running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ to process L units of load is

$$S^{S,L}_{N_1\times\cdots\times N_k} = \frac{LT_{cp} + \theta_{cp}}{T^{S,L}_{N_1\times\cdots\times N_k}}.$$

TABLE 3
The Computation-to-Communication Ratio (β_{k+1}^A) and Computation Start-Up Cost (θ_{k+1}^A) of a k-Dimensional Mesh of Size $N_1 \times N_2 \times \ldots \times N_k$ Processors, where A is Q, M, S, or MS, $\rho_k = \frac{\beta_k}{N_k - 1}$, $\Delta_k^A = \frac{\theta_k^A - \theta_{cm}}{T_{cm}}$, and the Base Conditions Are $\beta_1^A = \beta$ and $\theta_1^A = \theta_{cp}$

A	β_{k+1}^A	θ_{k+1}^A
Q	$\beta_{k+1}^Q = \frac{\left(1+1/\beta_k^Q\right)^{N_k-1}}{\left(1+1/\beta_k^Q\right)^{N_k}-1}$	$\theta_{k+1}^Q = (N_k - 1)\max\{\theta_k^Q, \theta_{cm}\} + \theta_k$
M	$\beta_{k+1}^M = \frac{\left(1+1/\beta_k^M\right)^{N_k-1} + \sum_{i=1}^{m_k-1}(\rho_k)^i}{\left(1+1/\beta_k^M\right)^{N_k} + \frac{N_k}{\beta_k^M}\sum_{i=1}^{m_k-1}(\rho_k^M)^i - 1}$	$\theta_{k+1}^M = 0$
S	$\beta_{k+1}^S = \frac{\left(1+1/\beta_k^S\right)^{N_k-1}}{\left(1+1/\beta_k^S\right)^{N_k}-1}$	$\theta_{k+1}^S = N_k\theta_k^S + \Delta_k^S\beta_k^S T_{cm} - N_k\Delta_k^S\beta_{k+1}^S$
MS	$\beta_{k+1}^{MS} = \frac{\left(1+1/\beta_k^{MS}\right)^{N_k-1} + \sum_{i=1}^{m_k-1}(\rho_k^{MS})^i}{\left(1+1/\beta_k^{MS}\right)^{N_k} + \frac{N_k}{\beta_k^{MS}}\sum_{i=1}^{m_k-1}(\rho_k^{MS})^i - 1}$	$\theta_{k+1}^{MS} = \begin{cases} \Delta_k^{MS}\left(\sum_{l=0}^{m_k-1}\sum_{i=0}^{l}(\rho_k^{MS})^i\right)\left(\beta_k^{MS}T_{cm} - N_k\beta_{k+1}^{MS}\right) + \\ ((N_k-1)m_k+1)\theta_k^{MS} \end{cases}$

Similarly, we can derive $T_{cp,k+1}$ and $\theta_{cp,k+1}$ for algorithms Q and MS. We summarize the results in Table 3. Therefore, the parallel execution time and the speed-up of algorithms Q, S, and MS running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ to process L units of load can be expressed recursively based on the results shown in Table 2 and Table 3. Notice that, for algorithms M and MS, $\rho_k = \frac{\beta_k}{N_k - 1}$.

Lemma 2. *The asymptotic β_{k+1} value of algorithm Q or S running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is $\beta_{k+1} = \frac{\beta}{k\beta+1}$.*

Proof. The proof is the same as that in Lemma 1. □

Theorem 11. *The asymptotic speed-up of algorithm Q or S running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times \ldots \times N_k$ is $S_{N_1\times\cdots\times N_k}^{MS} = k\beta + 1$ if N_i is very large for all $i = 1$ to k and $L >> N_k$.*

Proof. The proof can be shown directly from Lemma 2. □

Lemma 3. *The asymptotic β_{k+1} value of algorithm MS running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is 1) $\frac{\beta}{k\beta+1}$ if $N_i = \infty$ for all $i = 1$ to k, 2) $\frac{\beta}{k\beta+1}$ if $m_i = \infty$ and $0 < \rho_i < 1$ for all $i = 1$ to k, or 3) $\frac{\beta}{N_1\times\cdots\times N_k}$ if $m_i = \infty$ and $\rho_i \geq 1$ for all $i = 1$ to k.*

