DDE: A Modified Dimension Exchange Method for Load Balancing in k-ary n-cubes

Min-You Wu and Wei Shu
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
wu,shu@cs.buffalo.edu

Abstract— The dimension exchange method (DEM) was initially proposed as a load-balancing algorithm for the hypercube structure. It has been generalized to k-ary n-cubes. However, the k-ary n-cube algorithm must take many iterations to converge to a balanced state. In this paper, we propose a direct method to modify DEM. The new algorithm, *Direct Dimension Exchange (DDE)* method, takes load average in every dimension to eliminate unnecessary load exchange. It balances the load directly without iteratively exchanging the load. It is able to balance the load more accurately and much faster.

1. Introduction

The dimension exchange method (DEM) was initially proposed as a fully load-balancing algorithm for the hypercube structure [5, 1]. It balances the load for independent tasks on distributed memory machines. The experiment carried by Willebeek-LeMair and Reeves conformed that DEM is superior to other scheduling methods [7]. DEM for the hypercube network is a simple algorithm. Load balancing is performed iteratively in each of the $log\ N$ dimensions, in which only node pairs exchange their load information and attempt to average the number of tasks. After a *sweep* ($log\ N$ iterations), the load is balanced.

Unfortunately, when DEM applies to other structures, such as the mesh or the k-ary n-cube, it takes many sweeps to converge to the balanced load. Hosseini *et al.* extended it for arbitrary structures using the technique of edge-coloring of graphs [3]. Xu and Lau proposed the generalized dimension exchange (GDE) method [9]. The GDE method was extended to the k-ary n-cube network [10]. Because a node exchanges workload with only one of its neighbor at a time, GDE is not able to reach the balanced state in one sweep. The number of sweeps for convergence is linearly proportional to the number of nodes in a chain, and hence to the dimension order k of the k-ary n-cube structure.

We present a direct method for the k-ary n-cube, called the *Direct Dimension Exchange (DDE)* method. Unlike iterative algorithms, this direct method can balance the load in one sweep. The load in a chain is fully balanced by utilizing information of the total number of tasks, which can be easily obtained by a *sum* reduction. Each node in the chain knows whether it is overloaded or underloaded and subsequently exchange workload with other nodes. The DDE method can be applied to two or more dimensions to balance the load for the mesh, the torus, and the k-ary n-cube.

This paper is organized as follows. Section 2 briefly reviews the DEM and the GDE algorithms. Then, the direct method for the chain and the ring structures is described in sections 3 and 4, respectively. The algorithm for the k-ary n-cube is presented in section 5. In section 6, the direct method is compared to the GDE method. Section 7 concludes the paper.

2. The DEM and GDE Algorithms

The goal of load balancing is to schedule works so that each processor has the same workload. To achieve this goal, an estimation of the task execution time is needed, which can be done either by a programmer or by a compiler. Sometimes the estimation can be application-specific, and sometimes it is impossible to obtain such an estimation. Due to these difficulties, each task is presumed to require the equal execution time and the goal of the algorithm is to schedule tasks so that each processor has the same number of tasks.

The scheduling problem can be described as follows. In a parallel or distributed system, N computing nodes are connected by a given topology. Each node i has w_i tasks. A scheduling algorithm is to redistribute tasks so that the number of tasks in each node is equal. Assume the sum of w_i of all nodes can be evenly divided by N. The average number of tasks w_{avg} is calculated by

$$w_{avg} = \frac{\sum_{i=0}^{N-1} w_i}{N}.$$

Each node should have w_{avg} tasks after scheduling.

DEM was designed for the hypercube structure. In DEM, small domains are balanced first and then combined to form larger domains until ultimately the entire system is balanced. The "integer version" of DEM is described in Figure 1. All node pairs in the first dimension whose addresses differ in only the least significant bit balance the load between themselves. Next, all node pairs in the second dimension balance the load between themselves, and so forth, until each node has balanced its load with each of its neighbors.

After execution of the DEM algorithm, the load difference

$$D = \max(w_i) - \min(w_i)$$

DEM

for $l = 0$ *to* $n-1$
    node i exchanges with node j the current values of w_i and w_j, where $j = i \oplus 2^l$
    if $(w_i - w_j) > 1$, send $\lfloor (w_i - w_j)/2 \rfloor$ tasks to node j
    if $(w_j - w_i) > 1$, receive $\lfloor (w_j - w_i)/2 \rfloor$ tasks from node j
    $w_i = \begin{cases} \lceil (w_i + w_j)/2 \rceil & if \;\; w_i > w_j \\ \lfloor (w_i + w_j)/2 \rfloor & otherwise \end{cases}$

Figure 1: The DEM algorithm for the hypercube.

GDE

while (not terminate)
    for $l = 1$ *to* c
        for edge colored l connecting nodes i and j
            node i exchanges with node j the current values of w_i and w_j
            if $(w_i - w_j) > 1$, send $\lfloor (\lambda w_i - \lambda w_j) \rfloor$ tasks to node j
            if $(w_j - w_i) > 1$, receive $\lfloor (\lambda w_j - \lambda w_i) \rfloor$ tasks from node j
            $w_i = \begin{cases} \lceil (1-\lambda) \times w_i + \lambda \times w_j \rceil & if \;\; w_i > w_j \\ \lfloor (1-\lambda) \times w_i + \lambda \times w_j \rfloor & otherwise \end{cases}$

Figure 2: The GDE algorithm for the k-ary n-cube.

is bounded by n, the dimension of the hypercube [3]. The number of communication steps of the DEM algorithm is $3n$ [7].

