ACADEMIC PRESS

Available at www.ComputerScienceWeb.com POWERED BY SCIENCE @ DIRECT•

Journal of Computer and System Sciences 67 (2003) 1–25

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

http://www.elsevier.com/locate/jcss

A simple reduction of non-uniformity in dynamic load balancing of quantized loads on hypercube multiprocessors and hiding balancing overheads[☆]

Hwakyung Rim,[a,*] Ju-wook Jang,[b] and Sungchun Kim[c]

[a] *Department of Computer Education, Busan National University of Education, Busan, South Korea*
[b] *Department of Electronic Engineering, Sogang University, Seoul, South Korea*
[c] *Department of Computer Engineering, Sogang University, Seoul, South Korea*

Received 27 October 2000; revised 23 May 2002

Abstract

We reconsider the dimension exchange method (DEM), a known dynamic load balancing scheme on hypercube multiprocessors, for its inefficiency when the load is not divisible in arbitrary sizes but divisible only in a fixed size. We show that a direct application of the DEM to this kind of load may result in nonuniformity (the maximum difference in assigned units of load among the processors after balancing) as large as $\log N$ for a hypercube of size N. Since the processing time after the balancing depends on the processor with the maximum number of units, it is desirable to reduce the nonuniformity as small as possible. In this paper, we propose a new simple method, odd even method (OEM), which reduces the nonuniformity to no more than $\lceil \frac{1}{2}\log N \rceil$. The claim is proved and confirmed by enumerating all possible combinations of load for hypercubes of limited sizes using a computer. To estimate the accumulated effect of our balancing method under real-world parallel processing environment, a simulation for hypercube multiprocessors using SLAM II tool is performed. The result shows about 30% improvement in speedup in overall processing. In addition, we introduce new techniques for hiding communication overheads involved in balancing. The basic idea is coalescing some phases of balancing to overlap or pipeline the transmission of load whenever possible. They proved effective in making links busy transmitting load as soon as possible, thus reducing the transmission time. It is shown via simulation that pipelining is powerful even in the

[☆] A preliminary version of this paper appears in IPPS'99.

[*] Corresponding author. Department of Computer Education, Busan National University of Education, 263, Gojedong 1, Yeonjegu, Busan, 611-736, Korea.

E-mail addresses: ackyung@bnue.ac.kr (H. Rim), jjang@sogang.ac.kr (J-w Jang), ksc@arqlab1.sogang.ac.kr (S. Kim).

doi:10.1016/S0022-0000(03)00033-3

presence of severe unevenness of initial load distribution. Combined together, proposed techniques reduce (hide) communication overheads by 15–50% depending on initial load distribution.

Keywords: Dynamic load balancing; Hypercube multiprocessors; Communication overheads; Quantized loads; Parallel processing environment; Uniform distribution

1. Introduction

The balancing of loads has been an important matter in parallel and distributed processing which greatly affects the speedup in processing time. The underlying network topology or architecture, communication overheads involving transfer of data between processors, the size of load, the size of smallest unit of load, etc. should be carefully considered to obtain an efficient solution to this problem. From the viewpoint of balancing load among processors, most of the real-world processing loads can be classified into one of the following three categories.

- divisible in arbitrary sizes,
- divisible in multiples of a fixed size (quantum),
- indivisible.

Great part of real-world load falls into the second category, which we call quantized loads. Examples include matrix multiplication and transform. However, many previous balancing methods fail to address the effect of quantized loads on their balancing performance. In this paper, we consider dynamic load balancing based on the dimension exchange method (DEM) on hypercube multiprocessor (or multicomputer) with emphasis on quantized loads [3]. In dynamic load balancing, load is balanced in the middle of execution to suit the balancing need for applications in which load can be unexpectedly generated and destroyed in the course of execution. Applications such as partial differential equation solvers using adaptively generated grids or data fusion and tracking problems are good examples [2,4].

In the DEM method, load balancing happens in one dimension at a time, where a dimension corresponds to some subset of all pairs of directly connected processors. The result is an equal distribution of load between every pair of processors in the subset. Every dimension takes its turn, until some conditions are satisfied [7]. Specifically in a hypercube of N processors, balancing is performed in $\log N$ phases. In phase i, a subset consists of directly connected pairs of processors such that the binary representation of the processor id's in a pair differs only in ith bit position.

The DEM method for balancing quantized loads on hypercube of size N may lead to difference as large as $\log N$ in assigned loads after balancing. The balancing is performed between two processors in such a way that the processor with larger load sends part of its load to the other processor so that they have as equal load as possible. As the size of hypercube grows, the maximum difference of $\log N$ will also grow, which in turn increases the processing time.

In this paper we propose a new method which reduces the maximum difference by half to $\lceil \frac{1}{2} \log N \rceil$. The basic idea of our method is to reduce the instances in which total number of quantized loads from a pair of processors in a balancing phase is odd. The method is proved and confirmed by enumerating all possible cases for hypercubes of limited sizes using a computer. To estimate the accumulated effect of many balancing instances under real-world parallel processing environment, a simulation on a hypercube multicomputer using SLAM II tool is performed. The result shows about 30% improvement in speedup which results from reduced processing time, which in turn results from reduced nonuniformity.

In addition, we introduce new techniques for hiding communication overheads involved in balancing. We develop a technique to overlap transmissions in different phases. It turned out to be effective, resulting in up to 25% reduction (hiding) depending on initial load distribution. However, the overlapping fails when load is severely imbalanced. As an extreme example, consider the case where all the initial load resides in a processor P out of 2^N processors. Transmission on other links than the one connecting P cannot be initiated until enough load arrives from P. To remedy this, we employ pipelining which forwards a unit as soon as it arrives. This accelerates diffusion of load from heavily loaded processors to processors with scanty load. Pipelining combined with overlapping is shown to reduce (hide) communication overheads by as much as 50%, again depending on initial load distribution.

The rest of the paper is organized as follows. The DEM method is analyzed for its performance on quantized loads in Section 2. Our method called odd even method (OEM) is presented in Section 3 and simulation results are presented in Section 4. Section 5 compares OEM method against CWA which collects global load information for uniform balancing in terms of communication overheads. Section 7 concludes this paper.

2. The dimension exchange method (DEM) for load balancing

In this section, we provide a brief explanation of the DEM method and show that a direct application of the DEM method on quantized loads in a hypercube of size N may result in nonuniform distribution with difference as large as $\log N$ [1]. Compared with uniform distribution (difference $= 0$ or 1), this nonuniform distribution will increase the processing time by $\log N$ units. In the next section, we propose a new method which reduces the maximum difference on balancing to $\lceil \frac{1}{2} \log N \rceil$ for quantized loads. We start with the definition of the balancing problems on a hypercube.

Assume we have an d-dimensional hypercube of N processors, where $N = 2^d$. Let P_k (k is called processor id) and $W_{k,i}$ denote the kth processor and the load assigned to the processor P_k before balancing phase i, where $k = 0, 1, 2, \dots, N-1$ and $i = 0, 1, 2, \dots, \log N - 1$. $W_{k,i}$ is a nonnegative integer representing the number of quantum loads. $W_{k,\log N}$ represents the load on P_k after completion of balancing. Fig. 1 shows a hypercube of 8 processors with binary representation of the processor id.

In the DEM method, balancing is performed in $\log N$ phases. In phase i, balancing is performed along dimension i, where $i = 0, 1, 2, \dots, \log N - 1$. For $j, k = 0, 1, 2, \dots, N-1$, balancing is performed between P_k and P_j where the binary representation of k and j differs only in the ith bit position. For a hypercube of 32 processors, in phase 0, balancing is performed between P_0 and P_1, P_2 and P_3, P_4 and $P_5, \dots, P_{30}$ and P_{31}. In phase 1, balancing is performed between P_0 and P_2, P_1 and P_3, P_4 and $P_6, \dots, P_{29}$ and P_{31}. Continuing in this way, balancing is performed between P_0 and P_{16}, P_1 and P_{17}, P_2 and $P_{18}, \dots, P_{15}$ and P_{31} in phase 4.

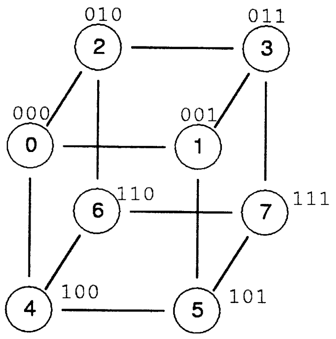


Fig. 1. A hypercube of 8 processors with the binary representation of the processor id.

In the DEM method, balancing is performed between two processors in such a way that the processor with larger load sends part of its load to the other processor so that they have as equal load as possible. If the load is infinitely divisible, each processor can always take exactly the same amount of load. For example, if P_0 and P_2 have 7 and 14 units of load before phase 1, P_0 and P_2 will both have 10.5 units of load after balancing in phase 1. The load distribution will be uniform after $\log N$ phases if the job is infinitely divisible. But this is not a practical assumption regarding reality in job distribution. It is more practical to assume that the job is divisible in a fixed quantum size and thus $W_{k,i}$ should be a nonnegative integer for all k, i. Since P_2 has larger load (14) before balancing than that of P_0 (7), P_2 sends 3 units to P_0, which results in $W_{0,2} = 10$ and $W_{2,2} = 11$. Balancing on a hypercube using the original DEM can be described as follows:

The original DEM

```
For $i = 0$ to $\log N - 1$ do
  For all pairs of processors $P_k$ and $P_j$, where $j = k \oplus 2^i$ do *in parallel*
  if $W_{k,i} \geqslant W_{j,i}$ then
                $W_{k,i+1} = \lceil (W_{k,i} + W_{j,i})/2 \rceil$
                $W_{j,i+1} = \lfloor (W_{k,i} + W_{j,i})/2 \rfloor$
        else
                $W_{k,i+1} = \lfloor (W_{k,i} + W_{j,i})/2 \rfloor$
                $W_{j,i+1} = \lceil (W_{k,i} + W_{j,i})/2 \rceil$
  end if
end do *in parallel*
```

Fig. 2 illustrates an example of balancing on a hypercube of 8 processors using the original DEM method. The direction of the arrows represents the dimension along which balancing is performed.