Proof. The proof is the same as that in Lemma 1. □

Theorem 12. *The asymptotic speed-up of algorithm MS running on a k-dimensional mesh $Mesh_k$ of size $N = N_1 \times N_2 \times \ldots \times N_k$ is 1) $S_{N_1\times\cdots\times N_k}^{MS} = k\beta + 1$ if $N_i = \infty$ for all $i = 1$ to k, 2) $S_{N_1\times\cdots\times N_k}^{MS} = k\beta + 1$ if $m_i = \infty$ and $0 < \rho_i \leq 1$ for all $i = 1$ to k, or 3) $S_{N_1\times\cdots\times N_k}^{MS} = N_1 \times \cdots \times N_k$ if $m_i = \infty$ and $\rho_i > 1$ for all $i = 1$ to k.*

Proof. The proof can be shown directly from Lemma 3. □

Theorem 13. *$S_{N_1\times\cdots\times N_k}^{S,L} \geq S_{N_1\times\cdots\times N_k}^{Q,L}$ and $S_{N_1\times\cdots\times N_k}^{S,L} \cong S_{N_1\times\cdots\times N_k}^{Q,L}$ if $L >> N_i$ for all $i = 1$ to k.*

Proof. The proof is the same as that in Theorem 5. □

Theorem 14. *$S_{N_1\times\cdots\times N_k}^{S,L} \geq S_{N_1\times\cdots\times N_k}^{MS,L}$ when L is a constant and N_i tends to infinity for all $i = 1$ to k.*

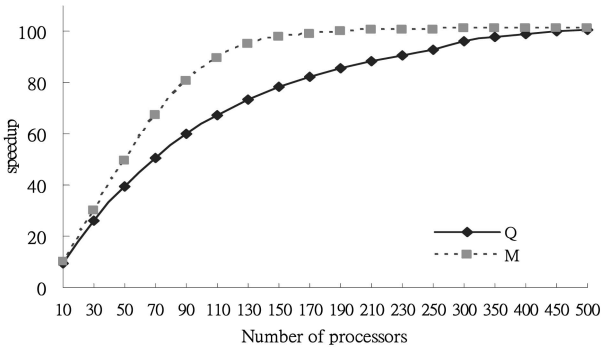


Fig. 7. Speed-up of algorithms Q and M, where $L = 10,000$, $T_{cm} = 1$, $T_{cp} = \beta = 100$, and $m = 5$.

Proof. The proof is the same as that in Theorem 7. □

Theorem 15. *$S_{N_1\times\cdots\times N_k}^{MS,L} \geq S_{N_1\times\cdots\times N_k}^{S,L}$ when N_i is a constant for all $i = 1$ to k and L tends to infinity.*

Proof. The proof is the same as that in Theorem 8. □

6 Discussions of the Results

In Fig. 7, we illustrate the numerical results for the speed-ups of algorithms Q and M without start-up costs (that is, $\theta_{cp} = \theta_{cm} = 0$). We assume that $\beta = 100$ and the number of installments for algorithm M is $m = 5$. Both algorithms Q and M approach their maximal speed-ups of $\beta + 1$ when N is very large. However, the speed-up of algorithm M approaches $\beta + 1$ more quickly than algorithm Q. For example, when the number of processors in the linear array is $N = 200$, the speed-up of algorithm M reaches 100.2, which is very close to the asymptotic speed-up, whereas the speed-up of algorithm Q only reaches 87.2. When $N = 500$, algorithm M reaches the maximal speed-up 101, whereas the speed-up of algorithm Q is 100.3. We also calculate the results for $T_{cp} = T_{cm}$, that is, $\beta = 1$. In this case, both algorithms Q and M reach the maximal speed-up $(1 + \beta = 2)$ very quickly because $(1 + 1/\beta)^N$ and $(1 + 1/\beta)^{N-1}$ dominate the other terms in the speed-up equations.