The GDE algorithm operates on color graphs derived from edge-coloring of the given system graph. The "integer version" of the algorithm is shown in Figure 2. A node finishes a complete sweep after c consecutive exchange operations, where c is the number of colors. In k-ary n-cubes, $c = 2n$ if k is an even number. The termination condition is that the difference of the number of tasks between neighboring nodes is less than or equal to one. The convergence rate depends on the *exchange parameter* λ. The value λ varies for different topologies and different network sizes. For the hypercube, the optimal $\lambda = \frac{1}{2}$, and GDE is equivalent to the original DEM algorithm. For other topologies, λ is to be optimized to maximize the convergence rate. For the k-ary n-cube, the load difference between any pair of nodes is bounded by $nk/2$. The convergence rate decreases when the dimension order k increases. There is no communication conflict in this algorithm.

Willebeek-LeMair and Reeves suggested another approach to extend DEM to an $M \times M$ mesh topology by "folding" the mesh in each dimension $\lceil logM \rceil$ times [7]. This method could be applied to k-ary n-cubes too. The load difference is bounded by $n\lceil logk \rceil$. However, in this approach, node pairs would no longer be directly linked to one another and communications would conflict.

3. The DDE Method for the Chain

Instead of using the GDE method which balances the load iteratively, we propose a direct method. The workload in a chain can be balanced directly. The basic idea is to calculate the total number of tasks in the chain and the average number of tasks per node. Thus, nodes in the chain can exchange tasks to balance the load.

DDE-chain

Let w_i be the number of tasks in node i, where $i = 0, 1, ..., k-1$.

1. **Global Information Collection:** Perform the *scan* with *sum* operation of w_i:
$$W_i = \sum_{l=i}^{k-1} w_l$$
2. **Average Load Calculation:** $T = W_0$, $w_{avg} = \lfloor T/k \rfloor$, and $R = T \ mod \ k$, where T is the total number of tasks.
3. **Quota Calculation:** The quota of each node q_i is computed:
$$q_i = \begin{cases} w_{avg} + 1 & if \ \ i < R \\ w_{avg} & otherwise \end{cases}$$
Also, an accumulation quota for each node is computed:
$$Q_i = \sum_{l=i}^{k-1} q_l$$
4. **Flow Calculation:** $x_{i-1,i} = Q_i - W_i$, for $i = 1, 2, ..., k-1$, where $x_{i,j}$ is the flow on edge (i, j).

Figure 3: The DDE algorithm for the chain.

The DDE algorithm for the chain shown in Figure 3 is its "integer version." It takes as input the node weight $w_i (i = 0, 1, ..., k-1)$ and outputs the calculated flow $x_{i-1,i} (i = 1, 2, ..., k-1)$ for every edge in the chain. The first step is to obtain the total number of tasks in the chain by using the *scan* with *sum* operation from node $k-1$ to node 0, where k is the length of the chain. Each node records a partial sum $W_i = \sum_{l=i}^{k-1} w_l$. The second step calculates the average number of tasks per node at node 0. If the number of tasks cannot be evenly divided by k, the remaining

R tasks are distributed to the first R nodes so that they have one more task than the others. The values of w_{avg} and R are broadcast to every node. In the third step, each node calculates its quota. The accumulation quota Q_i can be calculated directly as follows:

$$Q_i = w_{avg} * (k - i) + \min(0, R - i).$$

Each node keeps records of Q_i, W_i, Q_j, and W_j, where $j = i + 1$. In the fourth step, the flow is calculated by taking difference between Q_i and W_i. Node i calculates $x_{i-1,i}$ and $x_{i,i+1}$. When the flow is available, the workload is exchanged so that each node has the same number of tasks as its quota.

Example 1:

An example is shown in Figure 4. At the beginning of scheduling, each node has w_i tasks ready to be scheduled. Values of W_i are calculated in step 1. Node 0 calculates the value of w_{avg} and R:

$$w_{avg} = 4, \; R = 5.$$

Then, each node calculates the value of Q_i in step 3. The values of w_i, W_i, Q_i, and $x_{i-1,i}$ are as shown below:

i	w_i	W_i	Q_i	$x_{i-1,i}$
0	9	37	37	–
1	7	28	32	4
2	4	21	27	6
3	1	17	22	5
4	4	16	17	1
5	6	12	12	0
6	1	6	8	2
7	5	5	4	-1

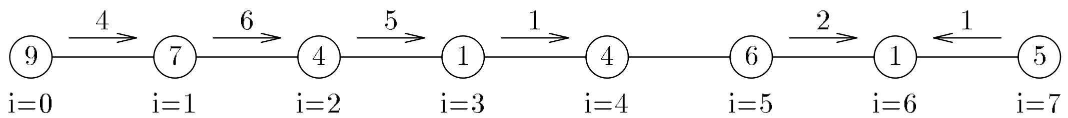


Figure 4: Example for DDE-chain.