2.1. The worst case in the original DEM method for quantized loads

One major problem of the original DEM method, when directly applied to quantized loads, is as follows. A balancing phase between two neighboring processors may cause one unit of

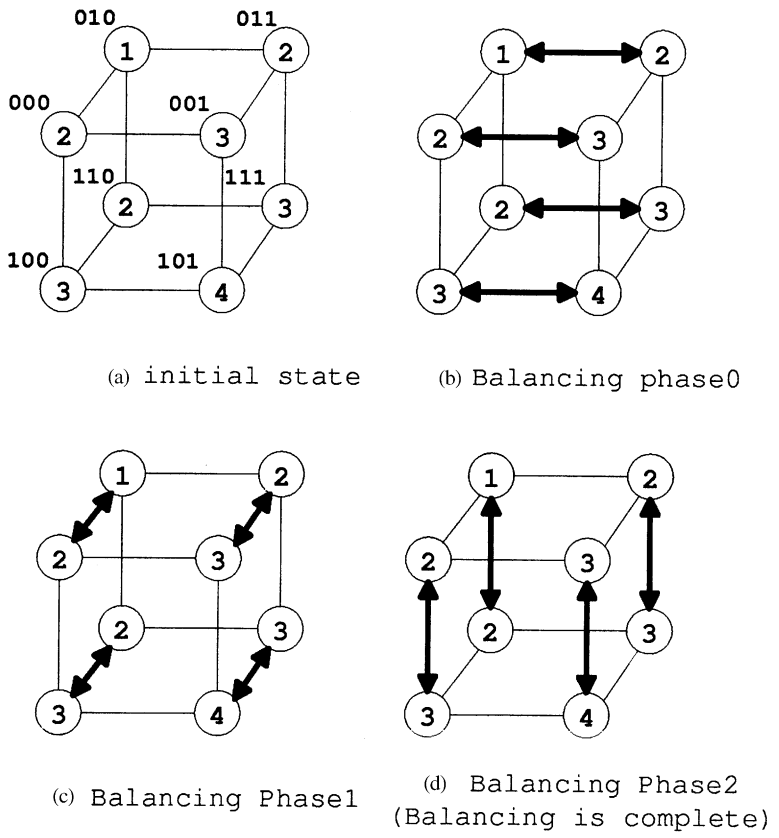


Fig. 2. Balancing of quantized loads on a hypercube of 8 processors using the original DEM method.

difference if the sum of the loads from the two processors is not an even number. If P_k has 14 units and P_j has 21, then after the balancing, P_k will have 17 and P_j will have 18. This difference can be accumulated to result in the difference of $\log N$ after the $\log N$ phases.

Fig. 3 illustrates one of the worst cases for balancing quantized loads on a hypercube of 16 processors. The arrows represent the dimensions along which balancing is performed. One can note that the DEM method will not change the number of quantized loads on any processor all through the $\log N$ (4 in this example) phases in this particular example. The processor with the smallest load has a units, while the processor with the largest load has $a + 4$ units. The a can be any nonnegative integer. If we put $a + 1$ in place of a, we have another hypercube of 16 processors in which there exists also a maximum difference of 4. If we combine the two hypercubes to form a hypercube of 32 processors, the resulting hypercube will have the difference as large as 5. Induction using the size of hypercubes from 16, 32 to 2^d leads us to the proof that balancing using the DEM method on a hypercube of N processors may result in the difference as large as $\log N$.

The difference after the balancing affects the processing time before another balancing is initiated. If the load is distributed as in Fig. 2 after balancing and there is no more balancing until the load is complete, then the processing time will depend on the processor holding the load of

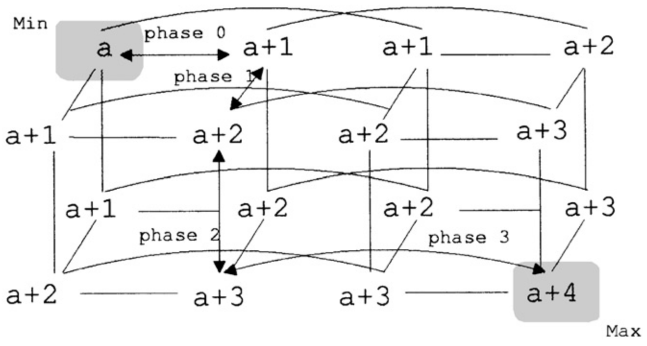


Fig. 3. Balancing of quantized loads on a hypercube of 8 processors using the original DEM method: a worst case.

$a+4$ units. If one quantum of loads takes one unit of time, then the processor with the largest load will finish its assigned load in $a+4$ time units, while the processor with the smallest load will finish in a time units. Since the whole load is complete only after all the processors finish their assigned load, the processor with the largest load will determine the processing time for the hypercube. If the size of hypercube is N, the processor with the largest load will finish in $a+\log N$ time. Thus, the impact of the maximum difference on the processing time depends on the ratio between a and $\log N$. If a is not large compared with $\log N$, the effect of the maximum difference cannot be ignored. As the size of hypercube grows, the maximum difference of $\log N$ will also grow. If balancing is initiated many times during the processing of the whole load, the effect of the maximum difference will be accumulated to increase the processing time. Later we will show from simulation how much the maximum increases the processing time, when accumulated. In the following section, we propose a new method which reduces the maximum difference by half.

3. OEM for balancing quantized loads

In this section, we present a new method called odd even method (OEM) which improves the original DEM method especially for balancing quantized loads in hypercubes. The basic idea of OEM is to reduce the instances in which total number of quantized loads from two neighboring processors participating in a balancing phase is odd. If the total number is odd, then one processor should have one more quantum of load, which results in difference of one unit after the balancing phase. If the total number is odd again in the next balancing phase, it is possible that the difference is incremented to two. Phases 0 and 1 in Fig. 2 illustrates this case. In OEM, we devised a scheme which prevents the difference from being incremented on every balancing phase. This is made possible by changing the way the load is divided between the two processors when the sum of quantized loads from two neighboring processors is odd. The following pseudo-code describes OEM:

Odd even method

```
For $i = 0$ to $\log N - 1$ do
For all pairs of processors $P_k$ and $P_j$, where $j = k \oplus 2^i$ do in parallel
        if $W_{k,i} + W_{j,i} = 2m + 1$ and $m$ is
                an odd integer then
                $W_{k,i+1} = m$
                $W_{j,i+1} = m + 1$
        else if $W_{k,i} + W_{j,i} = 2m + 1$ and $m$ is
                an even integer then
                $W_{k,i+1} = m + 1$
                $W_{j,i+1} = m$
        else
                        $W_{k,i+1} = m$
                        $W_{j,i+1} = m$
        end if
        end if
end do in parallel
```

For example, if P_6 (binary representation: 110) and P_7 (binary representation: 111) have 4 and 7 quanta of loads before balancing phase 0 ($W_{6,0} = 4$, $W_{7,0} = 7$), then P_6, P_7 will have 5, 6 units of load after balancing phase 0 ($W_{6,1} = 5$, $W_{7,1} = 6$). In this way, we increase the probability that loads of same type (odd or even) meet for balancing in the next phase. To see why, consider the two subcubes, 0-cube and 1-cube, where all the processors in the 0-cube (l-cube) have the ith bit of the binary representation of the processor id equal to 0(1). The set of processors belonging to 0-cube (1-cube) will change as i goes from 0 to $\log N - 1$. Note that two processors which belong to the same subcube (0-cube or 1-cube) in phase i will meet for balancing in phase $i + 1$. Therefore if we route loads of same type (odd or even) to same subcube during balancing in phase i, it increases the probability that loads of same type meet for balancing in phase $i + 1$. Furthermore, in OEM it is guaranteed that the maximum difference in a balanced subcube is not incremented in immediate succession. Here a balanced subcube after phase i consists of 2^{i+1} processors whose binary representation of their id's differ only in the $i + 1$ consecutive least significant bits. For example, P_0, P_1, P_2, P_3 form a balanced subcube after phase 1. When the size of a balanced subcube reaches the size of the hypercube, the balancing is complete. The maximum difference in a balanced subcube equals the maximum load minus the minimum load in the subcube. In the DEM method, the maximum difference may be incremented on each phase leading to a worst case of $\log N$.

However, in OEM, the difference may not be incremented in immediate succession. If the maximum difference in a balanced subcube is incremented in phase i, the maximum difference in a balanced subcube in phase $i + 1$ containing the half-size subcube is not incremented. To verify this, consider Figs. 4 and 5 where two representative cases incrementing difference for OEM are illustrated.

We prove the correctness of the OEM algorithm as in Theorem 1.