Next, we show the numerical results for the speed-ups of algorithms Q, S, and MS with two sets of start-up costs. Fig. 8a shows the results for $\theta_{cm} = 1$ and $\theta_{cp} = 2$. Algorithm S performs consistently better than algorithm Q. With $N_{Max}^Q = 395.65$ from Section 4.4 and speed-up calculations for $N = 395$ and 396, we find that the best speed-up of algorithm Q is 91.84 when $N = 396$. The speed-up of algorithm S approaches 91.84 when $N = 285$. The best speed-up of algorithm S is 94.66 when $N = 459$. When N is large, algorithm MS does not perform better than algorithm Q or S, which is in agreement with Theorem 7. However, algorithm MS performs better than algorithm Q or S when N is small. The best speed-up of algorithm MS is 89.82 when $N = 177$. Fig. 8b shows the results for large start-up costs, $\theta_{cm} = 100$ and $\theta_{cp} = 100$. Fig. 8b has a similar performance trend to Fig. 8a in that, when N is small $(N < 100)$, algorithm MS performs the best, and when N is large $(N > 100)$, algorithm MS performs the worst. Also, the speed-ups of algorithms Q and S are almost the same because the start-up costs are large.

Finally, we illustrate the numerical results for the speed-ups of algorithms Q, S, and MS on 2D and 3D meshes. We

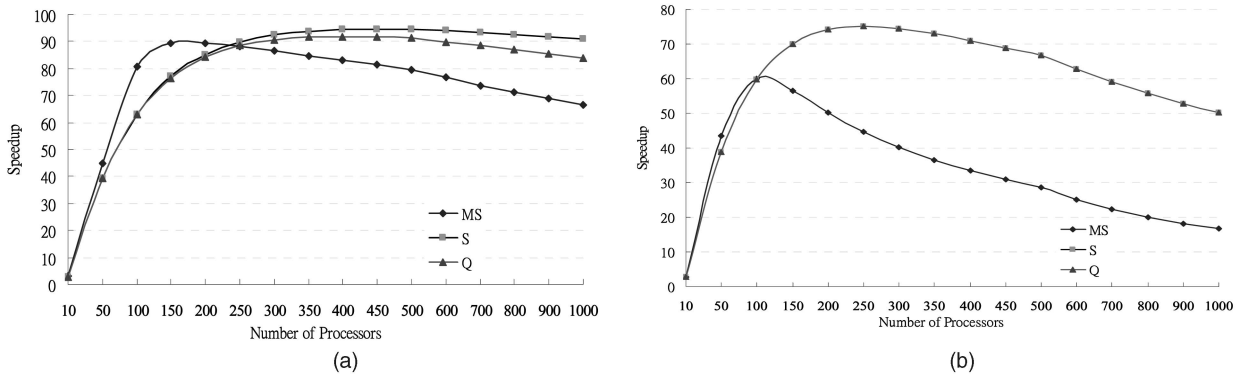


Fig. 8. Speed-up of algorithms Q, S, and MS. $L = 10,000$, $T_{cm} = 1$, $T_{cp} = \beta = 100$, and $m = 5$. (a) $\theta_{cp} = 2$ and $\theta_{cm} = 1$. (b) $\theta_{cp} = 100$ and $\theta_{cm} = 100$.