After task exchange, nodes 0–4 have five tasks each, and nodes 5–7 have four tasks each.

Lemma 1: After execution of DDE and task exchange, the number of tasks in each node is equal to its quota.

Proof: After execution of DDE and task exchange, the number of tasks in node i is

$$w'_i = w_i + x_{i-1,i} - x_{i,i+1}$$

Because

$$x_{i-1,i} = Q_i - W_i, \quad x_{i,i+1} = Q_{i+1} - W_{i+1}, \quad W_{i+1} = W_i - w_i, \quad \text{and} \quad Q_{i+1} = Q_i - q_i$$

$$w'_i = w_i + (Q_i - W_i) - (Q_{i+1} - W_{i+1}) = Q_i - Q_{i+1} = q_i$$

□

In this algorithm, steps 1 and 2 spend $2k$ communication steps. The number of communication steps in step 4 is at most k. Therefore, the total number of communication steps of this algorithm is no more than $3k$. This algorithm can be further improved by selecting node k/2 as the root and applying the TWA algorithm in [6]. Thus, the total number of communication steps of this algorithm can be reduced to $2k$. When T is evenly divided by k, this algorithm minimizes the total number of task transfers and the total number of communications. This algorithm also maximizes locality. That is, it minimizes the number of tasks that are migrated to other nodes.

The workload is exchanged according to the flow generated by DDE. There are two task-exchange algorithms. The first one, called *receive-before-send*, is shown in Figure 5.

Receive-before-send

For node i

1. if $i > 0$ and $x_{i-1,i} > 0$, wait to receive $x_{i-1,i}$ tasks from node $i-1$
2. if $i < k-1$ and $x_{i,i+1} < 0$, wait to receive $|x_{i,i+1}|$ tasks from node $i+1$
3. if $i > 0$ and $x_{i-1,i} < 0$, send $|x_{i-1,i}|$ tasks to node $i-1$
4. if $i < k-1$ and $x_{i,i+1} > 0$, send $x_{i,i+1}$ tasks to node $i+1$

Figure 5: Task exchange: receive-before-send.

Using the receive-before-send algorithm, the load exchange in Example 1 takes four communication steps to finish:

(1) node 0 to node 1, node 5 to node 6, node 7 to node 6
(2) node 1 to node 2
(3) node 2 to node 3
(4) node 3 to node 4

Send-before-receive
For node i
let $a_i = x_{i-1,i}, \quad b_i = x_{i,i+1}$
while ($a_i \neq 0$ or $b_i \neq 0$)

1. if $i > 0$ and $(w_i > -a_i > 0)$ send $|a_i|$ tasks to node $i-1$, and let $w_i = w_i + a_i$, $a_i = 0$
2. if $i < k-1$ and $(w_i > b_i > 0)$ send b_i tasks to node $i+1$, and let $w_i = w_i - b_i$, $b_i = 0$
3. if $i > 0$ and $a_i > 0$ and received a_i tasks from node $i-1$, and let $w_i = w_i + a_i$, $a_i = 0$
4. if $i < k-1$ and $b_i < 0$ and received $|b_i|$ tasks from node $i+1$, and let $w_i = w_i - b_i$, $b_i = 0$

Figure 6: Task exchange: send-before-receive.

In the receive-before-send algorithm, each node must receive an incoming message, if any, before sending out messages. By relaxing this constraint, a *send-before-receive* algorithm is shown in Figure 6. In this algorithm, a node can start sending messages out before it has received an incoming message. The communication time and processor idle time can be reduced. It takes only two communication steps for Example 1:

1) node 0 to node 1, node 1 to node 2, node 3 to node 4,
node 5 to node 6, node 7 to node 6
2) node 2 to node 3

The send-before-receive algorithm may have some negative impact in locality. In the receive-before-send algorithm, a node can keep the maximum number of local tasks and send non-local tasks to other nodes. But in the send-before-receive algorithm, a node may send local tasks to other nodes and then receive tasks from others. Therefore, the decision on use of the receive-before-send or send-before-receive algorithms is a trade-off between communication time and locality.

Most massively parallel computers use wormhole routing with which the effect of path length on communication time can often be ignored. The recursive doubling algorithm [2] can take advantage of the pipeline effect of wormhole routing while avoiding channel contention. This algorithm organizes the nodes in a chain to a tree. An example of eight nodes is shown in Figure 7. Applying the TWA algorithm in [6] to the tree, the load can be balanced within $4\,log\,k$ communication steps.