Theorem 1. *The difference in the units of load after balancing using the OEM algorithm is no more than* $\lceil \frac{\log N}{2} \rceil$, *where N is the number of nodes in the hypercube.*

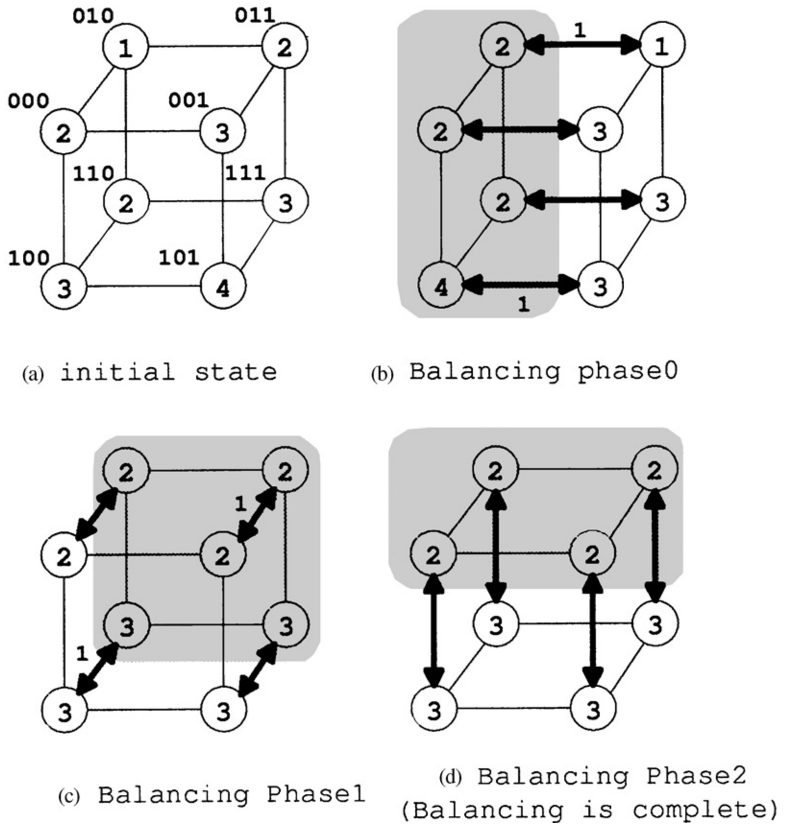


Fig. 4. Balancing of quantized loads on a hypercube of 8 processors using OEM in worst case of DEM method (Fig. 2).

Proof. First we prove for $N = 4$, and apply induction to complete the proof.

1. ($N = 4$) Initially, four nodes P_0, P_1, P_2, and P_3 have $W_{0,0}$, $W_{1,0}$, $W_{2,0}$, and $W_{3,0}$ units of load, respectively. We identify six different cases.
   - *Case* I: $W_{0,0} + W_{1,0} = 2m + 1$, for some odd m and $W_{2,0} + W_{3,0} = 2l + 1$, for some odd l. After balancing phase 0, $W_{0,1} = m$, $W_{1,1} = m + 1$ and $W_{2,1} = l$, $W_{3,1} = l + 1$. After balancing phase 1, $W_{0,2} = \frac{m+l}{2}$, $W_{1,2} = \frac{m+l}{2} + 1$ and $W_{2,2} = \frac{m+l}{2}$, $W_{3,2} = \frac{m+l}{2} + 1$. Note that $m + l$ is divisible by 2 and the difference in the number of units assigned to all the nodes is no more than 1 which is $\lceil \frac{\log 4}{2} \rceil$.
   - *Case* II: $W_{0,0} + W_{1,0} = 2m + 1$, for some even m and $W_{2,0} + W_{3,0} = 2l + 1$, for some odd l. After balancing phase 0, $W_{0,1} = m + 1$, $W_{1,1} = m$ and $W_{2,1} = l$, $W_{3,1} = l + 1$. After balancing phase 1, $W_{0,2} = \frac{m+l+1}{2}$, $W_{1,2} = \frac{m+l+1}{2}$ and $W_{2,2} = \frac{m+l+1}{2}$, $W_{3,2} = \frac{m+l+1}{2}$. Note that $m + l + 1$ is divisible by 2 and the maximum difference in the number of units assigned to all the nodes is 0.
   - *Case* III: $W_{0,0} + W_{1,0} = 2m + 1$, for some odd m and $W_{2,0} + W_{3,0} = 2l + 1$, for some even l. This can be similarly proved as in the proof for case II.

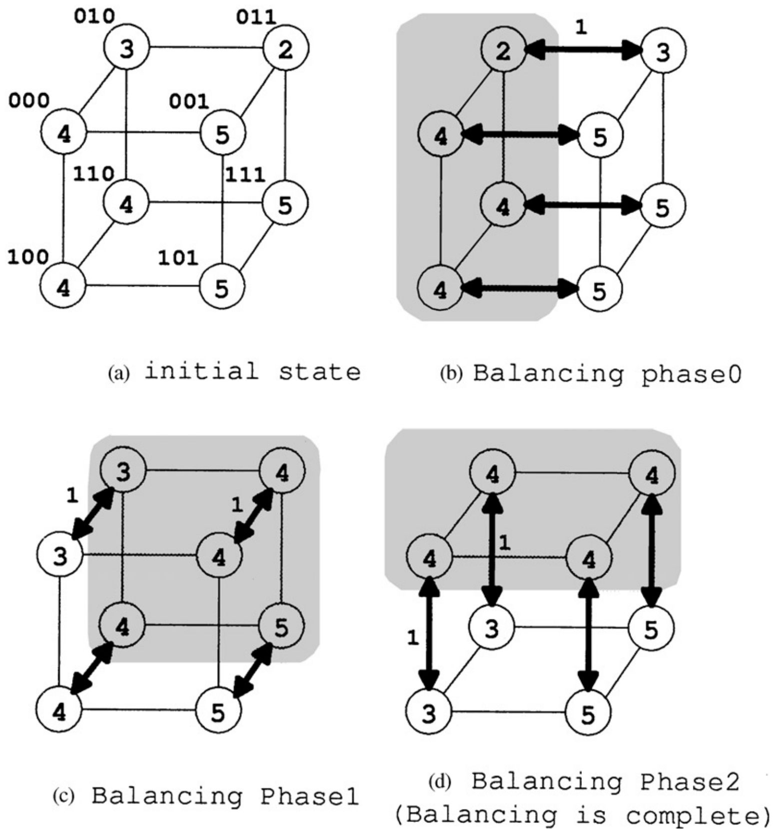


Fig. 5. Balancing of quantized loads on a hypercube of 8 processors using OEM.

- *Case* IV: $W_{0,0} + W_{1,0} = 2m + 1$, for some even m and $W_{2,0} + W_{3,0} = 2l + 1$, for some even l. After balancing phase 0, $W_{0,1} = m + 1$, $W_{1,1} = m$ and $W_{2,1} = l + 1$, $W_{3,1} = l$. After balancing phase 1, $W_{0,2} = \frac{m+l}{2} + 1$, $W_{1,2} = \frac{m+l}{2}$ and $W_{2,2} = \frac{m+l}{2} + 1$, $W_{3,2} = \frac{m+l}{2}$. Note that $m + l$ is divisible by 2 and the difference in the number of units assigned to all the nodes is no more than 1 which is $\lceil \frac{\log 4}{2} \rceil$.
- *Case* V: $W_{0,0} + W_{1,0} = 2m$, for some m and $W_{2,0} + W_{3,0} = 2l + 1$, for some l. After balancing phase 0, $W_{0,1} = m$, $W_{1,1} = m$ and $W_{2,1} = l + 1$, $W_{3,1} = l$ for even l or $W_{2,1} = l$, $W_{3,1} = l + 1$ for odd l. If $W_{0,1} + W_{2,1}$ is $m + l + 1$ then $W_{1,1} + W_{3,1}$ is $m + l$, or vice versa. Since either $m + l + 1$ or $m + l$ is divisible by 2, the maximum different in the number of assigned units of load after balancing phase 1 cannot be greater than 1.
- *Case* VI: $W_{0,0} + W_{1,0} = 2m + 1$, for some m and $W_{2,0} + W_{3,0} = 2l$, for some l. This can proved similarly as for case V.