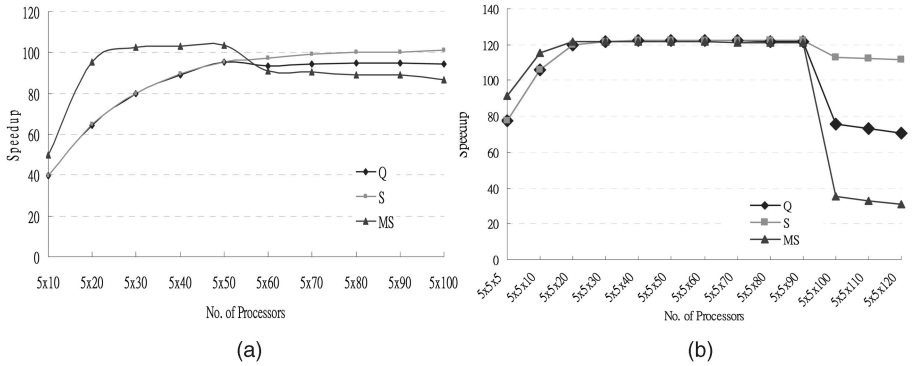


Fig. 9. Speed-ups of algorithms Q, S, and MS, where $L = 200,000$, $T_{cm} = 1$, $T_{cp} = \beta = 100$, $\theta_{cp} = 2$, and $\theta_{cm} = 1$. (a) Two-dimensional meshes. (b) Three-dimensional meshes.

set $L = 200,000$, $T_{cm} = 1$, and $T_{cp} = \beta = 100$. In Fig. 9a, m_1 is set to 2 and m_2 is set to the optimal numbers of installments (that is, m_{max}), which are 8, 25, 11, 7, 5, 2, 2, 2, 2, and 2 for algorithm MS on 2D meshes of 5×10, 5×20, 5×30, 5×40, 5×50, 5×60, 5×70, 5×80, 5×90, and 5×100 processors, respectively. When the mesh is smaller than or equal to 5×50 processors, the speed-ups of algorithms Q and S are almost the same, and algorithm MS performs the best. As in the linear array, when the number of processors in the mesh increases, algorithm MS does not perform better than algorithm Q or S. Algorithm S always perform better than algorithm Q. The speed-ups for 3D meshes of various sizes with $m_1 = m_2 = m_3 = 2$ are shown in Fig. 9b. As in 2D meshes, algorithm MS performs best when the mesh is small and algorithm S performs best when the mesh is large.

7 Conclusion

We have proposed a set of algorithms, M, S, and MS, which employ the multi-installment processing technique and computation and communication start-up costs to distribute divisible load on linear arrays. The extension to multidimensional meshes is also presented. We derived the closed-form solutions for the parallel execution times and speed-ups of the proposed algorithms. When the computation and communication start-up costs are not considered, the proposed algorithm M performs better than algorithm Q [21] in all cases. When the computation and communication start-up costs are considered, the proposed algorithm S performs better than algorithm Q [21] in all cases and the proposed algorithm MS performs better than algorithm S and Q when the number of processors is small or when the total load is very large.

Appendix A

The maximum number of processors $N^Q_{\max}$ such that, if the number of processors exceeds $N^Q_{\max}$, then the speed-up of algorithm Q decreases, is obtained by differentiating $T^{Q,L}_N$ with respect to N as

$$\frac{d}{dN}T^{Q,L}_N = \frac{-LT_{cm}(1+1/\beta)^{N^Q_{\max}-1}\ln(1+1/\beta)}{\left((1+1/\beta)^{N^Q_{\max}}-1\right)^2} + \max\{\theta_{cp},\theta_{cm}\} = 0.$$

This equation can be simplified by setting a new constant $u = (1+1/\beta)$ and a new variable $x = u^{N^Q_{\max}}$ as follows: $x^2 - (\frac{LT_{cm}\ln u}{u\max\{\theta_{cp},\theta_{cm}\}} + 2)x + 1 = 0$. As a result, $N^Q_{\max}$ can be obtained by solving the quadratic equation of x.

Appendix B

For the case of $\rho \neq 1$ in algorithm MS, the number of installments m_{max} such that, if the number of installments exceeds m_{max}, then the speed-up decreases, is obtained by differentiating $T^{MS,L}_{N,m}$ with respect to m as

$$\frac{dT^{MS,L}_{N,m}}{dm} = T_{cp}\frac{dL^{MS}_N}{dm} + (N-1)\theta_{cp} = 0.$$

Through a simple but tedious derivation process, we obtain $\frac{dT^{MS,L}_{N,m}}{dm} = 0 = D\rho^{2m} + Em\rho^m + F\rho^m + G$, where constants D, E, F, and G are complex and are shown below. This nonlinear exponential equation can be solved simply by the Newton numerical method to obtain m_{max}.