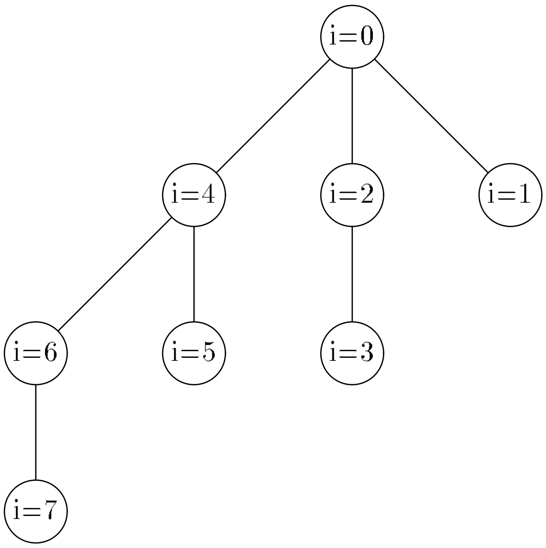


Figure 7: The tree for recursive doubling.

4. The DDE Algorithm for the Ring

A ring can be obtained by adding an end-round connection to a chain. The DDE-chain algorithm can be applied to the ring by ignoring the end-round edge. The load can be balanced, however, the communication may not be minimal. By utilizing the end-round edge, communication could be reduced. We describe an algorithm to minimize the total number of tasks transferred. The algorithm is derived from the minimum cost flow algorithm [4] and shown in Figure 8. In this algorithm, an initial solution is obtained by using DDE-chain without considering the end-round

---

DDE-ring

Apply DDE-chain to the ring without considering the end-round edge $(k-1, 0)$ to obtain $x_{0,1}, x_{1,2}, ..., x_{k-2,k-1}$, where $x_{i,j}$ is the flow on edge (i,j). Let $x_{k-1,0}$ be 0.
If the flow is clockwise, $x_{i,j}$ is positive; otherwise, it is negative.
Let n_p be the number of edges with $x_{i,j} > 0$, n_n the number of edges with $x_{i,j} < 0$, and n_z the number of edges with $x_{i,j} = 0$.

1. If $n_n + n_z - n_p < 0$, let x_m be the mth largest $x_{i,j}$ from all $x_{i,j} > 0$; and if $n_p + n_z - n_n < 0$, let x_m be the mth smallest $x_{i,j}$ from all $x_{i,j} < 0$, where $m = \lceil k/2 \rceil$.

2. For each edge, $x_{i,j} = x_{i,j} - x_m$.

Figure 8: The DDE algorithm for the ring.

edge. Then, an augmentation is applied to obtain an optimal solution. The complexity of this algorithm is $O(k \, log \, k)$.

We can use either the receive-before-send or send-before-receive algorithm for task exchange of DDE-ring. Here, we let $x_{-1,0} = x_{k-1,0}$.

The following lemma shows that this algorithm minimizes the total cost of flow, that is, the number of tasks transferred.

Lemma 2: After execution of DDE-ring, the total network flow is of minimum cost.

Proof: If $n_p + n_z - n_n \geq 0$ and $n_n + n_z - n_p \geq 0$, there is no flow augmenting cycle with negative cost. Therefore, the network flow is of minimum cost [4].

If $n_n + n_z - n_p < 0$, after modification of $x_{i,j} = x_{i,j} - x_m$, we have

$$n'_z + n'_p \geq m$$

Then,

$$n'_z + n'_p - n'_n \geq m - n'_n = m - (k - n'_z - n'_p) \geq 2m - k$$

Note that $n'_n + n'_z + n'_p = k$. Because of $m = \lceil k/2 \rceil$,

$$n'_z + n'_p - n'_n \geq 2m - k \geq 0$$

We also have

$$n'_n + n'_z \geq k - m + 1$$

Then,

$$n'_n + n'_z - n'_p \geq k - m + 1 - n'_p = k - m + 1 - (k - n'_n - n'_z) = 1 - m + n'_n + n'_z$$

$$\geq 1 - m + k - m + 1 = k - 2m + 2$$

Because of $m = \lceil k/2 \rceil$,

$$n'_z + n'_p - n'_n \geq k - 2m + 2 \geq 0$$

Thus, the network flow is of minimum cost.

The case of $n_p + n_z - n_n < 0$ can be proved similarly. Thus, the network flow is of minimum cost in all cases. □

An example is shown in Figure 9. An end-round edge is added to the chain in Figure 4 to construct a ring. Applying the DDE-chain algorithm to the ring without considering the end-round edge, the flow is shown in Figure 9(a). The number of tasks transferred is 19. The augmentation is applied to this flow:

$$n_p = 1, \quad n_z = 2, \quad n_n = 5$$

Because $n_p + n_z - n_n < 0$ and the 4th smallest $x_{i,j}$ is -2, every $x_{i,j}$ is subtracted by -2. The result is shown in Figure 9(b). The number of tasks transferred is reduced to 17.