2. (*Induction*: If it is true for $N = 2^{2k}$, then it is true for $N = 2^{2(k+1)}$.) The size of subcubes on which load balancing is complete is doubled after each balancing phase. For example, after balancing phase 1, the load balancing in each subcube of size 2^2 is complete. In two consecutive balancing phases, 4 nodes from 4 different subcubes exchange load among themselves. Let

a, b, c, and d represent the loads of the four such nodes from subcubes of size 2^{2k} before balancing among themselves is performed. Let a', b', c', and d' represent loads after the two consecutive balancing phases among themselves. Let α, β, γ, and δ represent the loads of the four nodes which belong to the same subcube as a, b, c, and d, respectively. Let α', β', γ', and δ' represent loads after the two consecutive balancing phases among themselves. After the balancing, a', b', c', d', α', β', γ', and δ' belong to the same subcube of size $2^{2(k+1)}$. Clearly we have Eqs. (1) and (2). Since a and α represent the loads from the same subcube, $a - \alpha \leqslant k$ from the above assumption that the maximum difference in the units of load in a balanced subcube of size 2^{2k} is no more than $\lceil \frac{\log 2^{2k}}{2} \rceil = k$. The same is true for b and β, and so on:

$$(a+b+c+d) - (\alpha+\beta+\gamma+\delta) \leqslant 4k, \tag{1}$$

$$(a'+b'+c'+d') - (\alpha'+\beta'+\gamma'+\delta') \leqslant 4k. \tag{2}$$

We prove by contradiction that it is impossible for any node in the subcube can have loads larger by more than $k+1$ than any other node in the subcube. Without loss of generality, assume a' and δ' represent the maximum and minimum load, respectively. Assume as a way of contradiction Eq. (3) is true.

$$a' - \delta' \geqslant k+2. \tag{3}$$

From Eqs. (2) and (3), we have Eq. (4).

$$(b'+c'+d') - (\alpha'+\beta'+\gamma') \leqslant 3k-2. \tag{4}$$

From the above proof for $N=4$ and Eq. (3), we have Eqs. (5) and (6).

$$b', c', d' \geqslant a' - 1 \geqslant \delta' + k + 1, \tag{5}$$

$$\alpha', \beta', \gamma' \leqslant \delta' + 1. \tag{6}$$

From Eqs. (5) and (6), we have equation

$$(b'+c'+d') - (\alpha'+\beta'+\gamma') \geqslant 3k. \tag{7}$$

From Eqs. (4) and (7), we have contradiction! Thus, the difference in the units of load among the nodes belong to the subcube of size $2^{2(k+1)}$ is no greater than $k+1$.

3. From the base for $N=4$ proved in 1 and the induction proved in 2, the proof follows. □

3.1. *The worst case in the OEM for quantized loads*

Even though using OEM the maximum difference in a balanced subcube is not incremented in each phase, we found that it is possible for the maximum difference to be incremented again in every other phase. Fig. 6 illustrates one worst case of load distribution before balancing. This combined with Theorem 1 leads us to the conclusion that, in the worst case, OEM will produce the maximum difference of exactly $\lceil \frac{1}{2} \log N \rceil$. Therefore, the maximum difference in our scheme is only about half of the maximum difference in the original DEM when applied to the quantized loads. In the next section, our proof is ascertained by simulation with random loads on

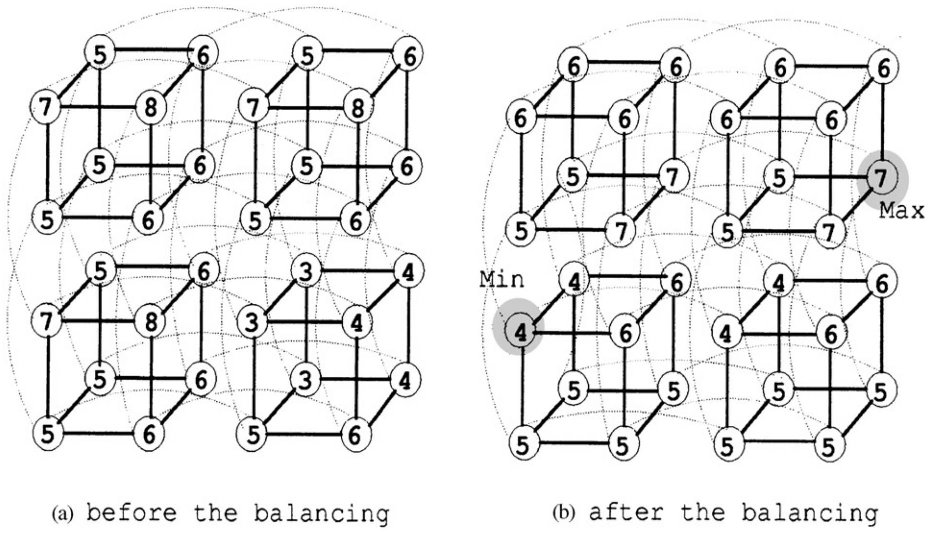


Fig. 6. Balancing of quantized loads on a hypercube of 32 processors using OEM: a worst case.

hypercubes of sizes up to 64. Further we will show how the maximum differences resulting from repeated balancing affect the total processing time. The SLAM II simulation tool is employed to show how our scheme performs better in a practical sense [1].

4. The simulation of OEM

Simulation is performed to satisfy two purposes. First is to verify Theorem 1 that the maximum difference in the worst case for OEM does not exceed $\lceil \frac{1}{2} \log N \rceil$. We enumerated all possible distribution using the numbers in the range as initial loads on processors. Since the maximum difference basically results from various combination of load types (odd or even), simulation can represent all possible cases for large range of numbers on large hypercubes. Table 1 shows the number of combination which produce certain maximum differences. As expected, OEM has 0 combinations that produce the maximum difference exceeding $\lceil \frac{1}{2} \log N \rceil$. The second and more practical purpose is to show that OEM performs better especially on quantized loads than the DEM method.

In addition to the above enumeration of all possible combinations of quantized loads for some restricted range of numbers, we performed a simulation using a discrete-event-based model. The purpose of this simulation is to estimate how the reduced maximum difference of OEM will affect the load balancing performance such as average queue length and speedup in processing time. SLAM II is used as a simulation tool [1]. Following parameters are used for simulation [6]:

- The number of processors in a hypercube: 8, 16, 32, 64.
- Architecture: loosely coupled multicomputer.
- Number of processes (λ) generated in each processor: 1000λ, 5000λ.
- Distribution of process arrival: poisson distribution.

Table 1
The number of maximum difference after balancing

	8 Processors		16 Processors		32 Processors	
	DEM	OEM	DEM	OEM	DEM	OEM
diff = 0	50,438	87,034	148,226	476,485	55,412	889,092
diff = 1	819,747	925,739	17,593,176	24,949,040	117,986,702	273,339,227
diff = 2	211,170	68,802	12,502,375	4,996,230	220,341,830	63,571,635
diff = 3	220	0	100,973	0	10,254,632	12,543,611
diff $\geqslant 4$	0	0	77,005	0	1,704,989	0

- Service time: exponential distribution.
- Run time unit: 10 runs, 1,000,000 time unit, event driven.
- Threshold for overload: 50–60% CPU utilization.
- Scheduling for loads in waiting queue: FIFO.
- No change in load distribution during balancing.

Figs. 7 and 8 show average of maximum differences after balancing for 1000λ, 5000λ, respectively. The average is taken over many balancing instances which happened during the 1,000,000 time units for the event-driven simulation. The maximum difference is directly related to the maximum of queue lengths in the hypercube multicomputer. Thus, the smaller the average is, the more uniform the load distribution is, which leads to reduced processing time. Comparison of the OEM against the DEM shows 30% improvement. Fig. 9 shows improvement of OEM in speedup over the DEM method. It shows about 30% improvement. The comparison also shows that improvement in speedup tends to grow for larger hypercube. This is as we expected. Since the processing time after balancing is $a + \log N$ for the DEM method and $a + \lceil \frac{1}{2} \log N \rceil$ for the OEM, when a is the minimum load after balancing which is assumed to take a time units to finish. The OEM will take much less time for larger N.

5. Comparison of proposed OEM method with the CWA algorithm

In this section, we are going to compare OEM with known Cube Walking Algorithm (CWA) algorithm in terms of communication overheads. A brief explanation of CWA [5] is provided for

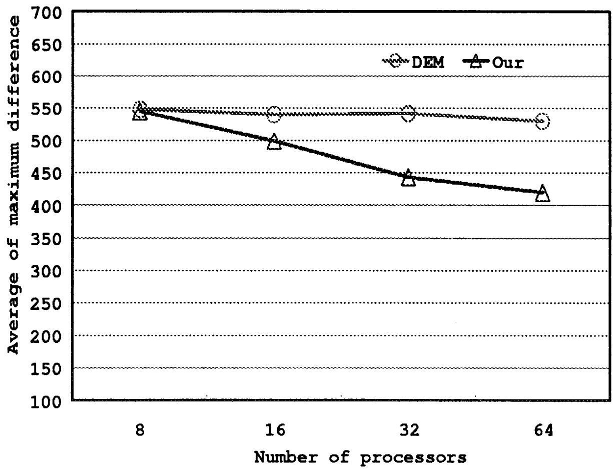


Fig. 7. Average of the maximum difference for 1000λ.