From Section 4.3, we have

$$
\begin{aligned}
&L_N^{MS} \\
&= \frac{1/\beta\left(L - \frac{N\Delta}{\rho-1}\left(\frac{\rho^{m+1}-\rho}{\rho-1} - m\right)\right)\left((1+1/\beta)^{N-1} + \frac{\rho}{\rho-1}(\rho^{m-1}-1)\right)}{(1+1/\beta)^N + \frac{N\rho}{\beta(\rho-1)}(\rho^{m-1}-1) - 1} \\
&\quad + \frac{\Delta}{\rho-1}\left(\frac{\rho^{m+1}-\rho}{\rho-1} - m\right) \\
&= \frac{\frac{1}{\beta}\left(L + \frac{N\Delta\rho}{(\rho-1)^2} - \frac{N\Delta}{(\rho-1)^2}\rho^{m+1} + \frac{N\Delta}{\rho-1}m\right)\left(\frac{1}{\rho-1}\rho^m + (1+1/\beta)^{N-1} - \frac{\rho}{\rho-1}\right)}{\frac{N\rho^m}{\beta(\rho-1)} + \left(1+1\frac{1}{\beta}\right)^N - \frac{N\rho}{\beta(\rho-1)} - 1} \\
&\quad + \frac{\Delta}{\rho-1}\left(\frac{\rho^{m+1}}{\rho-1} - m\right) - \frac{\Delta\rho}{(\rho-1)^2} \\
&= \frac{\rho^m\left(\frac{\Delta}{\rho-1}(1+\frac{1}{\beta})^{N-1} + \frac{L}{\beta(\rho-1)} - \frac{\Delta\rho}{(\rho-1)^2}\right) + m\left(\frac{\Delta}{(\rho-1)} - (1+\frac{1}{\beta})^{N-1}\right)}{\frac{N\rho^m}{\beta(\rho-1)} + C_0} \\
&\quad + \frac{C_2C_3/\beta - C_0C_1}{\frac{N\rho^m}{\beta(\rho-1)} + C_0} \\
&= \frac{A\rho^m + Bm + C_2C_3/\beta - C_0C_1}{\frac{N\rho^m}{\beta(\rho-1)} + C_0},
\end{aligned}
$$

where

$$
\begin{aligned}
&A = \frac{\Delta}{\rho-1}\left(1+\frac{1}{\beta}\right)^{N-1} + \frac{L}{\beta(\rho-1)} - \frac{\Delta\rho}{(\rho-1)^2}, \\
&B = \frac{\Delta}{(\rho-1)} - \left(1+\frac{1}{\beta}\right)^{N-1},\ C_0 = \left(1+1\frac{1}{\beta}\right)^N - \frac{N\rho}{\beta(\rho-1)} - 1, \\
&C_1 = \frac{\Delta\rho}{(\rho-1)^2}, C_2 = L + \frac{N\Delta\rho}{(\rho-1)^2}, \text{ and} \\
&C_3 = (1+1/\beta)^{N-1} - \frac{\rho}{\rho-1}.
\end{aligned}
$$

Because $\frac{dT_N^{MS}}{dm} = T_{cp}\frac{dL_N^{MS}}{dm} + (N-1)\theta_{cp} = 0$, we have