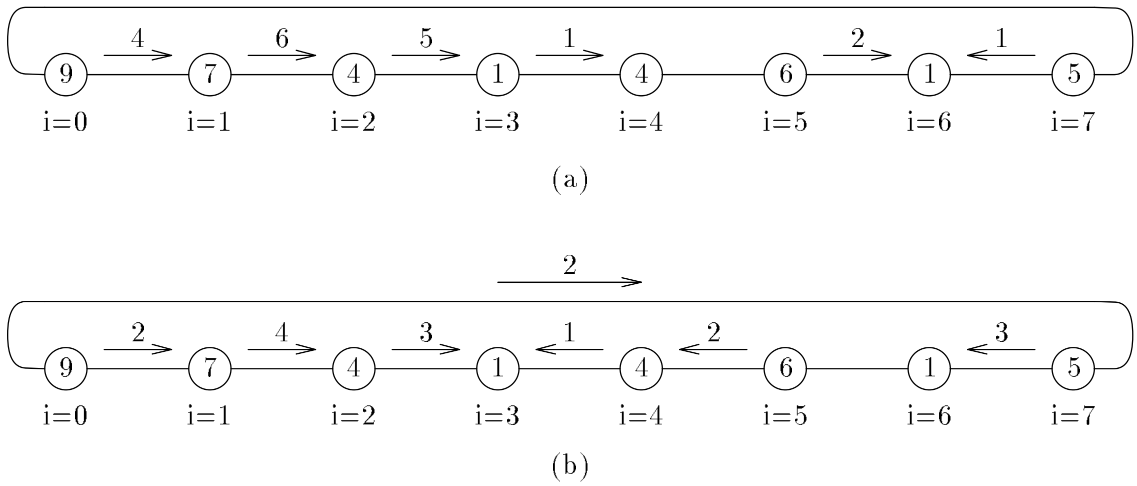


Figure 9: Example for DDE-ring.

5. The DDE Method for the k-ary n-cube

With the DDE-ring algorithm, it is not difficult to composite a DDE algorithm for the k-ary n-cube. The algorithm is shown in Figure 10. In iteration l of the DDE algorithm, subcube S_m^l is divided into k partitions S_{km+b}^{l+1} where $m = 0, 1, ..., k^l - 1$ and $b = 0, 1, ..., k-1$. S_m^l has k^{n-l} nodes. The nodes in each ring exchange their load and then each node i has w_i^{l+1} tasks. Executing DDE, node i will have w_i^n tasks. The task exchange step can use either the receive-before-send or send-before-receive algorithm.

This algorithm can be applied to the n-dimensional torus, which allows different number of nodes in different dimensions. Taking a torus and strip them of all the end-round connections, we get a mesh. This algorithm can be applied to the mesh by performing the DDE-chain algorithm instead of DDE-ring in each step.

The following theorem shows that the load difference of DDE is bounded by n, the dimension of the k-ary n-cube.

DDE for k-ary n-cube

Assume a k-ary n-cube S_0^0, the number of nodes is k^n, and node i has w_i^0 tasks.

for $l = 0$ *to* $n-1$
    apply the DDE-ring algorithm to k^{n-1} rings in the lth dimension independently,
        where every ring has k nodes $(a_0, a_1, ..., a_l, ..., a_{k-1})$ and $a_l = 0, 1, ..., k-1$
    exchange tasks according to the flow
    each node updates its weight $w_i^{l+1} = w_i^l + x_{i-1,i} - x_{i,i+1}$

Figure 10: The DDE algorithm for the k-ary n-cube.

Theorem 1: After execution of DDE, the load difference

$$D = \max(w_i^n) - \min(w_i^n)$$

is bounded by n.

Proof: In the lth step of DDE, a k-ary $(n-l)$-cube is partitioned into k k-ary $(n-l-1)$-cubes. The difference of the number of tasks between two partitions is maximal when in each ring every node in first partition, say S_{km}^{l+1}, has one more task than that possessed by the node in the other partitions, S_{km+b}^{l+1}, where $b = 1, 2, ..., k-1$. Thus

$$\sum_{j \in S_{km}^{l+1}} w_j^{l+1} = \sum_{j \in S_{km+k-1}^{l+1}} w_j^{l+1} + |S_{km}^{l+1}| = \frac{1}{k-1}(\sum_{j \in S_m^l} w_j^l - \sum_{j \in S_{km}^{l+1}} w_j^{l+1}) + k^{n-l-1}$$

where $|S_{km}^{l+1}|$ denotes the number of nodes in subcube S_{km}^{l+1} which is k^{n-l-1}. Therefore,

$$\sum_{j \in S_{km}^{l+1}} w_j^{l+1} = \frac{1}{k} \sum_{j \in S_m^l} w_j^l + (k-1)k^{n-l-2}$$

Similarly,

$$\sum_{j \in S_{km+k-1}^{l+1}} w_j^{l+1} = \sum_{j \in S_{km}^{l+1}} w_j^{l+1} - |S_{km}^{l+1}| = (\sum_{j \in S_m^l} w_j^l - (k-1) \sum_{j \in S_{km+k-1}^{l+1}} w_j^{l+1}) - k^{n-l-1}$$

Therefore,

$$\sum_{j \in S_{km+k-1}^{l+1}} w_j^{l+1} = \frac{1}{k} \sum_{j \in S_m^l} w_j^l - k^{n-l-2}$$

Let

$$A_{max}^l = \max_{0 \leq m < k^l} \sum_{j \in S_m^l} w_j^l$$

and

$$A^l_{min} = \min_{0 \le m < k^l} \sum_{j \in S^l_m} w^l_j.$$

When $l = 0$,

$$A^0_{max} = A^0_{min} = \sum_{j \in S^0_0} w^0_j = \sum_{0 \le j < k^n} w^0_j = T$$

where T is the total number of tasks. Thus,

$$A^l_{max} = \begin{cases} T & if \;\; l = 0 \\ \frac{1}{k} A^{l-1}_{max} + (k-1)k^{n-l-1} & otherwise \end{cases} \tag{1}$$