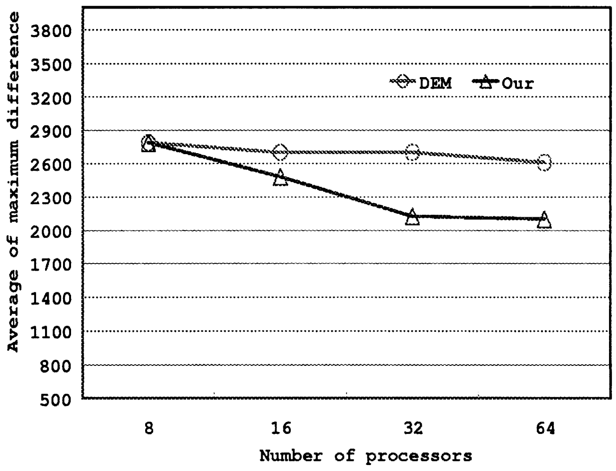


Fig. 8. Average of the maximum difference for 5000λ.

here self-containment. The CWA collects global load information from processors, computes the average load per processor and broadcast it to all processors before it starts balancing load among processors. Average load per processor is computed by dividing the total number of loads in a hypercube by the number of its processors. A processor is overloaded (underloaded) when it has more (less) loads than the average. Balancing is recursively performed between neighboring subcubes. At first, a d dimensional hypercube is divided into two $(d-1)$-dimensional subcubes of 2^{d-1} processors. Processor k sends part of its load to its counterpart processor (whose id is $k \oplus 2^{d-1}$) only if itself and the subcube it belongs to is overloaded. A subcube is overloaded (underloaded) if the total number of loads in the subcube is more (less) than the average

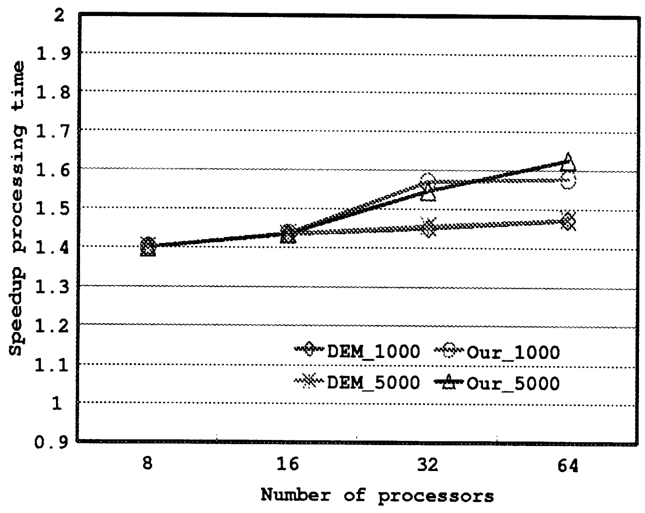


Fig. 9. Balancing effect on speedup.

multiplied by the number of processors in the subcube. Then, a $(d-1)$-dimensional subcube is divided into two $(d-2)$-dimensional subcubes of 2^{d-2} processors. Again, overloaded processors in overloaded subcubes send load to their counterparts. This continues until the size of subcubes reach 1.

5.1. *Uniform balancing*

The CWA is capable of uniform balancing after which the number of loads assigned to each processor differs by at most one. If 86 units are to be distributed among 8 processors, 2 processors have 10 units and the rest 6 processors has 11 units. In DEM and OEM, the difference can be as large as $\log N$ and $\lceil \frac{\log N}{2} \rceil$, respectively. Since the expected processing time after balancing depends on the processor with most load (assuming each processor has same available computing power), it is desirable to minimize the load difference. However, as shown in Table 1, the number of cases with large difference is small for DEM and even smaller for OEM.

5.2. *Communication overheads for transmission of loads*

In [5], *task-hop* is defined as a measure of the communication overheads for transmission of task units (or loads) in balancing phases. If T_k denotes the number of loads transmitted through the link k, $\sum_k T_k$ represents the task-hop. It is shown via simulation that the CWA outperforms the DEM in terms of average task-hop as reported in [5]. However, if we assume that execution of loads is held during balancing, some links are idle in some phases of balancing.

For example, all the links except those connecting nodes 000 and 010 or nodes 101 and 111 are idle in the balancing phase 1 of Fig. 10(c). This observation leads us to another measure of communication overheads. In each phase our new measure concerns with only the link in which the maximum number of units are transmitted. Transmission time in other links can be hidden. If

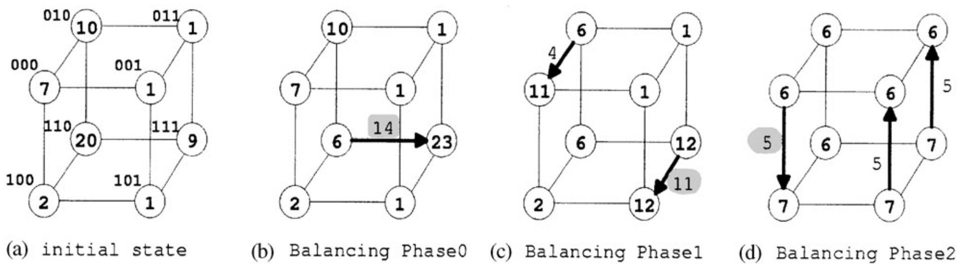


Fig. 10. A running example of CWA.

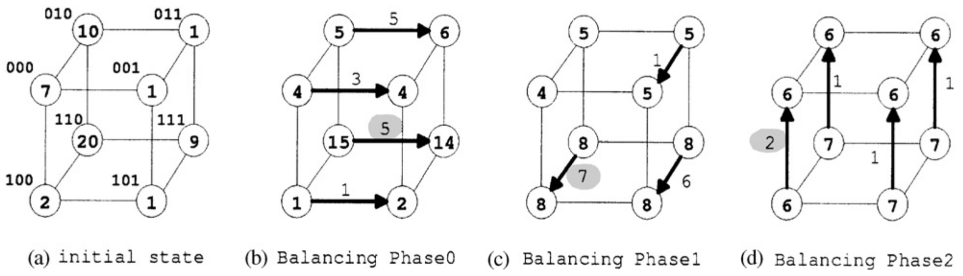


Fig. 11. A running example of OEM.

$Tmax_i$ represents the maximum transmission time in phase i, $\sum_i Tmax_i$ is our measure of communication overheads. $Tmax_1$ (in Fig. 10(c)) is 11 which is the maximum of 4 and 11. $Tmax_0$ and $Tmax_2$ are 14 and 5, respectively, which, together with $Tmax_1$, give 30 as communication overheads. In comparison, Fig. 11 shows transmission of loads using the OEM. Again $\sum_i Tmax_i$ is computed. The result is 14 which is much smaller than the transmission time for CWA. If we assume that execution is held until balancing is complete, the other links will be (most likely) idle until the link with largest units to be transmitted finished transmission. In this case, the $\sum_i Tmax_i$ provides better estimate of communication overheads than the *task-hop*. Figs. 12 and 13 show the communication overheads using this measure for the CWA and OEM balancing methods. Compared with CWA, the OEM exhibits reduced communication overheads by the margin of around 10%.

5.3. Communication overheads for collection of load distribution and broadcasting

The CWA collects global load information from processors, computes the average loads per processor and broadcast it to all processors before it starts balancing load among processors. Since both collection of load information and broadcasting the average take $\log N$ communication steps (as described in the above) for a hypercube of N processors, the communication overhead for collection of load distribution and broadcasting the average in the CWA method takes $2\log N$ phases. The OEM neither collects global load information nor broadcasts the average. Only the load information in the counterpart processor is used for balancing. In the first balancing phase, processor k sends its load to its counterpart processor $k \oplus 2^{d-1}$, if it has more

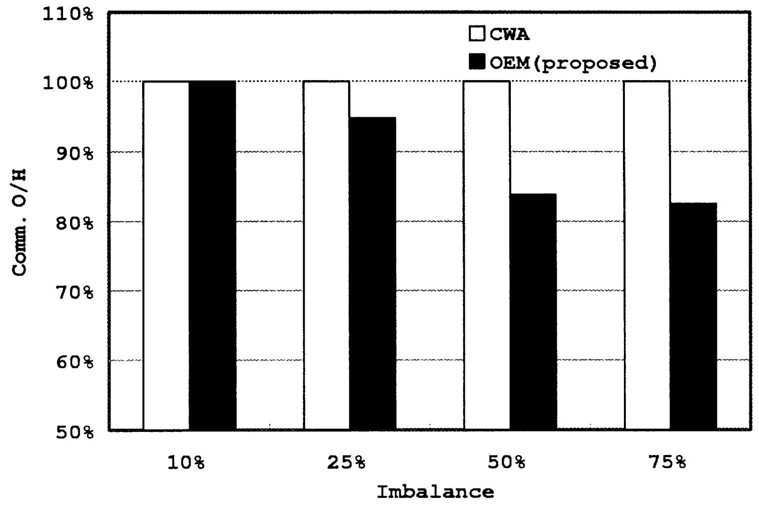


Fig. 12. Comparison of communication overheads for CWA and OEM: 8 processors.

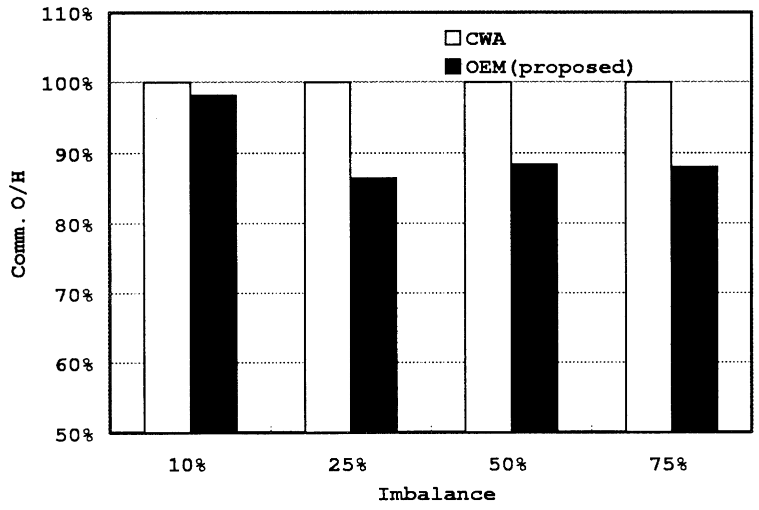


Fig. 13. Comparison of communication overheads for CWA and OEM: 16 processors.

load than the latter. One can say that CWA balances load between subcubes while OEM balances load between counterpart processors. The communication overheads for collection of load information in the OEM is nothing but one communication step.