$$
\begin{aligned}
0 &= T_{cp}\left(m\rho^m\left(\frac{B(\ln\rho)N}{\beta(\rho-1)}\right) + \rho^m\left(\frac{-(\ln\rho)N(C_2C_3/\beta - C_0C_1)}{\beta(\rho-1)}\right.\right. \\
&\quad \left.\left. + \frac{BN}{\beta(\rho-1)} + A(\ln\rho)C_0\right) + BC_0\right) \\
&\quad + \frac{(N-1)\theta_{cp}N^2\rho^{2m}}{\beta^2(\rho-1)^2} + 2C_0\frac{(N-1)\theta_{cp}N\rho^m}{\beta(\rho-1)} + (N-1)\theta_{cp}C_0^2 \\
&= \rho^{2m}\frac{(N-1)\theta_{cp}N^2}{\beta^2(\rho-1)^2} + m\rho^m\left(\frac{B(\ln\rho)NT_{cp}}{\beta(\rho-1)}\right) + BC_0T_{cp} \\
&\quad + (N-1)\theta_{cp}C_0^2 + \rho^m\left(\frac{-T_{cp}(\ln\rho)N(C_2C_3/\beta - C_0C_1)}{\beta(\rho-1)}\right. \\
&\quad \left. + \frac{T_{cp}BN}{\beta(\rho-1)} + T_{cp}A(\ln\rho)C_0 + 2C_0\frac{(N-1)\theta_{cp}N}{\beta(\rho-1)}\right) \\
&= D\rho^{2m} + Em\rho^m + F\rho^m + G,
\end{aligned}
$$

where

$$
\begin{aligned}
&D = \frac{(N-1)\theta_{cp}N^2}{\beta^2(\rho-1)^2}, E = \frac{B(\ln\rho)NT_{cp}}{\beta(\rho-1)}, \\
&G = BC_0T_{cp} + (N-1)\theta_{cp}C_0^2, \text{ and} \\
&F = \frac{-T_{cp}(\ln\rho)N(C_2C_3/\beta - C_0C_1)}{\beta(\rho-1)} + \frac{T_{cp}BN}{\beta(\rho-1)} \\
&\quad + T_{cp}A(\ln\rho)C_0 + 2C_0\frac{(N-1)\theta_{cp}N}{\beta(\rho-1)}.
\end{aligned}
$$

REFERENCES

[1] D. Altilar and Y. Paker, "Optimal Scheduling Algorithms for Communication Constrained Parallel Processing," *Proc. Eighth Int'l Euro-Par Conf. (Euro-Par '02)*, pp. 197-206, 2002.

[2] V. Bharadwaj and H.M. Wong, "Scheduling Divisible Loads on Heterogeneous Linear Daisy Chain Networks with Arbitrary Processor Release Times," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, no. 3, pp. 273-288, Mar. 2004.

[3] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Trans. Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 555-567, 1995.

[4] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE CS Press, 1996.

[5] V. Bharadwaj, D. Ghose, and T. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing*, vol. 6, no. 1, pp. 7-17, 2003.

[6] V. Bharadwaj and H.M. Wong, "Scheduling Divisible Loads on Heterogeneous Linear Daisy Chain Networks with Arbitrary Processor Release Times," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, no. 3, 273-288, Mar. 2004.

[7] J. Blazewicz, M. Drozdowski, and M. Markiewicz, "Divisible Task Scheduling—Concept and Verification," *Parallel Computing*, vol. 25, pp. 87-98, 1999.

[8] J. Blazewicz, M. Drozdowski, F. Guinand, and D. Trystram, "Scheduling a Divisible Task in a 2-Dimensional Mesh," *Discrete Applied Math.*, vol. 94, nos. 1-3, pp. 35-50, 1999.

[9] S. Chan, V. Bharadwaj, and D. Ghose, "Large Matrix-Vector Products on Distributed Bus Networks with Communication Delays Using the Divisible Load Paradigm: Performance and Simulation," *Math. and Computers in Simulation*, vol. 58, pp. 71-92, 2001.

[10] S. Charcranoon, T.G. Robertazzi, and S. Luryi, "Parallel Processor Configuration Design with Processing/Transmission Costs," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 987-991, Sept. 2000.

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

[12] M. Drozdowski, *Selected Problems of Scheduling Tasks in Multiprocessor Computing Systems.* Poznan Univ. of Technology Press, 1997.

[13] M. Drozdowski and W. Glazek, "Scheduling Divisible Loads in a Three-Dimensional Mesh of Processors," *Parallel Computing*, vol. 25, no. 4, pp. 381-404, 1999.

[14] M. Drozdowski and P. Wolniewicz, "Out-of-Core Divisible Load Processing," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 10, pp. 1048-1056, Oct. 2003.