Similarly,

$$A^l_{min} = \begin{cases} T & if \;\; l = 0 \\ \frac{1}{k} A^{l-1}_{min} - k^{n-l-1} & otherwise \end{cases} \tag{2}$$

The solution to the above recurrence is given by

$$A^l_{max} = \frac{T}{k^l} + (k-1) \times l \times k^{n-l-1} \tag{3}$$

$$A^l_{min} = \frac{T}{k^l} - l \times k^{n-l-1} \tag{4}$$

It clearly satisfies (1) and (2) for the basis, $l = 0$. If (3) satisfies (1) for $l = m$, then

$$A^{m+1}_{max} = \frac{T}{k^{m+1}} + (m+1)(k-1)k^{n-(m+1)-1} = \frac{1}{k}\left(\frac{T}{k^m} + (k-1) \times m \times k^{n-m-1}\right) + (k-1)k^{n-(m+1)-1}$$

$$= \frac{1}{k} A^{(m+1)-1}_{max} + (k-1)k^{n-(m+1)-1}$$

Therefore, it satisfies (1) for $l = m+1$. Thus, by induction on l we have shown that (3) satisfies (1) whenever $l \ge 0$. Similarly, it can be shown that (4) satisfies (2) whenever $l \ge 0$.

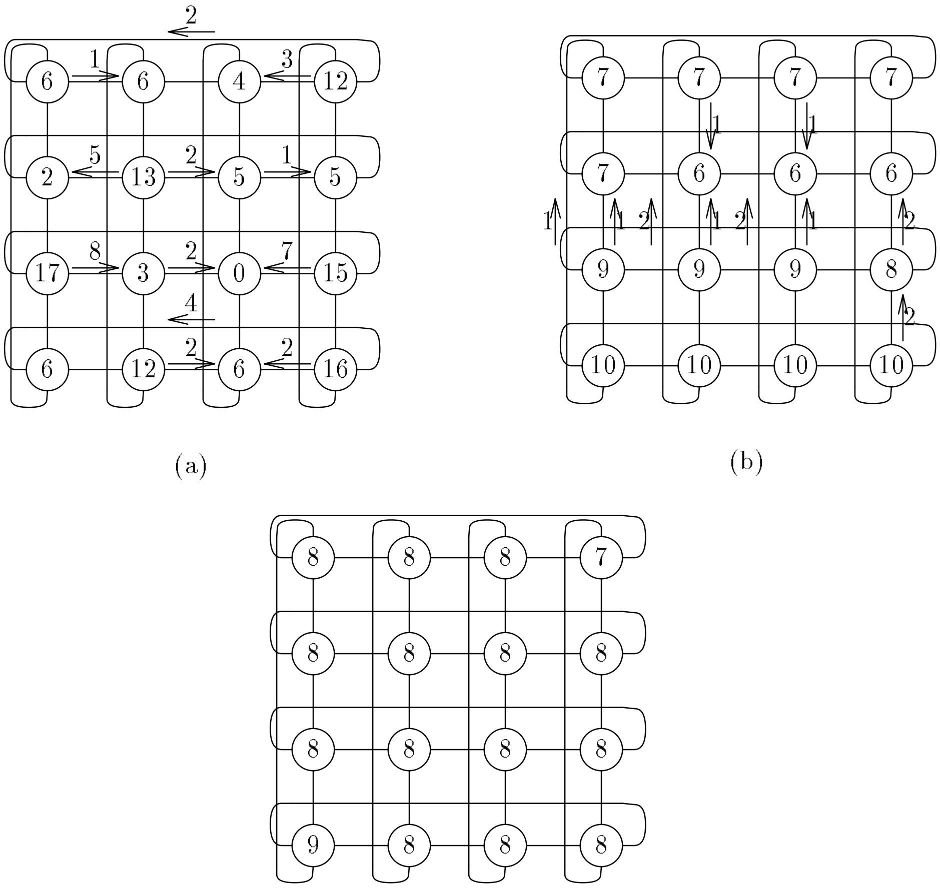
Let $l = n$

$$A^n_{max} = \max_{0 \le j < k^n} w^n_j = \frac{T}{k^n} + \frac{k-1}{k} \times n$$

$$A^n_{min} = \min_{0 \le j < k^n} w^n_j = \frac{T}{k^n} - \frac{1}{k} \times n,$$

Because $D = A^n_{max} - A^n_{min} = (\frac{k-1}{k} + \frac{1}{k}) \times n = n$, the number of tasks in any two processors differs at most by n. $\square$

An example is shown in Figure 11. This is a 4-ary 2-cube (i.e., torus 4×4). Figure 11(a) shows that the DDE-ring algorithm applies to each ring in the first dimension. Then, DDE-ring applies to each ring in the second dimension, as shown in Figure 11(b). The resultant load distribution is shown in Figure 11(c). The maximum load difference is 2.



(a) (b)

(c)

Figure 11: Example for DDE (4-ary 2-cube).