5.4. Scalability

Uniform balancing (maximum load difference is no more than one) is possible with CWA because it utilizes global load information. The DEM and OEM uses only local load information (only the load information in its counterpart processor). The DEM and OEM methods are more scalable since collection of load information takes only $O(1)$ time while CWA takes $O(\log N)$ time for a hypercube of N processors.

5.5. Summary

Table 2 compares asymptotic behavior of our OEM against that of CWA. For arbitrarily large N, CWA excels OEM in uniformity after balancing (maximum load difference), while OEM outperforms CWA in scalability (collection of load information, computation time, storage). This trade-off between the uniformity and the scalability would lead one to design some hybrid scheme by combining OEM and CWA. For example, for a hypercube of N processors, one can employ OEM in the first $\log P$ balancing phases and apply CWA in the rest $\log(N/P)$ phases. The maximum load difference will be $\log P + 1$ and collection of load information takes $2\log(N/P)$. P can be chosen as desired by applications or processing environment.

6. Hiding balancing overheads by overlapping and pipelining

We introduce new techniques for hiding communication overheads involved in balancing. If we assume execution is being held during balancing, our techniques will speed up the processing by reducing the balancing overheads. The basic idea behind our techniques is to overlap and pipeline the transmission of load as much as possible. Assumed here is that links not involved in balancing phases are idle and all loads are independent. Efforts are made to utilize the links which would have been idle in known balancing algorithms such as DEM, CWA or OEM. Explanation of techniques will be followed by simulation results.

6.1. Overlapping

First, the loads to be transmitted on each link in all the balancing phases are determined in advance. Transmissions in different balancing phases are overlapped as long as possible. Fig. 14 illustrates transmission of loads in the original DEM method. Transmission is performed by dimension as the name dimensional exchange method implies. When balancing is performed along dimension 0 as illustrated in Fig. 14(b), only transmissions on links aligned with dimension 0 are performed. Transmission on links aligned with other dimensions are held until their turn. If we define communication overhead for transmission of loads to be $\sum_i Tmax_i$ ($Tmax_i$ is the

Table 2
Asymptotic behavior of the OEM and CWA

	OEM (proposed)	CWA
Collection of load information	1 (neighbor)	$2\log N$ phases (global)
Additional computation	1	$\log N$ each phase
Total computation time	$\log N$	$\log^2 N$
Maximum load difference	$\lceil \frac{1}{2}\log N \rceil$	0 or 1
Storage	1	$\log N$

N–number of processors.

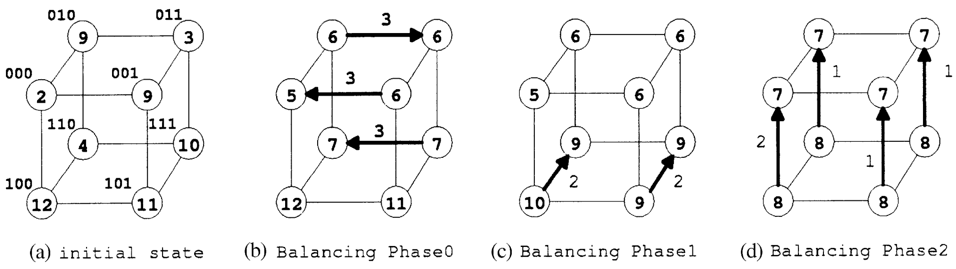


Fig. 14. A balancing example of original DEM method.

maximum transmission time in balancing phases i as explained in Section 5), it would take $7T_{\text{comm}}$. (Assume T_{comm} represents the time to move one load on a link.)

Our overlapping technique allows transmission on links aligned with different dimensions to be overlapped except when a processor involved in overlapping should delay sending out load until it has received enough. The following pseudo-code describes overlapping methods:

The overlapping algorithm

```
$Current(P_k)$ : the number of units of load in processor $P_k$
$Togo(P_k)$ : the total number of units to be transmitted from $P_k$ to adjacent processors
$Togo(P_{k,i})$ : the number of units to be transmitted from $P_k$ to $P_j$, where $j = k \oplus 2^i$
$Change(P_k)$ : Increment to the load of $P_k$ from adjacent processors
For $i = 0$ to $\log N - 1$ do
For all processors $P_k$ do in parallel
    Compute $Current(P_k)$, $Togo(P_k)$ by balancing algorithm (DEM, OEM, or CWA)
For $i = 0$ to $\log N - 1$ do
While $(\sum_k Togo(p_k) > 0)$ do
    For all processors $P_k$ do in parallel
        $Change(P_k) = 0$
        Find maximum $i$ such that $\sum_{m=0}^{i} Togo(P_{k,m}) \leqslant Current(P_k)$
        $Current(P_k) = Current(P_k) - \sum_{m=0}^{i} Togo(P_{k,m})$
        For all $m \leqslant i$ do
            $Change(P_l) = Change(P_l) + Togo(P_{k,m})$ where $l = k \oplus 2^m$
            $Togo(P_{k,m}) = 0$
    end do in parallel
    For all Processors $P_k$ do in parallel
        $Togo(P_k) = \sum_{m=0}^{\log N-1} Togo(P_{k,m})$
        $Current(P_k) = Current(P_k) + Change(P_k)$
    end do in parallel
end(While)
```

Fig. 15 shows the effect of overlapping on the same example as in Fig. 14. The effective transmission time, $\sum_i Tmax_i$, is reduced to $3T_{\text{comm}}$. For this specific example, overlapping saves

50% of transmission time. Actual saving will depend on initial distribution. Fig. 16 illustrates transmission of loads in the original CWA method. Again transmission is performed dimension by dimension leaving all the links aligned with other dimensions idle. The $\sum_i Tmax_i$ measure gives $10T_{\text{comm}}$ as transmission time and our overlapping technique reduces it to $5T_{\text{comm}}$ as shown in Fig. 17.

For overlapping to be applied to DEM or CWA, each processor should know in advance the number of loads to be sent out from itself. As for CWA, this can be done as a by product of collecting global load information taking $2\log N$ phases. It would take $\log N$ phases for DEM since it should simulate $\log N$ phases of transmission without actual movement of loads.

6.2. Overlapping and pipelining

There are some cases where the above overlapping technique is not quite effective. Consider balancing in Fig. 18 where one processor has 15 units while all other processors have only one unit of load. Fig. 18(a) shows initial load distribution and the number of units to be sent out from each processor. Here processor 001 is supposed to send 3 units to processor 011, but it cannot initiate sending since it has only one unit initially. Processor 001 should wait for 7 units from processor

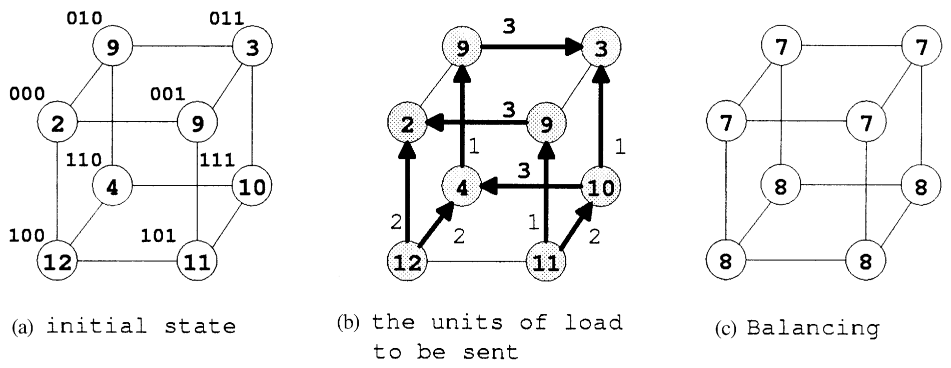


Fig. 15. A balancing example of DEM with overlapping.

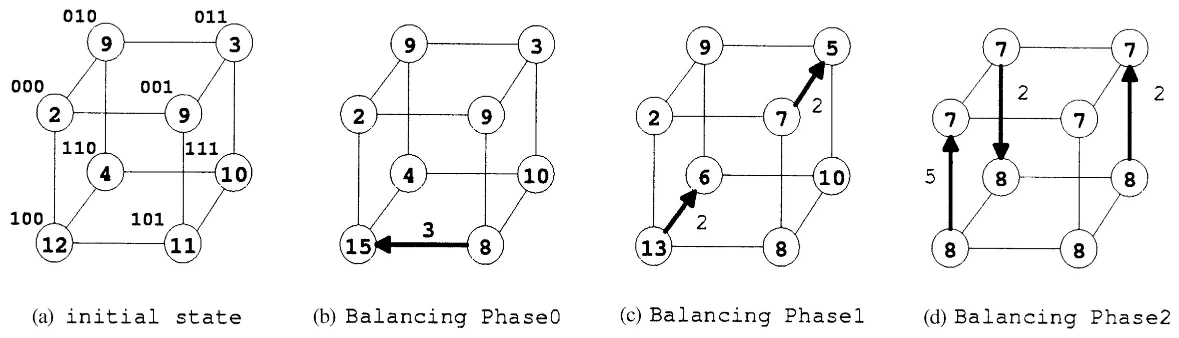


Fig. 16. A balancing example of original CWA method.