[15] M. Drozdowski and P. Wolniewicz, "Performance Limits of Divisible Load Processing in Systems with Limited Communication Buffers," *J. Parallel and Distributed Computing*, vol. 64, no. 8, pp. 960-973, 2004.

[16] D. Ghose and H.J. Kim, "Computing BLAS Level-2 Operations on Workstation Clusters Using the Divisible Load Paradigm," *Math. and Computer Modelling*, vol. 41, no. 1, pp. 49-70, Jan. 2005.

[17] W. Glazek, "Distributed Computation in a Three-Dimensional Mesh with Communication Delays," *Proc. Sixth Euromicro Workshop Parallel and Distributed Processing*, pp. 38-42, Jan. 1998.

[18] J. Guo, J. Yao, and L.N. Bhuyan, "An Efficient Packet Scheduling Algorithm in Network Processors," *Proc. INFOCOM*, Mar. 2005.

[19] C. Lee and M. Hamdi, "Parallel Image Processing Applications on a Network of Workstations," *Parallel Computing,* vol. 21, pp. 137-160, 1995.
[20] K. Li, "Managing Divisible Load on Partitionable Networks," *High Performance Computing Systems and Applications,* J. Schaeffer, ed., pp. 217-228, Kluwer Academic Publishers, 1998.
[21] K. Li, "Improved Methods for Divisible Load Distribution on k-Dimensional Meshes Using Pipelined Communications," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 12, pp. 1250-1261, Dec. 2003.
[22] K. Li, "Accelerating Divisible Load Distribution on Tree and Pyramid Networks Using Pipelined Communications," *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS '04),* p. 228, Apr. 2004.
[23] X. Li, B. Veeravalli, and C.C. Ko, "Divisible Load Scheduling on a Hypercube Cluster with Finite-Size Buffers and Granularity Constraints," *Proc. First IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '01),* pp. 660-667, May 2001.
[24] X. Li, V. Bharadwaj, and C.C. Ko, "Distributed Image Processing on a Network of Workstations," *Int'l J. Computers and Applications,* vol. 25, no. 2, pp. 1-10, 2003.
[25] T.G. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer,* pp. 63-68, May 2003.
[26] B. Veeravalli, X. Li, and C.C. Ko, "On the Influence of Start-Up Costs in Scheduling Divisible Loads on Bus Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 12, pp. 1288-1305, Dec. 2000.
[27] R. Wang, A. Krishnamurthy, R. Martin, T. Anderson, and D. Culler, "Modeling Communication Pipeline Latency," *Proc. Joint Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '98/PERFORMANCE '98),* pp. 22-32, 1998.
[28] Y. Yang, K.V.D. Raadt, and H. Casanova, "Multiround Algorithms for Scheduling Divisible Loads," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 11, pp. 1092-1102, Nov. 2005.

Yeim-Kuan Chang received the MS degree in computer science from the University of Houston, Clear Lake, in 1990 and the PhD degree in computer science from Texas A&M University in 1995. He is currently an assistant professor in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include computer architecture, multiprocessor systems, Internet router design, and computer networking.

Jia-Hwa Wu received the BS degree in mechanical engineering from Feng Chia University, Taiwan, in 1981 and the MBA degree in industrial management from the National Cheng Kung University, Taiwan, in 1986. He is currently a doctoral candidate in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include parallel compilers, data mining, and Internet computing.

Chi-Yeh Chen received the BS degree in communication engineering from Da-Yeh University, Changhua, Taiwan, ROC, in 2001 and the MS degree in computer science and information and engineering from the National Cheng Kung University, Tainan, Taiwan, ROC, in 2005. He is currently an engineer in the Plasma and Space Science Center, National Cheng Kung University, Tainan, Taiwan, ROC. His research interests include parallel algorithm and load distribution.

Chih-Ping Chu received the BS degree in agricultural chemistry from the National Chung Hsing University, Taiwan, the MS degree in computer science from the University of California, Riverside, and the PhD degree in computer science from Louisiana State University. He is currently a professor in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include parallelizing compilers, parallel computing, parallel processing, Internet computing, DNA computing, and software engineering.