6. Experimental Results

In this section, we compare performance of GDE and DDE. We consider a test set of load distributions, in which the load at each processor is randomly selected with the mean equal to a specified value. In this simulation experiment, the average number of tasks (average weight) per processor is 1,000. Each result is the average of 100 test cases. We tested an 8×8 mesh, a 16×16 torus, an 8×8×8 3D-mesh, and a 16×16×16 3D-torus. For these networks, the optimal value of λ for GDE is 0.723 [10].

First, we compare load imbalance of GDE and DDE. The load difference of DDE is bounded by n, whereas that of GDE is bounded by $n(k-1)$ for the mesh and $nk/2$ for the torus. Figure 12 shows its average in different networks. Here, the load difference of GDE is four to six times larger than that of DDE.

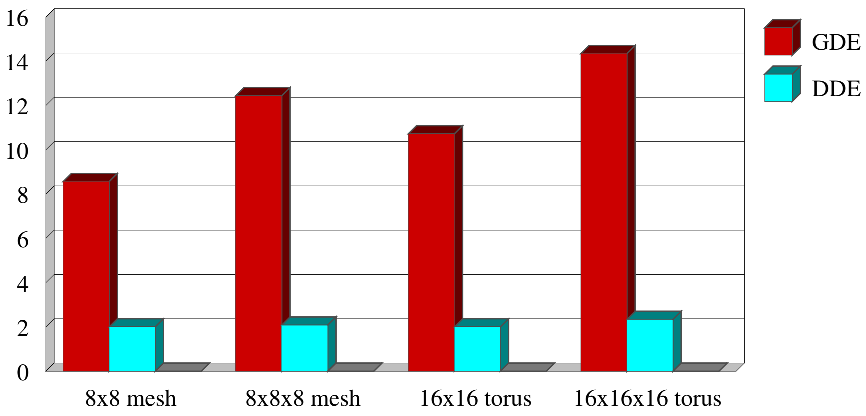


Figure 12: Load difference.

DDE completes load balancing in one sweep but GDE needs many sweeps. Figure 13 shows the number of sweeps s for different networks. The value of s is proportional to k [10]. Moreover, s increases with the average weight. Table I shows the relationship between the number of sweeps and the average number of tasks, measured on an 8×8 mesh.

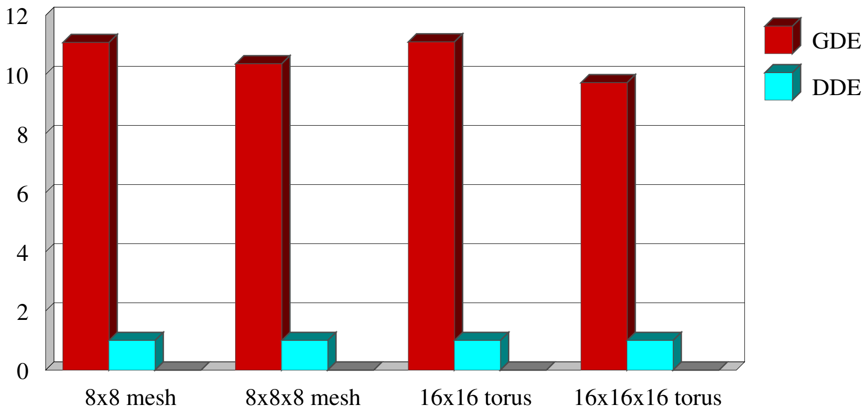


Figure 13: The number of sweeps.

Next, we compare the number of communication steps of GDE and DDE. For GDE, each sweep has c iterations, where c is the number of colors. For even number of k, $c = 2n$. Each iteration has three communications, two for exchanging load information and one for load balancing. Therefore, the total number of communications of s sweeps are $3sc = 6sn$. For DDE, there are k communication steps in each dimension for collection and broadcasting of load information. Load balancing needs at most $k-1$ and $k/2$ communication steps for the mesh and the torus, respectively. Therefore, $2kn$ or $\frac{3}{2}kn$ communication steps in total are required. DDE can reduce the number of communication steps significantly. The analysis has been confirmed by the experiment, as shown in Figure 14.

Table I: The Relationship Between the Number of Sweeps and the Average Weight

Average Number of Tasks	100	300	1,000	3,000	10,000
Average Number of Sweeps	7.28	9.20	11.08	13.02	14.67

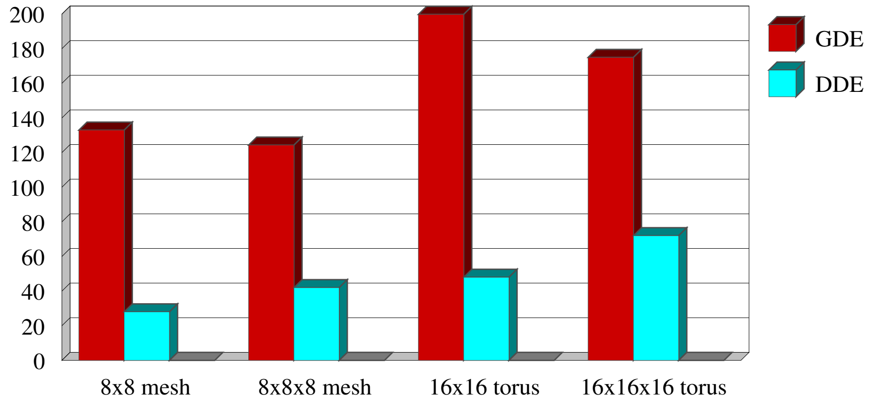


Figure 14: The number of communication steps

Figure 15 shows the normalized communication cost of GDE and DDE. The normalized communication cost is defined as the the total numbers of tasks transferred divided by the total number of tasks:

$$\frac{\sum_j e_j}{\sum_i w_i},$$

where e_j is the number of tasks transmitted through the edge j. The communication cost of GDE is about 50% larger than that of DDE. It is due to the fact that GDE transfers tasks unnecessarily. Finally, DDE has better locality than GDE. Figure 16 shows the percentage of local tasks that are not migrated to other nodes. DDE keeps 20% to 50% more tasks in local.

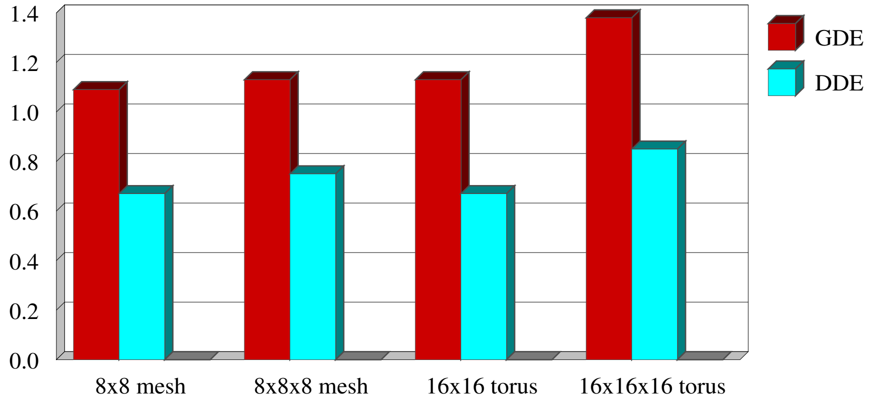


Figure 15: Normalized communication cost.

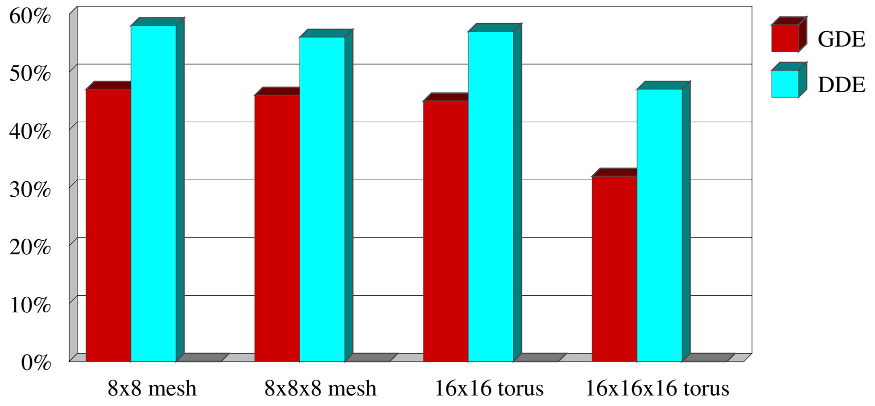


Figure 16: The percentage of local tasks.

7. Conclusion

This paper proposed a direct method for load balancing. It extended the DEM algorithm to the k-ary n-cube. Compared to the GDE algorithm, which also extended DEM to the k-ary n-cube, DDE is faster, balances the load well, reduces communications, and keeps better locality.

DDE can be further improved for a more balanced load and less communications by extending the Mesh Walking Algorithm [8]. However, DDE retains its simplicity of implementation and can deliver a satisfied performance at the same time.

References

[1] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. of Parallel Distrib. Comput.*, 7:279–301, 1989.

[2] M. Barnett et al. Broadcasting on meshes with wormhole routing. Technical Report TR-93-24, Univ. Texas at Austin, 1993.

[3] S.H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, 10:160–166, 1990.

[4] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[5] S. Ranka, Y. Won, and S. Sahni. Programming a hypercube multicomputer. *IEEE Software*, pages 69–77, September 1988.

[6] W. Shu and M.Y. Wu. Runtime parallel scheduling for distributed memory computers. In *Int'l Conf. on Parallel Processing*, pages II. 143–150, August 1995.

[7] Marc Willebeek-LeMair and Anthony P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel and Distributed System*, 9(4):979–993, September 1993.

[8] M.Y. Wu and W. Shu. High-performance incremental scheduling on massively parallel computers — a global approach. In *Supercomputing '95*, December 1995.

[9] C. Z. Xu and F. C. M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, December 1992.

[10] C. Z. Xu and F. C. M. Lau. The generalized dimension exchange method for load balancing in k-ary n-cubes and variants. *Journal of Parallel and Distributed Computing*, 24(1):72–85, January 1995.