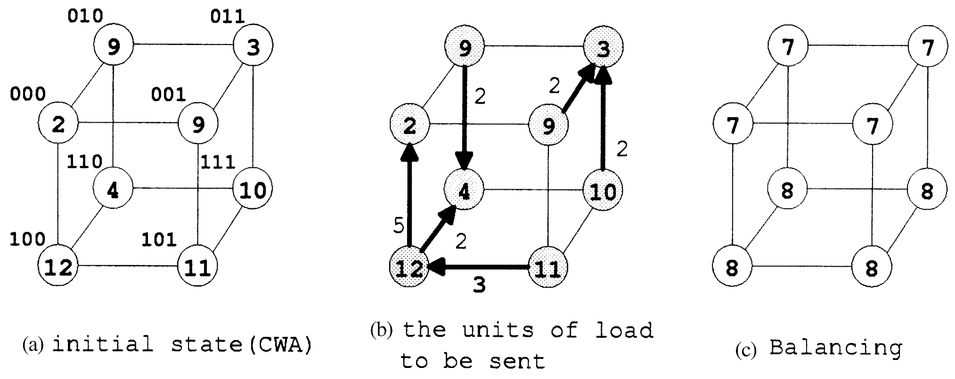


Fig. 17. A balancing example of CWA with overlapping.

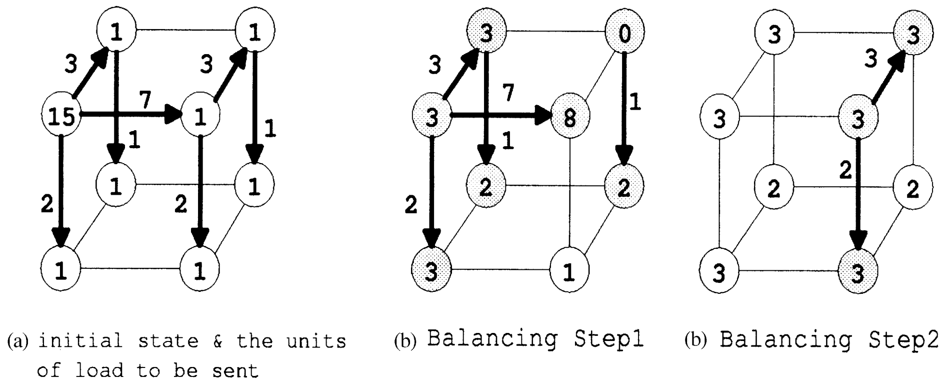


Fig. 18. An example where overlapping is not so effective.

000, which prevents overlapping. The time for transmission is $10T_{\mathrm{comm}}$. Similar situations will occur when initial load distribution is severely uneven. We introduce pipelining to achieve speedup even in the presence of severe unevenness which nullify the effect of overlapping. The rationale behind the pipelining is not to leave links idle whenever possible. A processor does not wait for whole number of units but sends out a unit of load immediately as soon as it receives one. In Fig. 19, processor 001 sends out one load of its own (labeled 0 in (b)) to processor 011 at first. In the meantime one unit (labeled 1) arrives from processor 000 and is forwarded to processor 101. Next, a load labeled 2 arrives from processor 000 and is forwarded to processor 001. Continuing in this way, balancing is complete after $7T_{\mathrm{comm}}$ with pipelining. By the time processor 000 completes sending 7 units to processor 001, all other transmission (including one from processor 0 to processor 3) would have been finished. Note that the transmission from processor 001 to 011 is concurrently performed with the transmission from processor 000 to 001, which is not the case in Fig. 18. This is made possible by forwarding a unit as soon as possible to increase the utilization of links. With pipelining, more links will be busy at a time. To increase utilization of links even further, we employ an interleaving scheme when a Processor has multiple receivers to send load. The following pseudo-code describes overlapping and pipelining method:

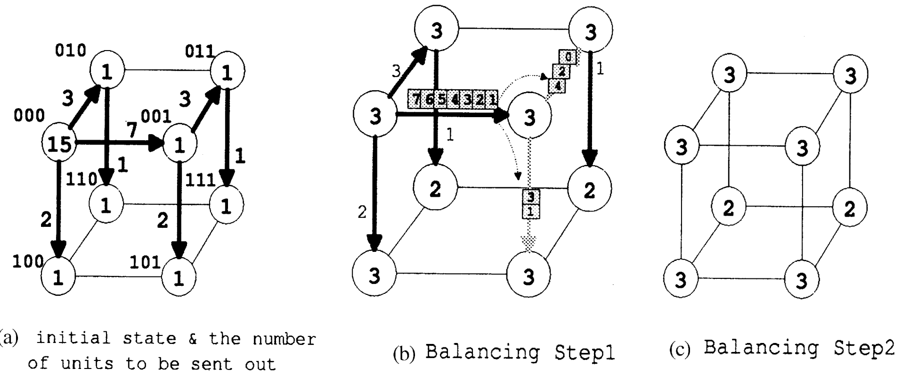


Fig. 19. An example of DEM balancing with pipelining.

The overlapping and pipelining algorithm

```
$last(P_k)$ : The index of processor to which $P_k$ sent out in the last
For $i = 0$ to $\log N - 1$ do
For all processors $P_k$ do in parallel
    Compute $Current(P_k)$, $Togo(P_{k,i})$ by balancing algorithm
$last(P_k) = -1$ for all $k$
While $\sum_k Togo(P_k) > 0$ do
        For all processors $P_k$ do in parallel
        If $Current(P_k) > 0$ then
            Find $m$ such that $Togo(P_{k,m}) > 0$ and $m > last(P_k)$
            $Current(P_k) = Current(P_k) - 1$
            $Togo(P_{k,m}) = Togo(P_{k,m}) - 1$
            $last(P_k) = m$ (if $m$ exits)
            $Change(P_l) = Change(P_l) + 1$ where $l = k \oplus 2^m$
        endif
        end do in parallel
        For = all processors $P_k$ do in parallel
            $Togo(P_k) = \sum_{m=0}^{\log N-1} Togo(P_{k,m})$
            $Current(P_k) = Current(P_k) + Change(P_k)$
        end do in parallel
end (while)
```

In Fig. 19, processor 001 should send 3 units to processor 011 and 2 units to processor 101. Instead of sending 3 units to processor 011 (with pipelining) and then initiate sending to processor 101, it interleaves sending to both receivers. It sends one unit to processor 011 and send the next unit to processor 101 and so on. In bigger hypercubes, receivers may have their own receivers and this interleaving will increase the chance of overlapping the receiver's sending to their receivers,

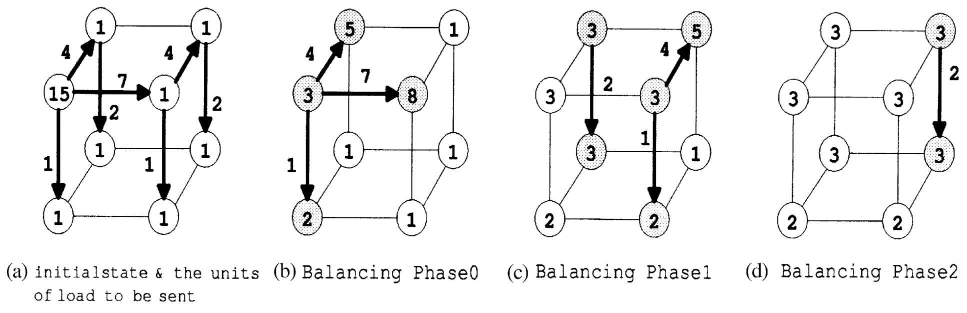


Fig. 20. An example where overlapping is not so effective.

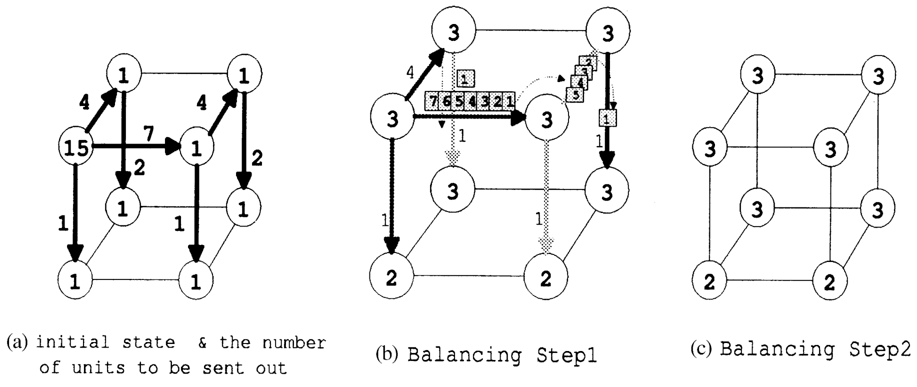


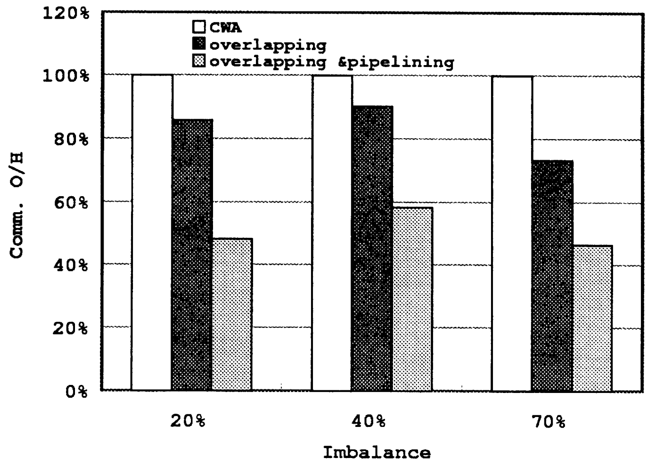
Fig. 21. An example of CWA balancing with pipelining.

thus contributing to overall speedup. If one receiver should wait it implies all successive receivers should wait and all related links should be idle.

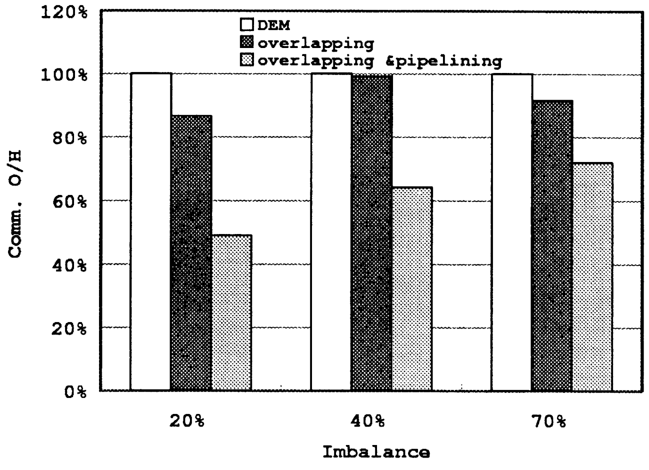
Compared with Fig. 18, pipelining in this example saves 30% in transmission time. Pipelining will not be quite effective if cost for setup or teardown of transmission is high, but it is a promising technique since efforts are being made to lighten the burden of communication setup. Fig. 21 shows the effect of pipelining applied to the example shown in Fig. 20. The transmission time is reduced from $13T_{\text{comm}}$ to $7T_{\text{comm}}$.

6.3. Simulation results: overlapping and pipelining

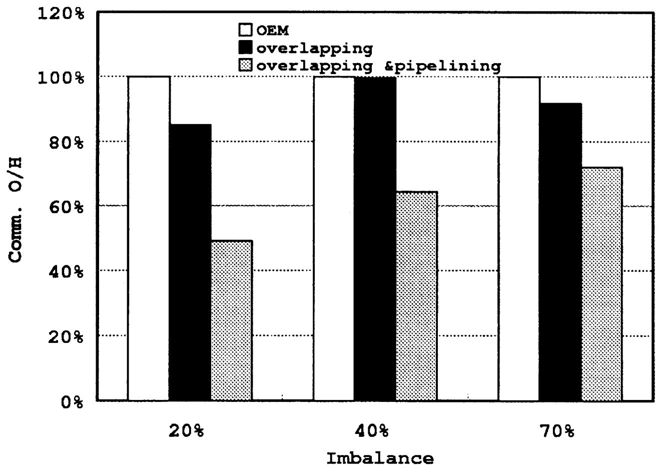
Simulation is performed to estimate the improvement obtained from proposed techniques for hiding balancing overheads, overlapping and pipelining. To quantify the effect of unevenness, we define *imbalance* as the percent of processors which have k times more load than other processors where k is any positive integer. Thus the smaller the imbalance the severer the unevenness is. In this specific simulation, we repeated simulation with k having values of 30, 50, 70, 100, 400, 700, 1000 and averaged the results to estimate the communication overhead. The communication overheads for transmission of load is normalized by the original CWA or DEM method. As seen in Fig. 22(a), for severe uneven initial load distribution (imbalance 20%), the gap between the



(a) CWA

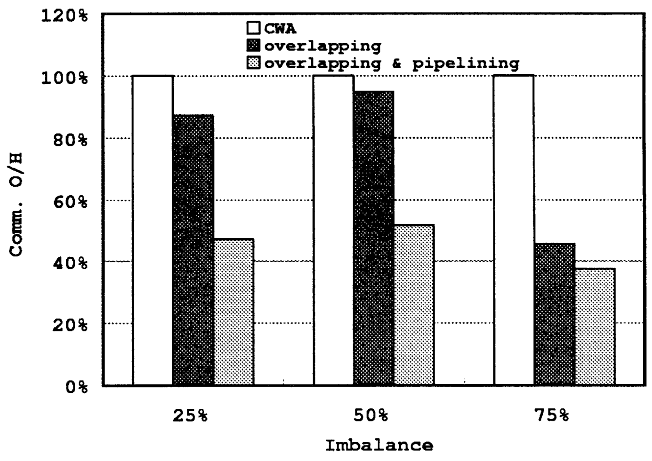


(b) DEM

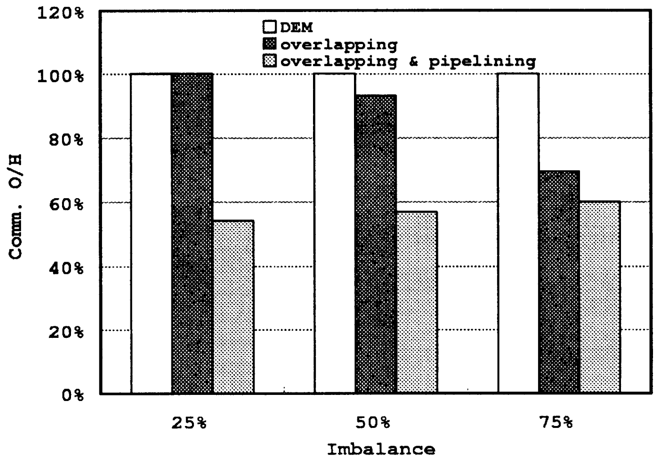


(c) OEM

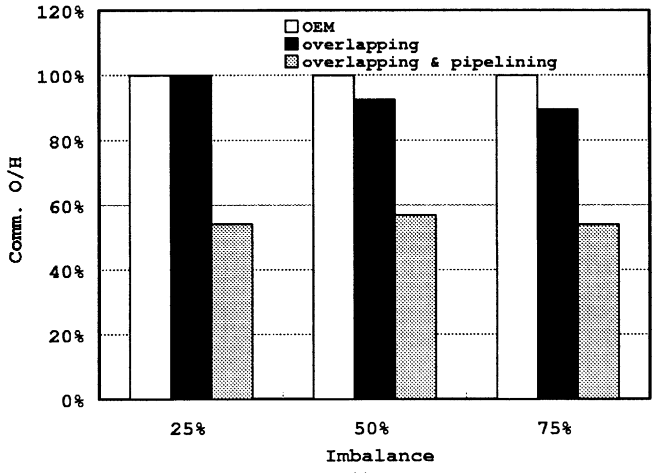
Fig. 22. Effect of overlapping and pipelining on balancing for an 8 processors hypercube with varying evenness in the initial load distribution.



(a) CWA



(b) DEM



(c) OEM

Fig. 23. Effect of overlapping and pipelining on balancing for a 16 processors hypercube with varying evenness in the initial load distribution.

original CWA and overlapping is small while the gap between overlapping and overlapping/pipelining is large. For more even (imbalance 70%) initial load distribution, the gap between the original CWA and overlapping is large while the gap between overlapping and overlapping/pipelining is small. This confirms our expectation that the improvement by overlapping is limited by severe uneven initial load distribution and it is remedied by the introduction of pipelining. Figs. 22(b) and (c) show the effect of overlapping and pipelining method on DEM and OEM for an 8 processors. Again, the pipelining has proven powerful for severe uneven distribution. Fig. 23 simulation results for a 16 processors, which reconfirms the results in Fig. 22.

7. Conclusion

A new method for dynamic load balancing on hypercube multiprocessors is proposed which performs better especially on quantized loads than the well known dimension exchange method (DEM). The maximum difference in the number of load units (quanta of load) after balancing on a hypercube of size N is reduced from $\log N$ to $\lceil \frac{1}{2} \log N \rceil$. A formal proof of correctness is provided. A simulation using the SLAM II on loosely coupled hypercube multicomputer shows about 30% improvement in speedup for processing time. We also propose new techniques for hiding transmission overhead for load exchange: overlapping and pipelining. They proved effective in making links busy, thus reducing the transmission time. It is shown via simulation that pipelining is powerful even in the presence of severe unevenness of initial load distribution which nullify the effect of overlapping.

References

[1] A. Alan, B. Pritsker, Introduction to Simulation and SLAM II, Wiley, New York, 1986.
[2] M.J. Berger, S. Bokhari, A partitioning strategy for non-uniform problems on multiprocessors, IEEE Trans. Comput. C-26 (1987) 570–580.
[3] G. Cybenko, Dynamic load balancing for distributed memory multiprocessor, J. Parallel Distrib. Comput. 7 (1989) 279–301.
[4] G. Cybenko, T.G. Allen, Parallel algorithms for classification and clustering, Proceedings of SPIE CAAASP, 1987.
[5] Min-You Wu, On runtime parallel scheduling for processor load balancing, IEEE Trans. Parallel Distrib. Systems 8 (2) (1997) 173–185.
[6] M.H. Willebeek-Lemair, Strategies for dynamic load balancing on highly parallel computers, IEEE Trans. Parallel Distrib. Systems 4 (9) (1993) 979–993.
[7] C.Z. Zu, F.C.M. Lau, Analysis of the generalized dimension exchange method for dynamic load balancing, J. Parallel Distrib. Comput. 16 (1992) 385–393.